ABSTRACT

This user's manual provides updated documentation and computer program listings for version 3.0 of the three-dimensional cement hydration and microstructure development model (**CEMHYD3D**) developed at the National Institute of Standards and Technology. Version 2.0 of **CEMHYD3D** was released in 2000 and since that time, numerous enhancements and improvements have been made to the programs. The online database of available cements has nearly doubled in size from 14 different cements to 27. Two significant updates have been made to the codes used to create starting three-dimensional microstructures: 1) the incorporation of particles of slag, inert fillers, calcium carbonate, etc. into the particle placement program and 2) streamlining of the chemical phase distribution process. Formerly, the user needed to execute three different programs, three times each to distribute the appropriate volumes (and surface areas) of the four major cement clinker phases throughout the cement particles in the initial microstructure. In version 3.0, these three programs have been combined into a single program and all ancillary calculations have been incorporated into the code so that minimal effort is required of the user. Most of the improvements and enhancements to **CEMHYD3D** for version 3.0 have been made in the programs simulating the hydration and microstructure development. These include: addition of the influences of limestone (and inert fillers) on hydration, incorporation of preliminary reactions for slags, prediction of the concentration of the pore solution during hydration and its concurrent influence on hydration rates, the ability to execute hydration under sealed/saturated conditions by specifying a number of cycles after which resaturation of the (empty) capillary porosity occurs, the ability to precipitate the C-S-H gel in either a random or a "plate" morphology, and the addition of a one-pixel dissolution bias that allows for the acceleration/retardation of the hydration rates of the smallest cement particles in the three-dimensional microstructure. Several example applications utilizing version 3.0 of **CEMHYD3D** are presented. Complete program listings are provided in the appendices. This manual and all of the computer programs it describes are freely available for downloading via anonymous ftp from NIST (ftp.nist.gov).

Keywords: Building technology; cement hydration; computer modeling; microstructure; simulation.

Contents

List of Figures

List of Tables

1 Introduction

This user's manual provides documentation, supporting data, and example applications for version 3.0 of the CEMHYD3D computer programs. Potential users of version 3.0 will benefit greatly from also reading the user's manuals for the previous two versions of CEMHYD3D, released in 1997 [1] and 2000 [2], respectively. In additional to being provided in the Appendices, each of the computer programs described in this manual is available for free downloading. The programs, written in the C programming language, may be accessed in the pub/bfrl/bentz/CEMHYD3D/version30 subdirectory from ftp.nist.gov (129.6.13.25), by logging in as user "anonymous" and providing your e-mail address as the password. This user's manual is also available as part of an electronic monograph available at http://ciks.cbt.nist.gov/monograph.

Several of the enhancements to CEMHYD3D v3.0 were made during Phase I of the Virtual Cement and Concrete Testing Laboratory (VCCTL) consortium. Phase II of this NIST/industry consortium is currently active and interested parties may contact the consortium manager, Dr. Edward J. Garboczi (e-mail: edward.garboczi@nist.gov) for more information.

DISCLAIMER

The U.S. Department of Commerce makes no warranty, expressed or implied, to users of CEMHYD3D and associated computer programs, and accepts no responsibility for its use. Users of CEMHYD3D assume sole responsibility under Federal law for determining the appropriateness of its use in any particular application; for any conclusions drawn from the results of its use; and for any actions taken or not taken as a result of analyses performed using these tools.

Users are warned that CEMHYD3D is intended for use only by those competent in the field of cement-based materials and is intended only to supplement the informed judgment of the qualified user. The software package is a computer model which may or may not have predictive value when applied to a specific set of factual circumstances. Lack of accurate predictions by the model could lead to erroneous conclusions with regard to materials selection and design. All results should be evaluated by an informed user.

INTENT AND USE

The algorithms, procedures, and computer programs described in this report constitute a methodology for modeling the microstructural development and performance properties of cement-based materials. They have been compiled from the best knowledge and understanding currently available, but have important limitations that must be understood and considered by the user. The program is intended for use by persons competent in the field of cement-based materials and with some familiarity with computers. It is intended as an aid in the materials selection, optimization, and design process.

2 Two-Dimensional Imaging of Cement Particles

Version 1.0 of CEMHYD3D included three computer programs (**statsimp.c**, **corrcalc.c**, and **corrxy2r.c**) for analyzing a segmented image of a two-dimensional cement (powder) microstructure to obtain the necessary correlation files for the two-dimensional to three-dimensional conversion of cement particles to be described subsequently [1]. These programs have not changed and are still available for downloading from the ftp site provided in the Introduction. For version 2.0 of CEMHYD3D [2], an online database of cement images was created that contains representative segmented images, along with the necessary particle size distribution (PSD) and phase correlation files for creating three-dimensional microstructures. The database contains links to the ftp locations from where the PSD and correlation files can be downloaded (available directly at ftp://ftp.nist.gov/pub/bfrl/bentz/CEMHYD3D/images). The complete database is available at http://ciks.cbt.nist.gov/bentz/phpct/database/images. Between the release of version 2.0 in 2000 and the issue of the current release, the database has nearly doubled in size, going from 14 to 27 different cements, of varying chemical composition and PSD. An example image for Cement and Concrete Reference Laboratory (CCRL) proficiency cement sample 152 is provided in Figure 1.

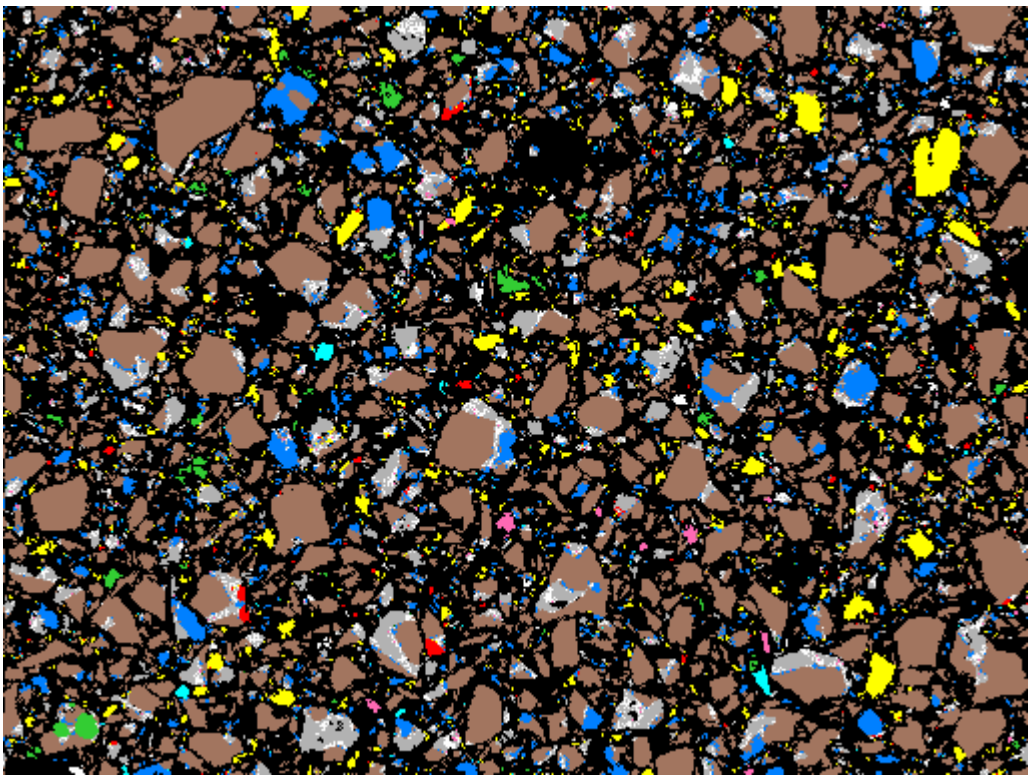

Figure 1: Two-dimensional processed scanning electron microscopy (SEM)/X-ray image for cement 152 issued by the CCRL in January 2004. Color assignments are: brown- C_3S[1], blue- C_2S, grey- C_3A, white- C_4AF, yellow- gypsum, red- K_2SO_4, green- free lime and/or calcite, aqua- free silica, and pink- periclase (magnesium containing phase). Image is 256 μm x 200 μm.

[1] Conventional cement chemistry is used in this report: C=CaO, S=SiO_2, A=Al_2O_3, F=Fe_2O_3, M=MgO,$\bar{S}$=SO_3 and H=H_2O.

3 Three-Dimensional Image Creation and Phase Distribution

3.1 Generation of Spherical Particles Following Measured PSD

The program **genpartnew.c**, whose listing is provided in Appendix A, is used to place digitized spherical particles of a user specified PSD into a three-dimensional computational volume, typically 100 elements on a side. Periodic boundaries are employed such that a particle that extends outward through one or more faces of the 3-D microstructure is completed extending inward through the opposite face(s). Additionally, the particles can be dispersed by specifying a minimum particle to particle separation distance or flocculated by moving all particles at random and flocculating those that impact one another. The major changes from version 2.0 to version 3.0 of CEMHYD3D for the **genpartnew.c** program have been the incorporation of additional particle types (limestone, slag, and dicalcium silicate) and implementation of a revised distribution algorithm for the various calcium sulfate phases (gypsum, hemihydrate, and anhydrite) that more closely matches the user-requested distribution.

The **genpartnew.c** program is menu-driven and detailed descriptions of the menu selections, taken from the version 2.0 manual [2] and repeated here for completeness, are as follows:

1) **Exit**: exit the program
2) **Add spherical particles (cement, gypsum, pozzolans, etc.) to microstructure**: allows the user to specify the discretized particle size distribution that should be used, the number of particles to place, the proportion of "cement" particles that should be calcium sulfate (gypsum, hemihydrate, and/or anhydrite), and whether the particles should be dispersed during placement. The PSD should be provided on a number basis. A useful tool for computing the number-based PSD for various cements can be found at http://ciks.cbt.nist.gov/cgi-bin/vcctl/vcctl11/gpf/listcementpsd.pl. Particles are typically at least 3 pixels (voxels[2]) in diameter and are added in order from largest to smallest. One-pixel monophase particles are added in the hydration program **disrealnew**, just prior to executing the hydration simulation (see section 5.2).
3) **Flocculate system by reducing number of particle clusters**: allows the user to create any desired number of flocs (clusters) by randomly moving each particle (cluster) centroid in one-pixel increments and aggregating any particles (clusters) that contact one another during this process. The only input required is the user requested number of clusters (flocs) to be present at the end of the execution of the routine. Typically, if no superplasticizer or water-reducing agent is used, the cement particles will have a large tendency to flocculate together, perhaps even into a single floc structure.
4) **Measure global phase fractions**: outputs the numbers of voxels of each phase currently present in the 3-D microstructure.
5) **Add an aggregate to the microstructure**: allows the user to add a single flat plate aggregate to the 3-D microstructure for studying the development of microstructure in the interfacial transition zone (ITZ). The only required input is the thickness of the aggregate to be placed, which must be an even integer. Typically, the user should place an aggregate (if desired) into

[2] The reader should note that the terms pixel and voxel are used interchangeably throughout this document. A voxel is basically just a three-dimensional pixel (pixel being a short form of picture element).

the microstructure before placing any of the cement or other particles. Otherwise, the aggregate will simply overlap and replace any solid particle pixels contained within its boundaries.

6) **Measure single phase connectivity (pores or solids)**: allows the user to employ a burning algorithm [3] to determine the percolation characteristics of either the porosity or the solids present in the 3-D microstructure. The user must specify the phase in which they are interested and the routine returns the number of voxels of that phase which are accessible from the top of the 3-D microstructure, along with the number of voxels that are contained in pathways that traverse (span) the microstructure.
7) **Measure phase fractions vs. distance from aggregate**: outputs a listing of the number of voxels of each phase present in parallel planes at various fixed distances (one pixel increments) from the aggregate surface. When this menu selection is executed, the results are reported to the standard output (screen) and are also written to a datafile named **agglist.out**. If the user wishes to preserve this output file, they must rename it to a name of their choosing immediately upon leaving the **genpartnew** program.
8) **Output microstructure to file**: allows the user to save the created microstructure to files. The user must supply two filenames, one for the storage of the actual microstructure (cement, calcium sulfate, porosity, etc.) and the second for the storage of the individual particle IDs (to be used in assessing the setting of the cement during hydration using the **disrealnew** program).

When adding calcium sulfate to the cement, the user has two choices. If only the composite PSD for the cement and gypsum (sulfate) is known, the user can simply specify the volume fraction of particles that should be randomly assigned to be calcium sulfate (in its three forms of gypsum, hemihydrate, and anhydrite). Conversely, if the actual separate PSD of the calcium sulfate is known, the user can use a value of 0.0 for the randomly-assigned volume fraction and specify the actual numbers of each size calcium sulfate particle to be placed, in a manner analogous to that used for cement. When the separate cement and calcium sulfate PSDs are "merged" in this fashion, the user should be sure to add all the largest radius particles (e.g., first the cement, then the calcium sulfate) first before proceeding to the next smaller radius particles, etc. If this second option is to be utilized, in the program **genpartnew**, a phase ID of 1 corresponds to cement, 5 to gypsum, 6 to hemihydrate, and 7 to anhydrite, as indicated in the program listing provided in Appendix A.

An example annotated datafile for using **genpartnew** is as follows:

```
-3034                   random number seed
2                       menu choice to place particles
16                      number of size classes to place
0                       dispersion distance between particles in pixels
0.0604                  calcium sulfate (total) volume fraction
0.515 0.041             fractions of calcium sulfate that are hemihydrate and anhydrite
1                       number of spheres of size class 1
17                      radius of spheres of size class 1
1                       phase ID of spheres of size class 1 (1 = cement)
1                       number of spheres of size class 2
15                      radius of spheres of size class 2
1                       phase ID of spheres of size class 2 (1 = cement)
1                       number of spheres of size class 3
```

14	radius of spheres of size class 3
1	phase ID of spheres of size class 3 (1 = cement)
1	number of spheres of size class 4
13	radius of spheres of size class 4
1	phase ID of spheres of size class 4 (1 = cement)
2	number of spheres of size class 5
12	radius of spheres of size class 5
1	phase ID of spheres of size class 5 (1 = cement)
2	number of spheres of size class 6
11	radius of spheres of size class 6
1	phase ID of spheres of size class 6 (1 = cement)
4	number of spheres of size class 7
10	radius of spheres of size class 7
1	phase ID of spheres of size class 7 (1 = cement)
5	number of spheres of size class 8
9	radius of spheres of size class 8
1	phase ID of spheres of size class 8 (1 = cement)
8	number of spheres of size class 9
8	radius of spheres of size class 9
1	phase ID of spheres of size class 9 (1 = cement)
13	number of spheres of size class 10
7	radius of spheres of size class 10
1	phase ID of spheres of size class 10 (1 = cement)
21	number of spheres of size class 11
6	radius of spheres of size class 11
1	phase ID of spheres of size class 11 (1 = cement)
38	number of spheres of size class 12
5	radius of spheres of size class 12
1	phase ID of spheres of size class 12 (1 = cement)
73	number of spheres of size class 13
4	radius of spheres of size class 13
1	phase ID of spheres of size class 13 (1 = cement)
174	number of spheres of size class 14
3	radius of spheres of size class 14
1	phase ID of spheres of size class 14 (1 = cement)
450	number of spheres of size class 15
2	radius of spheres of size class 15
1	phase ID of spheres of size class 15 (1 = cement)
2674	number of spheres of size class 16
1	radius of spheres of size class 16
1	phase ID of spheres of size class 16 (1 = cement)
4	menu selection to report phase counts
3	menu selection to flocculate particles
1	number of separate flocs (particle clusters) to create
8	menu selection to output current microstructure to file
cem152w04floc.img	filename to save image to
pcem152w04floc.img	filename to save particle IDs to
1	menu selection to exit program (end)

The output created by executing the program **genpartnew** with the above input datafile (using a command such as ***genpartnew <genpartnew.dat >genpartnew.out*** at the command line prompt of a Linux or UNIX-based system) is as follows (user inputs are in boldface):

Enter random number seed value (a negative integer)
-3034

Input User Choice
1) Exit
2) Add spherical particles (cement, gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
2
Enter number of different size spheres to use(max. is 200)
16
Enter dispersion factor (separation distance in pixels) for spheres (0-2)
0 corresponds to totally random placement
1
Enter probability for gypsum particles on a random particle basis (0.0-1.0)
0.060400
Enter probabilities for hemihydrate and anhydrite forms of gypsum (0.0-1.0)
0.515000 0.041000
Enter number, radius, and phase ID for each sphere class (largest radius 1st)
Phases are 1- Cement and (random) calcium sulfate, 2- C2S, 5- Gypsum, 6- hemihydrate 7- anhydrite
8- Pozzolanic, 9- Inert, 10- Slag, 26- CaCO3 30- Fly Ash
Enter number of spheres of class 1
1
Enter radius of spheres of class 1
(Integer <=33 please)
17
Enter phase of spheres of class 1
1
Enter number of spheres of class 2
1
Enter radius of spheres of class 2
(Integer <=33 please)
15
Enter phase of spheres of class 2
1
Enter number of spheres of class 3
1
Enter radius of spheres of class 3
(Integer <=33 please)
14
Enter phase of spheres of class 3
1

Enter number of spheres of class 4
1
Enter radius of spheres of class 4
(Integer <=33 please)
13
Enter phase of spheres of class 4
1
Enter number of spheres of class 5
2
Enter radius of spheres of class 5
(Integer <=33 please)
12
Enter phase of spheres of class 5
1
Enter number of spheres of class 6
2
Enter radius of spheres of class 6
(Integer <=33 please)
11
Enter phase of spheres of class 6
1
Enter number of spheres of class 7
4
Enter radius of spheres of class 7
(Integer <=33 please)
10
Enter phase of spheres of class 7
1
Enter number of spheres of class 8
5
Enter radius of spheres of class 8
(Integer <=33 please)
9
Enter phase of spheres of class 8
1
Enter number of spheres of class 9
8
Enter radius of spheres of class 9
(Integer <=33 please)
8
Enter phase of spheres of class 9
1
Enter number of spheres of class 10
13
Enter radius of spheres of class 10
(Integer <=33 please)
7
Enter phase of spheres of class 10
1
Enter number of spheres of class 11
21

Enter radius of spheres of class 11
(Integer <=33 please)
6
Enter phase of spheres of class 11
1
Enter number of spheres of class 12
38
Enter radius of spheres of class 12
(Integer <=33 please)
5
Enter phase of spheres of class 12
1
Enter number of spheres of class 13
73
Enter radius of spheres of class 13
(Integer <=33 please)
4
Enter phase of spheres of class 13
1
Enter number of spheres of class 14
174
Enter radius of spheres of class 14
(Integer <=33 please)
3
Enter phase of spheres of class 14
1
Enter number of spheres of class 15
450
Enter radius of spheres of class 15
(Integer <=33 please)
2
Enter phase of spheres of class 15
1
Enter number of spheres of class 16
2674
Enter radius of spheres of class 16
(Integer <=33 please)
1
Enter phase of spheres of class 16
1

Input User Choice
1) Exit
2) Add spherical particles (cement, gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
4

Phase counts are:
Porosity= 627570
Cement= 349940
C2S= 0
Gypsum= 10662
Anhydrite= 247
Hemihydrate= 11581
Pozzolan= 0
Inert= 0
Slag= 0
CaCO3= 0
Fly Ash= 0
Aggregate= 0

Input User Choice
1) Exit
2) Add spherical particles (cement, gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
3
Enter number of flocs desired at end of routine (>0)
1
Number left was 0 but number of clusters is 3322
Number left was 339 but number of clusters is 2937
Number left was 711 but number of clusters is 2557
Number left was 1080 but number of clusters is 2214
Number left was 1387 but number of clusters is 1938
Number left was 1652 but number of clusters is 1688
Number left was 1883 but number of clusters is 1453
Number left was 2102 but number of clusters is 1260
Number left was 2285 but number of clusters is 1093
Number left was 2457 but number of clusters is 932
Number left was 2594 but number of clusters is 806
Number left was 2713 but number of clusters is 706
Number left was 2794 but number of clusters is 624
Number left was 2873 but number of clusters is 545
Number left was 2948 but number of clusters is 489
Number left was 2998 but number of clusters is 435
Number left was 3052 but number of clusters is 394
Number left was 3085 but number of clusters is 355
Number left was 3128 but number of clusters is 312
Number left was 3166 but number of clusters is 282
Number left was 3191 but number of clusters is 253
Number left was 3223 but number of clusters is 225
Number left was 3250 but number of clusters is 208
Number left was 3263 but number of clusters is 199

Number left was 3273 but number of clusters is 181
Number left was 3291 but number of clusters is 158
Number left was 3314 but number of clusters is 148
Number left was 3324 but number of clusters is 141
Number left was 3330 but number of clusters is 128
Number left was 3342 but number of clusters is 117
Number left was 3354 but number of clusters is 108
Number left was 3362 but number of clusters is 96
Number left was 3374 but number of clusters is 85
Number left was 3385 but number of clusters is 74
Number left was 3396 but number of clusters is 66
Number left was 3404 but number of clusters is 59
Number left was 3411 but number of clusters is 53
Number left was 3417 but number of clusters is 47
Number left was 3422 but number of clusters is 44
Number left was 3425 but number of clusters is 40
Number left was 3429 but number of clusters is 38
Number left was 3431 but number of clusters is 35
Number left was 3433 but number of clusters is 34
Number left was 3435 but number of clusters is 33
Number left was 3436 but number of clusters is 31
Number left was 3438 but number of clusters is 27
Number left was 3442 but number of clusters is 24
Number left was 3445 but number of clusters is 23
Number left was 3446 but number of clusters is 22
Number left was 3447 but number of clusters is 21
Number left was 3448 but number of clusters is 19
Number left was 3449 but number of clusters is 16
Number left was 3453 but number of clusters is 15
Number left was 3453 but number of clusters is 14
Number left was 3454 but number of clusters is 13
Number left was 3455 but number of clusters is 13
Number left was 3455 but number of clusters is 13
Number left was 3455 but number of clusters is 11
Number left was 3458 but number of clusters is 10
Number left was 3458 but number of clusters is 10
Number left was 3458 but number of clusters is 9
Number left was 3459 but number of clusters is 9
Number left was 3459 but number of clusters is 9
Number left was 3459 but number of clusters is 9
Number left was 3459 but number of clusters is 9
Number left was 3460 but number of clusters is 8
Number left was 3461 but number of clusters is 7
Number left was 3461 but number of clusters is 6
Number left was 3463 but number of clusters is 4
Number left was 3464 but number of clusters is 4
Number left was 3464 but number of clusters is 4
Number left was 3464 but number of clusters is 4
Number left was 3464 but number of clusters is 3
Number left was 3465 but number of clusters is 3
Number left was 3465 but number of clusters is 3

Number left was 3465 but number of clusters is 3
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 2
Number left was 3466 but number of clusters is 1

Input User Choice
1) Exit
2) Add spherical particles (cement, gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
8
Enter name of file to save microstructure to
cem152w04floc.img
Enter name of file to save particle IDs to
pcem152w04floc.img

Input User Choice
1) Exit
2) Add spherical particles (cement, gypsum, pozzolans, etc.) to microstructure
3) Flocculate system by reducing number of particle clusters
4) Measure global phase fractions
5) Add an aggregate to the microstructure
6) Measure single phase connectivity (pores or solids)
7) Measure phase fractions vs. distance from aggregate surface
8) Output current microstructure to file
1

3.2 Assignment of Phases to 3-D Cement Particles

In version 2.0 of the CEMHYD3D programs, an extensive series of steps was necessary to properly distribute the four major cement clinker phases (C_3S, C_2S, C_3A, and C_4AF) amongst the cement particles (or more specifically the voxels comprising the cement particles in the three-dimensional microstructure). The goal of this phase distribution process is to generate a three-dimensional starting cement microstructure that matches the two-dimensional SEM/X-ray images of the real cement powder in terms of its particle size distribution (completed via **genpartnew**), phase volume (area in 2-D) fractions, and phase surface area (perimeter in 2-D) fractions. Matching of the latter two of these characteristics is achieved using the program **distrib3d** in version 3.0 of CEMHYD3D. This program accepts as input the starting microstructure consisting of cement (including calcium sulfate, limestone, etc.) particles, and the measured two-dimensional correlation files for various phase combinations (typically silicates, C_3S, and either C_3A or C_4AF) [2]. It then distributes the four major cement clinker phases amongst all of the particles labeled as cement, to achieve the user-requested volume and surface area fractions for each of these phases. The measured correlation files for the cements available in the cement images database are typically available at the ftp site, as outlined in section 2. These (three) correlation files must be downloaded to the directory where the **distrib3d** program is being executed, prior to its execution. The files typically all have the same root name (e.g., cement152), with the extensions **sil**, **c3s**, **c3a**, and **c4f** indicating the correlation files for the silicates, C_3S, C_3A, and C_4AF, respectively.

An example annotated datafile for using **distrib3d** is as follows:

-99	random number seed
cem152w04floc.img	filename of original microstructure image
cement152	file root name for correlation filters (sil, c3s, c3a, etc.)
cement152w04flocf.img	filename under which to save final microstructure
0.7344	volume fraction of C_3S in microstructure to be created
0.6869	surface area fraction of C_3S in microstructure to be created
0.0938	volume fraction of C_2S in microstructure to be created
0.1337	surface area fraction of C_2S in microstructure to be created
0.1311	volume fraction of C_3A in microstructure to be created
0.1386	surface area fraction of C_3A in microstructure to be created
0.0407	volume fraction of C_4AF in microstructure to be created
0.0408	surface area fraction of C_4AF in microstructure to be created

The output created by executing the program **distrib3d** with the above input datafile (using a command such as ***distrib3d <distrib3d.dat >distrib3d.out*** at the command line prompt of a Linux or UNIX-based system) is as follows:

Enter random number seed (negative integer)
-99
Enter name of cement microstructure image file
cem152w04floc.img
Enter root name of cement correlation files
cement152
Enter name of new cement microstructure image file
cem152w04flocf.img

0.734400
0.686900
0.093800
0.133700
0.131100
0.138600
0.040700
0.040800
Number of points in correlation file is 60
Critical volume fraction is 3.194648

Phase ID	Volume count	Surface count	Volume fraction	Surface fraction
0	627570	0		
1	290463	267309	0.83004	0.83543
2	0	0	0.00000	0.00000
3	59477	52656	0.16996	0.16457
4	0	0	0.00000	0.00000
Total	349940	319965		
5	10662	19661		
6	11581	14196		
7	247	634		
8	0	0		
9	0	0		
10	0	0		
11	0	0		
20	0	0		
24	0	0		
25	0	0		
26	0	0		
27	0	0		
28	0	0		

nsph is 179
Phase area count is 290463
Phase surface count is 267309
Hydraulic radius is 1.629928
Now: 1.629928 Target: 1.659389
Cycle: 1
ntot is 400
Ave. solid curvature: 59.118914
Ave. air curvature: 112.222842
Phase area count is 290463
Phase surface count is 266566
Hydraulic radius is 1.634471
Now: 1.634471 Target: 1.659389
Cycle: 2
ntot is 400
Ave. solid curvature: 59.017920
Ave. air curvature: 112.716058
Phase area count is 290463
Phase surface count is 265868
Hydraulic radius is 1.638762

Now: 1.638762 Target: 1.659389
Cycle: 3
ntot is 400
Ave. solid curvature: 58.920623
Ave. air curvature: 113.191217
Phase area count is 290463
Phase surface count is 265161
Hydraulic radius is 1.643132
Now: 1.643132 Target: 1.659389
Cycle: 4
ntot is 400
Ave. solid curvature: 58.825730
Ave. air curvature: 113.654640
Phase area count is 290463
Phase surface count is 264478
Hydraulic radius is 1.647375
Now: 1.647375 Target: 1.659389
Cycle: 5
ntot is 400
Ave. solid curvature: 58.732592
Ave. air curvature: 114.109488
Phase area count is 290463
Phase surface count is 263750
Hydraulic radius is 1.651922
Now: 1.651922 Target: 1.659389
Cycle: 6
ntot is 400
Ave. solid curvature: 58.641355
Ave. air curvature: 114.555055
Phase area count is 290463
Phase surface count is 263055
Hydraulic radius is 1.656287
Now: 1.656287 Target: 1.659389
Cycle: 7
ntot is 400
Ave. solid curvature: 58.551970
Ave. air curvature: 114.991577
Phase area count is 290463
Phase surface count is 262474
Hydraulic radius is 1.659953
Number of points in correlation file is 60
Critical volume fraction is 3.287097

Phase ID	Volume count	Surface count	Volume fraction	Surface fraction
0	627570	0		
1	257658	232284	0.73629	0.72597
2	32805	30190	0.09374	0.09435
3	59477	57491	0.16996	0.17968
4	0	0	0.00000	0.00000
Total	349940	319965		
5	10662	19661		

```
 6       11581        14196
 7         247         634
 8           0           0
 9           0           0
 10          0           0
 11          0           0
20           0           0
24           0           0
25           0           0
26           0           0
27           0           0
28           0           0
nsph is 179
Phase area count is 257658
Phase surface count is 232284
Hydraulic radius is 1.663855
Now: 1.663855  Target: 1.759084
Cycle: 1
ntot is 400
Ave. solid curvature: 58.195903
Ave. air curvature: 95.943789
Phase area count is 257658
Phase surface count is 231676
Hydraulic radius is 1.668222
Now: 1.668222  Target: 1.759084
Cycle: 2
ntot is 400
Ave. solid curvature: 58.082707
Ave. air curvature: 96.832861
Phase area count is 257658
Phase surface count is 231100
Hydraulic radius is 1.672380
Now: 1.672380  Target: 1.759084
Cycle: 3
ntot is 400
Ave. solid curvature: 57.971901
Ave. air curvature: 97.703155
Phase area count is 257658
Phase surface count is 230527
Hydraulic radius is 1.676537
Now: 1.676537  Target: 1.759084
Cycle: 4
ntot is 400
Ave. solid curvature: 57.863501
Ave. air curvature: 98.554550
Phase area count is 257658
Phase surface count is 229905
Hydraulic radius is 1.681073
Now: 1.681073  Target: 1.759084
Cycle: 5
ntot is 400
```

Ave. solid curvature: 57.757291
Ave. air curvature: 99.388752
Phase area count is 257658
Phase surface count is 229289
Hydraulic radius is 1.685589
Now: 1.685589 Target: 1.759084
Cycle: 6
ntot is 400
Ave. solid curvature: 57.653141
Ave. air curvature: 100.206767
Phase area count is 257658
Phase surface count is 228660
Hydraulic radius is 1.690226
Now: 1.690226 Target: 1.759084
Cycle: 7
ntot is 400
Ave. solid curvature: 57.550843
Ave. air curvature: 101.010242
Phase area count is 257658
Phase surface count is 228086
Hydraulic radius is 1.694479
Now: 1.694479 Target: 1.759084
Cycle: 8
ntot is 400
Ave. solid curvature: 57.450454
Ave. air curvature: 101.798720
Phase area count is 257658
Phase surface count is 227451
Hydraulic radius is 1.699210
Now: 1.699210 Target: 1.759084
Cycle: 9
ntot is 400
Ave. solid curvature: 57.351901
Ave. air curvature: 102.572779
Phase area count is 257658
Phase surface count is 226840
Hydraulic radius is 1.703787
Now: 1.703787 Target: 1.759084
Cycle: 10
ntot is 400
Ave. solid curvature: 57.255082
Ave. air curvature: 103.333211
Phase area count is 257658
Phase surface count is 226297
Hydraulic radius is 1.707875
Now: 1.707875 Target: 1.759084
Cycle: 11
ntot is 400
Ave. solid curvature: 57.160092
Ave. air curvature: 104.079287
Phase area count is 257658

Phase surface count is 225773
Hydraulic radius is 1.711839
Now: 1.711839 Target: 1.759084
Cycle: 12
ntot is 400
Ave. solid curvature: 57.066433
Ave. air curvature: 104.814906
Phase area count is 257658
Phase surface count is 225188
Hydraulic radius is 1.716286
Now: 1.716286 Target: 1.759084
Cycle: 13
ntot is 400
Ave. solid curvature: 56.974470
Ave. air curvature: 105.537205
Phase area count is 257658
Phase surface count is 224642
Hydraulic radius is 1.720457
Now: 1.720457 Target: 1.759084
Cycle: 14
ntot is 400
Ave. solid curvature: 56.883893
Ave. air curvature: 106.248621
Phase area count is 257658
Phase surface count is 224060
Hydraulic radius is 1.724926
Now: 1.724926 Target: 1.759084
Cycle: 15
ntot is 400
Ave. solid curvature: 56.794332
Ave. air curvature: 106.952050
Phase area count is 257658
Phase surface count is 223511
Hydraulic radius is 1.729163
Now: 1.729163 Target: 1.759084
Cycle: 16
ntot is 400
Ave. solid curvature: 56.706394
Ave. air curvature: 107.642737
Phase area count is 257658
Phase surface count is 222932
Hydraulic radius is 1.733654
Now: 1.733654 Target: 1.759084
Cycle: 17
ntot is 400
Ave. solid curvature: 56.620039
Ave. air curvature: 108.320988
Phase area count is 257658
Phase surface count is 222344
Hydraulic radius is 1.738239
Now: 1.738239 Target: 1.759084

Cycle: 18
ntot is 400
Ave. solid curvature: 56.534926
Ave. air curvature: 108.989483
Phase area count is 257658
Phase surface count is 221765
Hydraulic radius is 1.742777
Now: 1.742777 Target: 1.759084
Cycle: 19
ntot is 400
Ave. solid curvature: 56.451195
Ave. air curvature: 109.647127
Phase area count is 257658
Phase surface count is 221258
Hydraulic radius is 1.746771
Now: 1.746771 Target: 1.759084
Cycle: 20
ntot is 400
Ave. solid curvature: 56.368861
Ave. air curvature: 110.293797
Phase area count is 257658
Phase surface count is 220684
Hydraulic radius is 1.751314
Now: 1.751314 Target: 1.759084
Cycle: 21
ntot is 400
Ave. solid curvature: 56.287602
Ave. air curvature: 110.932023
Phase area count is 257658
Phase surface count is 220117
Hydraulic radius is 1.755825
Now: 1.755825 Target: 1.759084
Cycle: 22
ntot is 400
Ave. solid curvature: 56.207942
Ave. air curvature: 111.557689
Phase area count is 257658
Phase surface count is 219538
Hydraulic radius is 1.760456
Number of points in correlation file is 60
Critical volume fraction is 1.328790

Phase ID	Volume count	Surface count	Volume fraction	Surface fraction
0	627570	0		
1	257658	219538	0.73629	0.68613
2	32805	42936	0.09374	0.13419
3	45335	43659	0.12955	0.13645
4	14142	13832	0.04041	0.04323
Total	349940	319965		
5	10662	19661		
6	11581	14196		

```
 7        247        634
 8          0          0
 9          0          0
 10         0          0
 11         0          0
 20         0          0
 24         0          0
 25         0          0
 26         0          0
 27         0          0
 28         0          0
nsph is 179
Phase area count is 14142
Phase surface count is 13832
Hydraulic radius is 1.533618
Now: 1.533618  Target: 1.622425
Cycle: 1
ntot is 400
Ave. solid curvature: 58.462594
Ave. air curvature: 104.723878
Phase area count is 14142
Phase surface count is 13119
Hydraulic radius is 1.616968
Now: 1.616968  Target: 1.622425
Cycle: 2
ntot is 400
Ave. solid curvature: 56.526800
Ave. air curvature: 105.327738
Phase area count is 14142
Phase surface count is 12439
Hydraulic radius is 1.705362
```

4 Enhancements to the Three-Dimensional Cement Hydration Model in Version 3.0

4.1 Individual Phase Reactivities

In the past, for a given cement, the cycle-to-time conversion factor in CEMHYD3D has been adjusted to provide the best fit to measured degree of hydration data for the cement as a whole, usually based on loss-on-ignition (LOI) measurements. The data set of Gutteridge and Dalziel [4] offers the possibility to examine and adjust the individual phase reactivities. They prepared pastes with a water-solids mass ratio (*w/s*) of 0.71 and monitored the consumption of the four major cement clinker phases using quantitative X-ray diffraction from ages of 6 h to 180 d. Their cement had a mass composition of 68 % C_3S, 14 % C_2S, 6 % C_3A, and 7 % C_4AF, with 2 % calcium sulfate. Its density was 3160 kg/m^3 and its fineness was 320 m^2/kg. For modeling in CEMHYD3D, it was assumed that the individual phase surface area fractions were equal to their bulk volume fractions. Then, a cement in the cement images database with a similar composition and fineness was selected to provide the needed correlation files. Model hydration was conducted under sealed conditions to mimic the experimental procedure and the individual phase volume fractions were monitored as a function of hydration cycles. A conversion factor of 0.00045 h/cycle^2 was used to convert between hydration cycles and real time, to provide a good fit to the overall degree of hydration data (based on LOI) provided by Gutteridge and Dalziel [4]. In comparing the degree of hydration data for the individual phases between the experiment and model, the fits for tricalcium silicate and tricalcium aluminate were quite reasonable without any further adjustment from version 2.0 of CEMHYD3D. But, it was observed that the dicalcium silicate and tetracalcium aluminoferrite were too reactive in version 2.0 of CEMHYD3D. Thus, their individual phase dissolution probabilities were each reduced by 50 % from their values in version 2.0. As can be seen in Figure 2, this adjustment resulted in reasonable fits for **all four** sets of the degree of hydration data for the individual cement clinker phases.

4.2 Influence of Non-Reactive Fine Fillers

The data set of Gutteridge and Dalziel [4] also provides quantitative data on the influence of inert fine fillers on cement hydration. In addition to an ordinary portland cement paste, they also investigated a cement with a 16 % by mass fraction substitution of rutile (TiO_2) for cement. The rutile had a reported average particle size of 0.5 μm [4], and was modeled in the CEMHYD3D simulations by 1-voxel (1 μm) particles of inert material. The density of rutile was taken as 4260 kg/m^3 for the purposes of converting from mass proportions to the volumetric ones required for the CEMHYD3D simulation. Next, consideration was given to the role of these fine inert particles in the cement hydration processes.

In Version 2.0 of the CEMHYD3D model [2], the "induction" period of cement hydration had been modeled by making the initial dissolution probabilities of all four of the major cement clinker phases (C_3S, C_2S, C_3A, and C_4AF) proportional to the volume of C-S-H that had formed. The best fit to experimental degree of hydration data for ordinary portland cements was obtained when these initial dissolution probabilities were made to be proportional to the normalized volume of C-S-H (the volume of C-S-H formed divided by the volume of the initial cement present) raised to the second power [2]. To

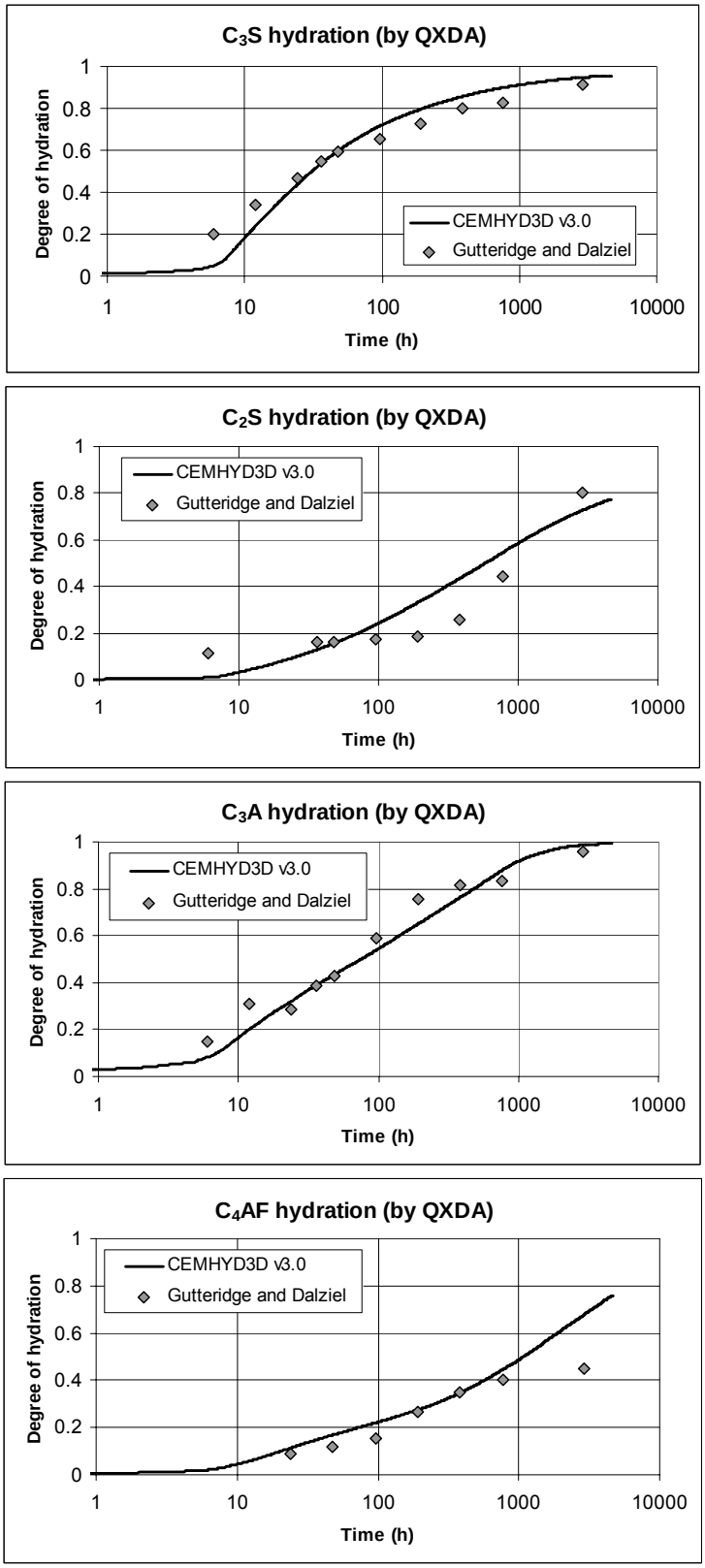

Figure 2: Comparison of CEMHYD3D v3.0 degrees of hydration for individual cement clinker phases with experimental data of Gutteridge and Dalziel [4].

model the "fine filler" effect in pastes with inert (rutile, etc.) substitutions for cement, the early time dissolution probabilities in CEMHYD3D have been further modified in version 3.0 of the codes to be also proportional to the ratio of the initial total (cement clinker and inert) surface area divided by the initial cement clinker surface area, once again raised to the second power. Modeling the influence of the substituted filler in this manner implies that hydration during the induction period is "accelerated" (or the length of the induction period is decreased) when a thinner C-S-H layer is formed over a larger surface area. It could also imply that less time is needed for the calcium (and hydroxide) ions to build up to some critical concentration in solution when the initial C-S-H is "dispersed" over a larger surface area than that provided solely by the initial cement particles. While neither of these mechanisms were included directly in the CEMHYD3D version 3.0 model, making the initial dissolution rates proportional to the ratio of the surface areas as described above would be consistent with either of them, and provides a simple approach for obtaining the desired effects. One could also consider a proportionality based on filler and cement clinker volumes, instead of surface areas. However, utilizing surface areas has the advantage that the fineness of the substituted filler, as well as its overall volume fraction, can influence the hydration.

The results of making this enhancement in the CEMHYD3D version 3.0 codes are summarized in Figure 3 which compares the experimentally measured degree of hydration (LOI-based) of the filled cement to the predictions provided by version 2.0 (no filler acceleration) and version 3.0 of the CEMHYD3D codes. Clearly, the above adjustment to the role of chemically inert fillers during hydration results in a much better fit to the available experimental data for times up to 100 h, beyond which the two versions provide essentially identical predictions. Further support for this approach will be provided in the Example Applications section to follow, where results on the influence of limestone additions on cement hydration will be presented.

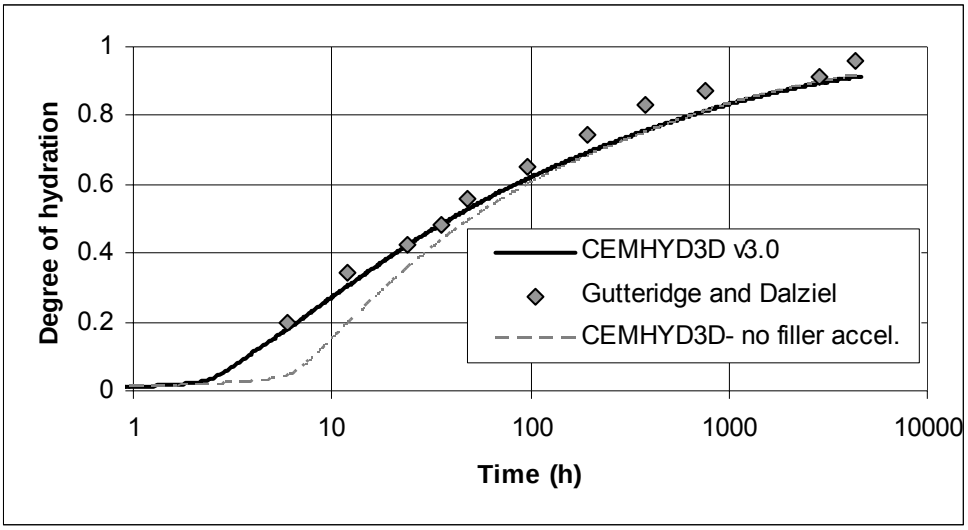

Figure 3: Comparison of CEMHYD3D model and experimental data of Gutteridge and Dalziel [4] for the influence of fine (rutile- TiO_2) filler on cement hydration.

4.3 Pore Solution Composition and its Influence on Hydration

Previous versions of CEMHYD3D considered only the formation of solid hydration products and offered no concrete information concerning the composition of the pore solution filling the pores during hydration. In version 3.0, a special module, **pHpred.c**, has been added to the CEMHYD3D codes to provide quantitative predictions of the pore solution composition and its electrical conductivity during the course of the hydration. Basically, the module considers the dissolution of sodium and potassium (sulfates) to supply ions in the pore solution. Further, the absorption of the Na+ and K+ ions by both the conventional and pozzolanic forms [5] of the C-S-H is considered.

Assumptions/Procedures:

1) Equilibrium exists between the ionic species [Na+], [K+], [OH-], [SO_4--], and [Ca++], and the solids calcium hydroxide ($Ca(OH)_2$), gypsum ($CaSO_4$-$2H_2O$), and syngenite ($K_2Ca(SO_4)_2$-H_2O).
2) The pore solution remains electrically neutral.
3) Calculations are performed based on activities and not simply concentrations. Activities are calculated according to a modified version of the Davies equation [6].
4) [K+] and [Na+] concentrations are calculated based on their measured release rate from the cement powder, adjusted due to their sorption by the primary and pozzolanic C-S-H phases produced during hydration, according to the procedure outlined by Taylor [5] and previously employed by van Eijk and Brouwers [7]. The user must thus provide estimates of the total and readily (1 h) soluble sodium oxide and potassium oxide mass fractions of the original cement (each as a **percentage**). These are typically provided in a datafile named **alkalichar.dat**, as will be outlined in the Execution of the Three-Dimensional Cement Hydration Model section to follow. For the readily soluble alkalis, 90 % are assumed to be released immediately and the remaining 10 % over the course of the first 1 h of hydration. The remainder of the alkalis is assumed to be released from the cement in proportion to its degree of hydration at any given time (beyond 1 h).
5) The sulfate concentration is controlled by the presence of gypsum and goes to zero when the gypsum is basically consumed (and ettringite becomes unstable in the CEMHYD3D model).
6) Equilibrium concentrations in the pore solution are first computed with respect to gypsum and calcium hydroxide, and then syngenite precipitation is considered. As syngenite precipitates, only the [K+] concentration is adjusted, as it is assumed that gypsum dissolution and calcium hydroxide precipitation will maintain the sulfate and calcium ion concentrations at their equilibrium levels.
7) Once concentrations are calculated, pore solution conductivity is estimated using the approach outlined previously by Snyder et al. [8].
8) The activity products, K_{SP}, for calcium hydroxide [9], calcium sulfate [9], and syngenite [10] are taken from the available literature and are provided in Table 1. Only that of calcium hydroxide is adjusted for the effects of temperature, based on the solubility data provided in Taylor [11].

Notation

γ = activity coefficient
K_{SP} = activity product
[X] = concentration of ionic species X

Equations:

Gypsum equilibrium:

$$\gamma_{Ca++}[Ca++]\gamma_{SO4--}[SO_4--] = K_{SP}^{Gypsum} \quad (1)$$

Calcium hydroxide equilibrium:

$$\gamma_{Ca++}[Ca++]\gamma_{OH-}^{2}[OH-]^2 = K_{SP}^{CH} \quad (2)$$

Electroneutrality:

$$2[Ca++] + [Na+] + [K+] - [OH-] - 2[SO_4--] = 0 \quad (3)$$

Equations 1 and 2 are solved for the $[SO_4--]$ and [OH-] concentrations, respectively, and substituted into equation 3 resulting in a fourth order equation for the [Ca++] concentration which is solved numerically [12] in the **pHpred.c** code.

Table 1: Activity products for solid phases of relevance to pHpred code [9, 10].

Phase	Activity product (K_{SP}) at 25 ^{o}C
Calcium hydroxide	6.46E-6
Gypsum	2.63E-5
Syngenite	1.0E-7

The above methodology has been used to predict pH, [K+], [Na+], etc. for several cements, with good agreement observed between the model predictions and experimental measurements [13].

In addition to predicting the development of the composition of the pore solution, it is also of interest to model its concurrent influence on hydration rates. In version 3.0 of CEMHYD3D, the pH and sulfate ion concentration of the pore solution are used to compute a "pHfactor" that can either increase or decrease the nominal dissolution rates of the four cement clinker phases. The following multi-step function is used to compute this pH factor:

pHfactor = $[SO_4--]$ +

1.5	for pH $<$ 12.5
1.0	12.5 $\leq$ pH $<$ 12.75
0.667	12.75 $\leq$ pH $<$ 13.0
0.333	13.0 $\leq$ pH $<$ 13.25
0.0	13.25 $\leq$ pH $<$ 13.75
-0.25	pH $\geq$ 13.75

The dissolution probabilities of the four major cement clinker phases, slag, and aluminosilicate phases (in fly ash, for example) are divided by the value of (1+pHfactor) to obtain the final influence of pore solution composition on hydration rates. Alternately, the pHfactor can be made to be a continuous function of pH and $[SO_4--]$; with the experimental data available to date, either the step or the continuous function provides a reasonable fit and neither approach is clearly superior to the other. But, the general effect of either procedure is the well-known one that higher pH (hydroxyl ion concentration) solutions tend to accelerate the hydration reactions of cements (fly ashes, and slags) [11]. The use of the

CEMHYD3D v3.0 model to predict the hydration rates of cements with and without alkali additions will be presented in the Example Applications section to follow.

4.4 Incorporation of Reactions for Limestone in CEMHYD3D

In addition to the fine filler (surface area) effect described above, a chemical reaction between calcium carbonate (limestone) filler and cement has been included in version 3.0 of CEMHYD3D. Based on experimental observations in the published literature [14-18], the formation of a monocarboaluminate phase [AFmc= $(CaO)_3(Al_2O_3)\bullet CaCO_3\bullet 11H_2O$] in preference to the formation of a monosulfoaluminate phase [AFm= $(CaO)_3(Al_2O_3)\bullet CaSO_4\bullet 12H_2O$] was added by modifying the CEMHYD3D computer codes to include the following reaction:

$$3\ (CaO)_3(Al_2O_3)\bullet CaSO_4\bullet 12H_2O + 2\ CaCO_3 + 18\ H_2O \rightarrow$$
$$2\ (CaO)_3(Al_2O_3)\bullet CaCO_3\bullet 11H_2O + (CaO)_3(Al_2O_3)\bullet 3\ CaSO_4\bullet 32H_2O$$

This reaction favors the production of AFmc (and ettringite) over that of the conventional AFm phase in the presence of calcium carbonate. In the CEMHYD3D model, this reaction becomes active only when the initial calcium sulfate is depleted and the previously formed ettringite begins to convert to the AFm phase by reaction with more of the cement clinker aluminate phases. This is in general agreement with experimental observations [15, 16]. While other reaction paths could be written for the formation of AFmc in a cement-based system, the above scheme was chosen for its simplicity in implementation in the CEMHYD3D codes and the fact that it does yield the desired effect: the formation of the AFmc phase at the expense of the AFm phase. Calcium carbonate generally has a low reactivity (because of its low solubility), and in typical simulations using the updated CEMHYD3D codes, for a 20 % by mass fraction substitution of ground limestone for cement, only about 5 % of the limestone present reacts during the first180 d of simulated hydration.

To incorporate limestone and AFmc into the CEMHYD3D codes, they were simply assigned phase IDs, and their relevant properties (specific gravity, molar volume, etc.) included into the computer program **disrealnew.c** (see program listing in Appendix B). Then, the reaction scheme outlined above, with the $CaCO_3$ dissolving to create ***DIFFCACO3*** species that react at the surfaces of available AFm, was added to the **disrealnew.c** and **hydrealnew.c** modules (listings provided in Appendix B).

4.5 Preliminary Incorporation of Reactions for Slag in CEMHYD3D

In version 3.0 of CEMHYD3D, hydration reactions for slag have been preliminary incorporated into the hydration programs. At this point, the simplest and most direct approach has been taken by assuming that the slag reacts with calcium hydroxide (pozzolanic-type reaction) to produce a single (mixed phase) hydration product. This product would be a mixture of a C-S-H-type gel with a lower Ca/Si ratio than conventional C-S-H, AFm, hydrotalcite, etc. [19]. The user must supply estimates of the specific gravity, molar volume, and Ca/Si and H_2O/Si molar ratios of this slag hydration product, in a datafile called **slagchar.dat**, as will be outlined in the Execution of the Three-Dimensional Cement Hydration Model section to follow. An estimate of the Ca/Si molar ratio of the slag hydration product may be obtained using SEM/X-ray analysis of well hydrated specimens [20, 21]. In the CEMHYD3D v3.0 model, for slags where the Mg/Al ratio is significantly less than the value of 2 that is commonly

encountered in a hydrotalcite-type product, the extra aluminate in the slag participates in hydration reactions similar to those of the C_3A phase in portland cement (e.g., formation of hydrogarnet, ettringite, and AFm).

Because the slag hydration product is typically observed to form in close vicinity to the initial slag grains [22] (perhaps due to a low mobility of the Mg++ or other ions), in version 3.0 of CEMHYD3D, the slag grains are basically hydrated in place, with slag voxels that are in contact with water-filled capillary porosity being eligible for conversion to the slag hydration product. The appropriate volume stoichiometry is maintained for this reaction by creating extra voxels of the slag hydration product in the local neighborhood of the reacting voxel, as needed. In addition, unlike the hydration of portland cement where the chemical shrinkage and self-desiccation occurs globally with the largest remaining water-filled capillary pores in the three-dimensional microstructure emptying first [1], for slag hydration, chemical shrinkage and self-desiccation occurs locally in an attempt to mimic the real world observation of (empty) porous regions forming within the hydrating slag grain. Compositions of two different slags that have been used to validate this portion of the CEMHYD3D codes, along with the assumed stoichiometry of their slag hydration products, are summarized in Table 2. For modeling the slag reactions under variable temperature conditions, it is necessary for the user to input an activation energy for the slag reactions as well. While for a Type I ordinary portland cement the activation energy for the hydration reactions is nominally on the order of 40 kJ/mol [23], for slag, a default value on the order of 80 kJ/mol, based on the work of DeSchutter [24], is recommended when no specific data on slag reactivity vs. temperature is available.

Table 2: Compositions of slags (in mass percentages) used in preliminary modeling of slag reactions in CEMHYD3D v 3.0.

SiO_2 (S) %	Al_2O_3 (A) %	CaO (C) %	MgO (M) %	SO_3 ($\overline{S}$) %	Slag comp.	Slag hydration product composition	Extra C_3A in slag
39.2	7.9	36.25	10.3	3.1	$C_{16.5}S_{17}M_{6.5}A_2\overline{S}$	$C_{21.25}S_{17}M_{6.5}A_2\overline{S}H_{86}$	No
34.7	11.4	45.5	8.5	3.0	$C_{21.5}S_{15}M_{5.5}A_3\overline{S}$	$C_{20.25}S_{15}M_{5.5}A_2\overline{S}H_{83}$	Yes

4.6 Other Additional Features in CEMHYD3D v 3.0

In addition to the new features and enhancements outlined in the previous five subsections, several miscellaneous additions have been made to CEMHYD3D v 3.0 computer codes. First, the ability to begin hydration under sealed conditions and later switch to saturated conditions has been included by allowing the user to specify a number of cycles of hydration after which the capillary porosity is once again filled (saturated) with water. Experimental studies have indicated that for intermediate *w/c* (0.4 to 0.45), some combination of sealed followed by saturated curing may result in an earlier depercolation of the capillary porosity, as hydration will initially be somewhat concentrated in the pore entryways and smaller pores, and not in the larger pores that will be emptying due to self-desiccation during sealed curing [25].

Second, the user has the option to specify the preferred morphology of the C-S-H gel formed during the hydration as either “random” or “plates”. In the presence of sufficient alkalis, SEM evidence

indicates that plate (lath-like) formations of the C-S-H gel may form as opposed to the more random morphologies observed in low-alkali cement pastes [26, 27]. The formation of a plate vs. a random morphology hydration product may result in an earlier depercolation of the capillary porosity, as exemplified by recent low temperature calorimetry measurements on a low-alkali cement with and without additional sodium and potassium sources (either hydroxides or sulfates) [28]. With this option, the assumption is being made that the morphological changes observed by SEM at scales of tens and hundreds of nanometers are also present at the one micrometer and higher scale implemented in the CEMHYD3D v3.0 model; this would infer a degree of self-similarity to the overall cement paste microstructure [29].

Finally, the user has the option of specifying a special dissolution bias factor for the one-voxel particles in the hydrating microstructure. Bullard has recently presented a derivation relating this factor to the PSD of particles smaller than a certain size (e.g., 2 μm) [30]. The basic premise is that in cements with a significant fraction of their particles smaller than 1 μm in diameter, representing the particles as 1-voxel (1 μm) particles in the CEMHYD3D model may result in the initial hydration rates being much lower than those observed experimentally (due to surface area effects, for instance). Via this factor, the user may increase (or decrease) the dissolution rates of (**only**) the one-voxel particles to achieve better agreement between model and experiment.

5 Execution of the Three-Dimensional Cement Hydration Model

5.1 Auxiliary HTML/Javascript Pages to Support Hydration Model

Two auxiliary HTML/JavaScript web pages are available to support the execution of the CEMHYD3D version 3.0 model. They can be accessed over the Internet and could be downloaded (saved) to any local computer for subsequent execution (as a file://.... address instead of an http://... one). Because these web pages use only HTML and JavaScript code, they can be executed locally without subsequent communication with the providing server. The two web pages are:

Calculation of number of one-pixel particles to add prior to hydration
Internet address: http://ciks.cbt.nist.gov/~bentz/onepixel.html
Purpose: To aid the user in calculating the number of one-voxel particles of each cement clinker and calcium sulfate phase to add to maintain their desired overall volume fractions.

Calculation of slag characteristics
Internet address: http://ciks.cbt.nist.gov/~bentz/slagcalc.html
Purpose: To aid the user in computing the characteristics of an individual slag that are needed as inputs for the CEMHYD3D cement hydration codes. The user must supply the molar values for the slag composition, the Ca/Si molar ratio for the slag hydration product (gel), and the specific gravities for the original slag and its hydration product.

5.2 Inputs

Proper execution of the CEMHYD3D v3.0 **disrealnew.c** computer program requires the prior creation of two supporting datafiles that provide the characteristics of the alkalis in the cement and the slag (if any) blended with the cement. Examples of these datafiles, which **must** have the filenames of **alkalichar.dat** and **slagchar.dat**, respectively, are as follows. Note also that these files must be located in the same physical directory as the executable **disrealnew** program.

Example Annotated alkalichar.dat datafile

0.093	sodium oxide content of the portland cement being modeled
0.186	potassium oxide content of the portland cement being modeled
0.017	readily-soluble (1 h) sodium oxide content of the portland cement
0.078	readily-soluble (1 h) potassium oxide content of the portland cement

All oxide contents are provided on a mass percentage basis.

Example Annotated slagchar.dat datafile (corresponds to the slag in second row of Table 2)

2714.219	molar mass of initial slag (g/mol)
4036.2	molar mass of slag hydration product (g/mol)
2.87	specific gravity of initial slag
2.35	specific gravity of slag hydration product
945.72	molar volume of initial slag (cm^3/mol)
1717.53	molar volume of slag hydration product (cm^3/mol)

1.4333	Ca/Si molar ratio in initial slag
1.35	Ca/Si molar ratio in slag hydration product
15.0	Si atoms per molar unit of slag
5.533	H_2O/Si molar ratio in slag hydration product
1.0	moles of "C_3A" per mole of slag
1.0	slag reactivity factor (default value of 1.0)

The following inputs are required for execution of version 3.0 of the CEMHYD3D hydration model (main program **disrealnew.c** provided in Appendix B):

- a negative integer random number seed,

- the filename of the file containing the initial 3-D microstructure to be used in the hydration model,

- for this file, the integer values assigned to C_3S, C_2S, C_3A, C_4AF, gypsum, hemihydrate, anhydrite, aggregate, and $CaCO_3$ (separated by spaces). Typical values following phase distribution using the **distrib3d.c** program would be 1, 2, 3, 4, 5, 6, 7, 28, and 26.

- the phase ID assigned to C_3A in any fly ash particles present in the starting microstructure [31]. Often, there is no fly ash present in the microstructure (or no C_3A present in the fly ash) and this value can be set to its default value of 35.

- the filename of the file containing the initial 3-D particle ID microstructure to be used in the hydration model (for assessing setting behavior). Typically, this file is created during the execution of the **genpartnew.c** program.

- the number of one-voxel (1 μm) particles of a phase to add. The program contains an iterative loop to continue to accept non-zero values for this parameter, so that the user can place different phase one-voxel particles, at their discretion. This iterative process is terminated by the user inputting a value of 0 for the number of one-voxel particles to add. Every time a non-zero entry is input, the next entry must be the phase ID of the particles to be placed. The phase IDs corresponding to each phase can be found in a series of **#define** statements near the top of the **disrealnew.c** listing in Appendix B.

- the number of cycles of the hydration model to execute. Note that this value can be set to zero if, for example, the user wants to output the initial microstructure before any hydration, but after addition of all of the one- voxel particles.

- a flag indicating if hydration is to be initially under (0) saturated or (1) sealed conditions,

- the maximum number of diffusion steps to take in a given dissolution cycle (typically a value of 500 is used here),

- a prefactor and a scale factor for the nucleation probability of CH according to an exponential function [32],

- a prefactor and a scale factor for the nucleation probability of calcium sulfate dihydrate (gypsum) forming from the hemihydrate and anhydrite forms of calcium sulfate,

- a prefactor and a scale factor for the nucleation probability of C_3AH_6,

- a prefactor and a scale factor for the nucleation probability of FH_3,

- the frequency (in cycles) for examining the percolation properties of the capillary pore space (this examination can be totally avoided by setting this parameter to a value larger than the requested number of hydration cycles). **If active, for hydrations that are initiated under saturated conditions, the CEMHYD3D model will automatically switch to sealed curing (with its accompanying self-desiccation, etc.), when the capillary porosity becomes depercolated in all three principal directions through the 3-D microstructure.**

- the frequency (in cycles) for examining the percolation properties of the solids (set point) with a typical value being 5,

- the frequency (in cycles) for outputting the hydration characteristics of all cement particles present in the microstructure,

- the frequency (in cycles) for outputting the hydrating microstructure to a file with a filename that indicates the starting microstructure name, the number of cycles, hydration conditions, etc. as will be detailed in the ***Outputs*** subsection to follow,

- the induction time in hours (*time_induct*) for use in converting model cycles to real time (now that the induction period is directly included in the hydration modeling, this value will normally be set to zero),

- the initial temperature of the system in degrees Celsius,

- the ambient temperature in degrees Celsius,

- an overall heat transfer coefficient between the hydrating specimen and its environment in units of J/(g•C•s) for hydration under semi-adiabatic conditions,

- the activation energy (kJ/mol) for the cement hydration reactions (if no further information is available, a value of 40 kJ/mol is suggested for Type I ordinary portland cements [23]),

- the activation energy (kJ/mol) for pozzolanic reactions,

- the activation energy (kJ/mol) for slag hydration reactions,

- the calibration factor (β) for converting model cycles to real time in hours based on an equation of the form $time=time\_induct+\beta \bullet cycles \bullet cycles$ [32],

- the mass fraction of aggregates (0.0 for cement paste hydration) in the concrete mixture proportions (**not** that present in the 3-D microstructure being used, but that present in the concrete mixture being simulated; this is used for the direct simulation of adiabatic heat signature curves [33]),

- a flag indicating if hydration is to be under (0) isothermal, (1) adiabatic/semi-adiabatic, or (2) temperature-programmed conditions. In the latter case, the time-temperature curing history for the simulation is read in from the file called **temphist.dat** located in the same directory as the executable **disrealnew** file.

- a flag indicating if conversion of conventional C-S-H to pozzolanic C-S-H is (0) prohibited or (1) allowed,

- a flag indicating if precipitation of calcium hydroxide on aggregate surfaces is (0) prohibited or (1) allowed [34, 35],

- the number of slices to include in a hydration movie (the central slice of the microstructure will be output every nth hydration cycle to obtain a "movie" of the hydration with the number of individual frames provided here by the user); a value of 0 will avoid the output of a hydration movie completely,

- a real number dissolution bias factor for the dissolution of one-voxel (soluble) particles in the 3-D microstructure (default value of 1.0) [30],

- the number of cycles of hydration to execute before totally resaturating the 3-D microstructure. This allows for the simulation of sealed/saturated curing conditions [25], and is only relevant when the user selects to initiate hydration under sealed conditions, or the hydration has previously switched over to sealed conditions due to depercolation of the capillary porosity as described above.

- a flag indicating if C-S-H morphology (geometry) is to be (0) random or (1) plate-like, and

- a flag indicating if the computed pH of the pore solution influences the hydration kinetics: (0) no or (1) yes.

An annotated example datafile for 1000 cycles of saturated hydration of CCRL cement 152 (w/c=0.40) could be as follows:

-2794	random number seed
cem152w04flocf.img	filename containing input 3-D phase ID microstructure
1 2 3 4 5 6 7 28 26	phase assignments for C_3S, C_2S, etc.
35	phase ID for C_3A in fly ash particles
pcem152w04floc.img	filename containing input 3-D particle ID microstructure
44990	number of one-pixel particles to add
1	add one-pixel particles of phase C_3S
5850	number of one-pixel particles to add

2	add one-pixel particles of phase C_2S
8692	number of one-pixel particles to add
3	add one-pixel particles of phase C_3A
2631	number of one-pixel particles to add
4	add one-pixel particles of phase C_4AF
1100	number of one-pixel particles to add
5	add one-pixel particles of gypsum
2062	number of one-pixel particles to add
6	add one-pixel particles of hemihydrate
839	number of one-pixel particles to add
7	add one-pixel particles of anhydrite
0	number of one-pixel particles to add
1000	number of cycles of hydration model to execute
0	flag for executing model under saturated conditions
500	maximum number of diffusion steps per cycle
0.0001 9000.	nucleation parameters for CH
0.01 9000.	nucleation parameters for gypsum (dihydrate)
0.00002 10000.	nucleation parameters for C_3AH_6
0.002 2500.	nucleation parameters for FH_3
50	cycle frequency for checking pore space percolation
5	cycle frequency for checking total solids percolation
5000	cycle frequency for outputting particle hydration stats
5000	cycle frequency for outputting hydrated microstructures
0.00	induction time in hours
20.0	initial specimen temperature in degrees Celsius
20.0	ambient temperature in degrees Celsius
0.0	overall heat transfer coefficient
40.0	activation energy for cement hydration
83.14	activation energy for pozzolanic reactions
80.0	activation energy for slag hydration
0.00035	conversion factor to go from cycles to real time
0.72	aggregate volume fraction in actual concrete mixture
0	flag indicating that hydration is under isothermal conditions
0	flag indicating that C-S-H conversion is prohibited
1	flag indicating that CH/aggregate precipitation is allowed
0	number of slices to include in a hydration movie file
1.0	one-voxel dissolution bias factor
0	number of cycles to execute before resaturation
0	flag indicating that C-S-H morphology is random
1	flag indicating that pH does influence hydration kinetics

5.3 Model Execution

To compile version 3.0 of CEMHYD3D, a typical command line on a UNIX (Linux) workstation might be:

cc disrealnew.c –o disrealnew –lm –O2.

Here, the **–lm** instructs the compiler to include the math libraries and the **–O2** (with a capital letter **O**) specifies the optimization level to be used by the compiler. In addition to the **disrealnew.c** program, the following programs and files must be in the same directory for the above compile command to function properly: **ran1.c**, **burn3d.c**, **burnset.c**, **hydrealnew.c**, **parthyd.c**, **pHpred.c**, **complex.c**, and **nrutil.c**. The latter two are used for computing the composition of the pore solution, as outlined in section 4.3, and are readily available from reference [12]. Their listings are not included in Appendix B, but they are available from for downloading from the pub/bfrl/bentz/CEMHYD3D/version30 subdirectory of the NIST anonymous ftp site (**ftp.nist.gov**).

If an input file **disrealnew.dat** had been created with an editor (similar to the annotated example datafile provided above), the program could be executed (with piping of the input and output streams) as:

disrealnew <disrealnew.dat >disrealnew.out &.

Here, the **&** indicates that the program will be executed in the background and the command line prompt will thus be returned immediately to the user for the execution of further commands.

The 3-D cement hydration and microstructure development model is described in detail elsewhere [32, 36]. Once the initial 3-D microstructure and particle ID structures are read in and the desired number of one-voxel particles of the various phases are added, hydration is executed as a series of dissolution/diffusion/reaction cycles. In dissolution, any voxels of a soluble phase that are in contact with water-filled capillary pore space may dissolve and enter solution as one or more diffusing species. These diffusing species then undergo random walks within the water-filled pore space until a reaction occurs. After a specified number (typically 500) of diffusion steps have taken place, phase counts are tabulated, empty porosity created to account for the chemical shrinkage (if hydration is under sealed conditions), the composition of the pore solution is recomputed, and another cycle of dissolution is executed. All diffusing species IDs and locations are stored in a dynamically allocated doubly linked list, so that the diffusing species may remain in solution from one dissolution cycle to the next. However, during the last diffusion step of the last requested cycle of hydration, all diffusing species are converted into either the phase from which they dissolved or the appropriate hydration product.

In addition, after each dissolution cycle, the heat released by the reactions and the degree of hydration that has occurred are used to update the heat capacity and temperature of the system (if the hydration is being executed under adiabatic or semi-adiabatic conditions). The heat capacity is a function of degree of hydration, and undergoes about a 10 % decrease during the course of the hydration due to changes in the state of the water present in the hydrating cement paste [37]. Dividing the incremental heat release by the current heat capacity determines the incremental temperature rise of the system under adiabatic conditions. In this way, the adiabatic response of a concrete system may be predicted by estimating its temperature as a function of equivalent real time, calculated based on the application of the well-developed principles of the maturity method [38].

5.4 Outputs

As the **disrealnew** program executes, a series of output files are created to capture various characteristics of the hydration simulation. All of the output files can be easily imported into a spreadsheet package for further analysis and plotting purposes. The names of the different output files are chosen to capture the parameters of the hydration simulation, in the following manner. Each created filename is of the form:

microstructurename.key.cycles.temp.flags

where

microstructurename corresponds to the root portion of the filename of the input 3-D phase microstructure that is being hydrated (for **cem152w04flocf.img**, it would be **cem152w04flocf**)

key is a code identifying the different types of output files including

adi – adiabatic temperature rise output file
chs – chemical shrinkage output file
heat – heat release and degree of hydration output file
ima – images of the hydrated microstructure during the course of the simulation
img – image of the final hydrated microstructure
mov – 2-D slice images comprising the movie of the ongoing hydration simulation
par – input parameter list file
pha – voxel counts for each phase output file
phr – degree of hydrations of individual particles in the 3-D microstructure
phv – pore solution composition, pH, and electrical conductivity output file
pps – pore space percolation vs. degree of hydration or real time output file
pts - total solids percolation vs. degree of hydration or real time output file

cycles is either the number of user-requested hydration cycles (e.g., 1000) or the current number of executed hydration cycles (for the **ima** files)

temp is the user-supplied initial temperature of the hydrating specimen (e.g., 20)

flags is a series of three values (codes) indicating the values of the flags for C-S-H conversion (0 for prohibited or 1 for allowed), heat transfer boundary conditions for curing (0 for isothermal, 1 for adiabatic/semi-adiabatic, or 2 for temperature-programmed), and initial moisture transfer conditions for curing (0 for saturated or 1 for sealed)

Each output file contains a header row with the column headings. Examples based on the **disrealnew.dat** input file given above (with 5 rows or so of output for each) are as follows:

cem152w04flocf.adi.1000.20.000

Time(h)	Temperature	Alpha	Krate	Cp_now	Mass_cem	kpozz/khyd	kslag/khyd
0.000000	20.000000	0.000000	0.759398	0.000000	0.201171	0.743166	0.759398
0.000461	20.000000	0.000088	0.759398	0.000000	0.201171	0.743166	0.759398
0.001844	20.000000	0.000401	0.759398	0.000000	0.201171	0.743166	0.759398
0.004148	20.000000	0.000947	0.759398	0.000000	0.201171	0.743166	0.759398
0.007374	20.000000	0.001242	0.759398	0.000000	0.201171	0.743166	0.759398

Here, for each cycle, the file contains the estimated equivalent time (age) in hours, the specimen temperature in degrees Celsius, the degree of hydration of the cement on a mass basis, the estimated

reaction rate at the current temperature, the current estimate of the heat capacity of the mortar or concrete in J/g•^{o}C, the current mass fraction of cement in the 3-D microstructure, the ratio of the rate constant for the pozzolanic reactions to that for the cement hydration, and the ratio for the rate constant for the slag hydration to that for the cement hydration. The current mass fraction of cement should remain constant unless the hydration is being executed under saturated conditions, in which case the additional water imbibed into the hydrating cement paste will reduce the mass fraction of cement in the overall 3-D microstructure. In the above example executed under isothermal conditions, all values for the heat capacity are 0.0, as they are only determined (and needed) when the requested hydration is under adiabatic or semi-adiabatic conditions.

cem152w04flocf.chs.1000.20.000

Cycle time(h) alpha_mass Chemical shrinkage (ml/g cement)
30 0.414802 0.012101 0.000150
31 0.442917 0.012415 0.000186
32 0.471953 0.013049 0.000223
33 0.501911 0.013596 0.000255
34 0.532791 0.014182 0.000278
35 0.564592 0.014483 0.000316

This file simply indicates for each hydration cycle, the equivalent real time, the degree of hydration on a mass basis, and the computed chemical shrinkage for the cement paste (ml water per gram of cement). The model-predicted chemical shrinkage values can be conveniently compared to physical measurements. Currently, a draft ASTM standard for a test method for chemical shrinkage of hydraulic cement paste is being balloted in the ASTM C01 (Cement) main committee. When performing such a comparison, however, it is important to account for the sample preparation and equilibration time before the first physical chemical shrinkage reading is obtained. The draft standard takes a "zero" reading 1 h after the casting of the cement paste, so that all model values should first be reduced by the model 1 h chemical shrinkage value before comparing to the experimental results.

cem152w04flocf.heat.1000.20.000

Cycle time(h) alpha_vol alpha_mass heat4(kJ/kg_solid) Gsratio2 G-s_ratio
0 0.000000 0.000000 0.000000 0.000000 0.000078 0.000000
1 0.000461 0.000092 0.000088 0.139287 0.000422 0.000148
2 0.001844 0.000410 0.000401 0.396576 0.000764 0.000676
3 0.004148 0.000971 0.000947 1.027811 0.001102 0.001597
4 0.007374 0.001269 0.001242 1.267228 0.001387 0.002092
5 0.011522 0.001514 0.001483 1.477603 0.001770 0.002498

The heat release output file contains the hydration cycle, the equivalent real time, the degree of hydration of the cement paste on a volume and a mass basis, the cumulative heat released (in units of kJ/kg solid), and two estimates of the Powers gel-space ratio [11] that can be used in compressive strength predictions [2, 32, 36]. The first estimate, provided in the sixth column of the output file, is based on actual counting of the remaining voxels of "gel" and "space" in the hydrating microstructure. The second, in column seven, is based on a simple formula derived from Powers [39]:

$$X = \frac{0.68\alpha}{0.32\alpha + \frac{w}{c}} \tag{4}$$

where X is the gel-space ratio, α is the degree of hydration of the cement on a mass basis, and *w/c* is the water-cement ratio of the cement paste. As the second approach is not directly applicable (without modification) to blended cements, the first is preferred for general use in CEMHYD3D v 3.0.

cem152w04flocf.ima.100.20.000
When the user elects to output the complete microstructure after every n cycles of hydration, these files contain the individual microstructures as a set of 100 x 100 x 100 phase ID values.

cem152w04flocf.img.1000.20.000
At the end of the hydration execution, the final 3-D microstructure is output to this file. Once again, the file consists of a 100 x 100 x 100 array of the phase ID values corresponding to each voxel in the 3-D microstructure.

cem152w04flocf.mov.1000.20.000
This file contains the 2-D image (slices) comprising the hydration movie, if requested by the user. The file consists of a series of 100 x 100 digital images and can be easily animated to display a "movie" of the in-situ hydration.

cem152w04flocf.par.1000.20.000
This file is simply a list of the output created by the initialization and parameter input portions of the **disrealnew.c** program. It can be used to verify all of the hydration parameters that were utilized in a particular execution of the hydration simulation.

cem152w04flocf.phr.1000.20.000
When activated by the user, this file contains a list of the hydration characteristics of each particle (3 voxels and greater in diameter) present in the hydrating 3-D microstructure. The first line of each separate output section of this file contains the number of hydration cycles and the cumulative degree of hydration on a mass basis. Following this, there is a list containing particle ID, initial volume in voxels, current volume in voxels, and achieved degree of hydration for this particle on a volume basis. In this output list, particles with initial volumes of 0 voxels indicate the presence of calcium sulfate particles that do not contain any of the four major cement clinker phases on which the degree of hydration calculations are based. If the user were interested in monitoring the dissolution of the various calcium sulfate particles in the microstructure, they would have to modify the **parthyd.c** module to perform this task. As has been observed experimentally, generally, the smaller particles present in the microstructure will hydrate at a faster rate than the larger ones, due mainly to their higher surface-to-volume ratio.

cem152w04flocf.pha.1000.20.000

Cycle Porosity C3S C2S C3A C4AF GYPSUM HEMIHYD ANHYDRITE POZZ INERT SLAG ASG CAS2 CH CSH C3AH6 ETTR ETTRC4AF AFM FH3 POZZCSH SLAGCSH CACL2 FREIDEL STRAT GYPSUMS CACO3 AFMC AGG ABSGYP EMPTYP water_left

0 561406 302648 38655 54027 16773 11762 13643 1086 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 561406

1 561373 302648 38655 53990 16772 11756 13632 1086 0 0 0 0 0 0 0 0 52 5 0 0 0 0 0 0 0 4 0 0 0 0 0 561364

2 561224 302563 38651 53950 16770 11750 13621 1081 0 0 0 0 0 0 120 0 123 10 0 1 0 0 0 0 0 11 0 0 0 0 0 561225

3 561083 302488 38649 53809 16757 11742 13610 1080 0 0 0 0 0 1 234 0 202 10 0 1 0 0 0 0 0 13 0 0 0 0 0 561092

4 560947 302406 38647 53772 16755 11737 13602 1079 0 0 0 0 0 1 359 0 262 10 0 1 0 0 0 0 0 13 0 0 0 0 0 560965

This file contains a simple list of the voxel counts for all of the phases present in the 3-D microstructure as a function of number of hydration cycles (column 1). It can be used, for example, to compute the degree of hydration of individual cement clinker phases, such as C_3S, as was done for the results provided in section 4.1.

cem152w04flocf.phv.1000.20.000

Cycle time(h) alpha_mass pH sigma(S/m) [Na+] [K+] [Ca++] [SO4--] activityCa activityOH activitySO4 activityK molesSyngenite

290 38.7610 0.375138 13.2703 5.748907 0.070109 0.192016 0.001302 0.000000 0.1604 0.7109 1.0000 0.6480 0.000000

291 39.0288 0.377057 13.2711 5.758643 0.070310 0.192307 0.001299 0.000000 0.1603 0.7108 1.0000 0.6478 0.000000

292 39.2975 0.379023 13.2718 5.767858 0.070507 0.192577 0.001296 0.000000 0.1601 0.7107 1.0000 0.6477 0.000000

293 39.5671 0.380871 13.2724 5.776635 0.070693 0.192835 0.001294 0.000000 0.1599 0.7106 1.0000 0.6475 0.000000

294 39.8376 0.382661 13.2731 5.785333 0.070875 0.193093 0.001291 0.000000 0.1597 0.7105 1.0000 0.6474 0.000000

This file provides the results of the pore solution composition calculations in the program module **pHpred.c**. It includes the number of hydration cycles, the equivalent time in hours, the degree of hydration of the cement on a mass basis, the pore solution pH, its conductivity (sigma in S/m), the concentrations of sodium, potassium, calcium, and sulfate ions, their corresponding activities, and the cumulative moles of syngenite (if any) that have precipitated from the pore solution.

cem152w04flocf.pps.1000.20.000
Cycle time(h) alpha_mass conn_por total_por frac_conn
450 93.330540 0.572453 173327 259618 0.667623
450 93.330540 0.572453 174019 259618 0.670289
450 93.330540 0.572453 174771 259618 0.673185
500 115.222889 0.608860 134856 243181 0.554550
500 115.222889 0.608860 135854 243181 0.558654
500 115.222889 0.608860 135723 243181 0.558115

This file contains the results of the water-filled capillary pore percolation examination as a function of hydration cycles, equivalent time, and degree of hydration on a mass basis. The last three columns provide the number of voxels of water-filled capillary porosity that are part of a percolated pathway through the microstructure, the total number of voxels of water-filled capillary porosity, and the computed fraction for the connected porosity. Percolation is examined in each of the three principal directions (x, y, and z) through the 3-D microstructure. Thus, there are three rows of results presented (one for each direction) for each cycle value.

cem152w04flocf.pts.1000.20.000
Cycle time(h) alpha_mass conn_solid total_solid frac_conn
105 5.081329 0.052169 0 424559 0.000000
105 5.081329 0.052169 0 424559 0.000000
105 5.081329 0.052169 0 424559 0.000000
110 5.576787 0.056711 193474 426576 0.453551
110 5.576787 0.056711 0 426576 0.000000
110 5.576787 0.056711 188284 426576 0.441384
115 6.095291 0.062294 218143 428821 0.508704
115 6.095291 0.062294 227920 428821 0.531504
115 6.095291 0.062294 204861 428821 0.477731

This file contains the results of the total solids percolation examination (setting behavior) as a function of hydration cycles, equivalent time, and degree of hydration on a mass basis. The last three columns provide the number of voxels of "total solids" that are a part of a percolated pathway through the microstructure, the total number of voxels of these solids, and the computed fraction for the connected solids.

6 Example Applications

6.1 Influence of Limestone on Hydration

Recently, the ASTM C 150 standard specification for portland cement has been modified to permit the use of up to 5 % by mass limestone in ordinary portland cements [40]. Much higher limestone contents have been routinely employed throughout the rest of the world. The CEMHYD3D v3.0 model can be conveniently applied to simulating the influence of such limestone additions on hydration (rates) and microstructure development [41]. When considering the influences of limestone on cement hydration, as introduced in section 4, two effects contribute to the observed influences: a fine

particle effect where the limestone provides additional surfaces for the precipitation of hydration products, and a reactivity effect where the limestone participates in reactions with the calcium aluminate phases present in the hydrating cement paste microstructure. Generally, limestone is not very reactive, so that the influences are usually dominated by the first effect, that of a fine inert material.

Figures 4 and 5 present a comparison between CEMHYD3D v3.0 model predictions and measured experimental values for the degree of hydration of cement pastes with and without a 20 % by mass fraction substitution of limestone for portland cement. The simulations were conducted using starting cement paste microstructures that matched the particle size distributions (both cement and limestone), phase compositions, and phase distributions of the pastes investigated experimentally [41]. For all of the simulations conducted using CEMHYD3D v3.0, a conversion factor of 0.00035 h/cycle^2 was used to convert between model cycles and real time [41]. In the figures, the relative acceleration provided by the additional limestone surfaces is seen to be a strong function of the *w/s* of the cement paste. Thus, for *w/s*=0.435, little if any acceleration is observed when limestone is substituted for cement while at *w/s*=0.35, a considerable acceleration is observed. The substitution of limestone increases the effective *w/c* of the hydrating paste and this effect is much more important for *w/s* < 0.4, where the ultimate degree of hydration in the original cement paste is already limited to a value less than 1.0 due to space limitations (e.g., there is insufficient initial capillary pore space available for the precipitation of the volume of hydration products that would be created with complete hydration of the cement). Since the limestone is basically inert, its substitution at a constant *w/s* effectively provides a greater relative volume of pore space in which the hydration products can form. It must be recognized, however, that degree of hydration and strength development are not the same, as the projected increase in strength due to the acceleration "provided" by the limestone may be offset by the lower cement content (higher effective starting *w/c*) of the cement/limestone paste [41]. A quantitative analysis of these effects can be performed using the CEMHYD3D v3.0 model and its strength predictions based on Powers' gel-space ratio theory [39, 41].

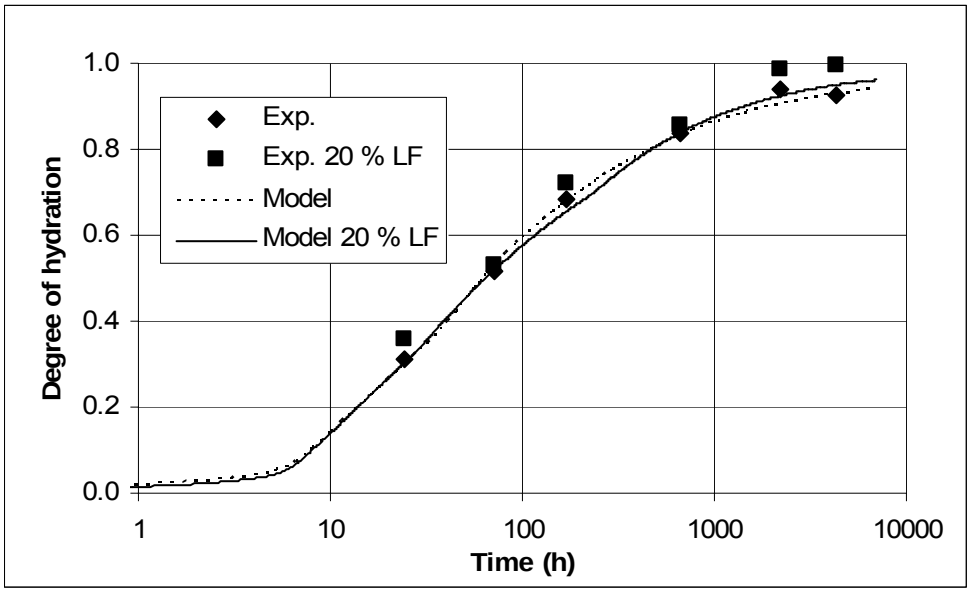

Figure 4: Experimental and CEMHYD3D v3.0 model estimated degrees of hydration for CCRL cement 152 with and without 20 % by mass fraction limestone substitution for w/s=0.435, cured under saturated conditions at 20 ^{o}C [41].

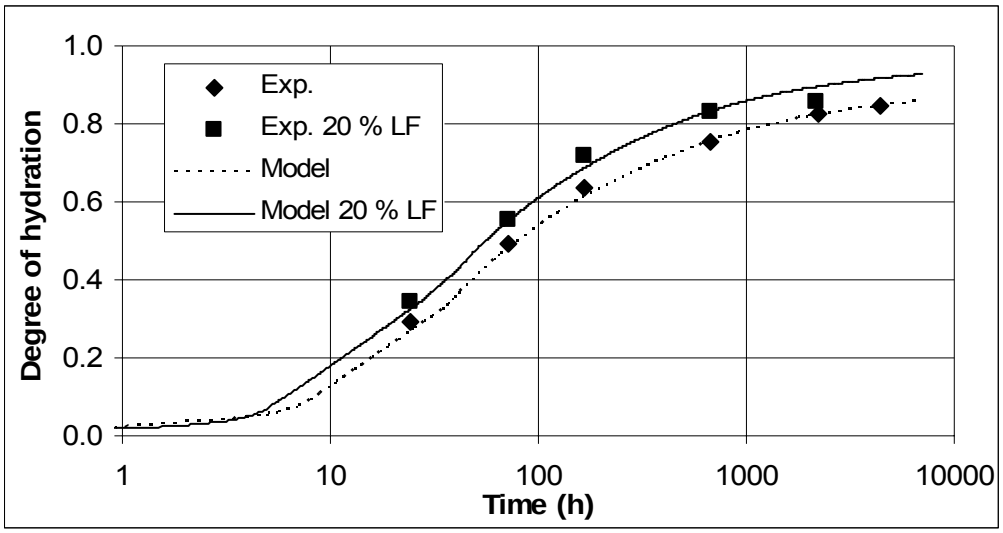

Figure 5: Experimental and CEMHYD3D v3.0 model estimated degrees of hydration for CCRL cement 152 with and without 20 % by mass fraction limestone substitution for w/s=0.35, cured under saturated conditions at 20 ^{o}C [41].

6.2 Influence of Added Alkalis on Hydration

To investigate the influence of added alkalis on hydration rates using CEMHYD3D v 3.0, a proficiency cement sample with a low initial alkali content was employed. Specifically, cement pastes with a *w/c* of 0.40 were prepared by mixing Cement and Concrete Reference Laboratory (CCRL) proficiency cement sample 140 [42] with water at 20 ^{o}C, using a high-speed blender. The mixing water was either distilled water or a solution of alkalis (sulfates or hydroxides), prepared by adding the appropriate sodium and potassium compounds to distilled water and stirring with a glass rod until complete dissolution. Cement 140 contains 0.093 % Na_2O and 0.186 % K_2O per unit mass of cement [42]. To prepare the cement paste with additional alkali sulfates, 0.76 % Na_2SO_4 and 0.93 % K_2SO_4 per unit mass of cement were added to the mixing water. For the alkali hydroxides, to maintain the same molar addition rates of the sodium and potassium, 0.43 % NaOH and 0.60 % KOH per unit mass of cement were added, being sure to account for the quoted 89 % purity of the commercially available KOH pellets. After mixing, cast wafers (≈5 g) of the paste were placed in sealed plastic vials. A small quantity of distilled water was added to the top of the wafers to maintain saturated curing conditions. The capped vials were placed in a walk-in environmental chamber maintained at 20 ^{o}C. At various ages, specimens of the pastes were removed from the vials for degree of hydration analysis based on loss-on-ignition (LOI) techniques.

The prepared cement 140 pastes were also simulated using CEMHYD3D v3.0. A starting microstructure was created to match the measured (spatial) characteristics of cement 140 and hydration was simply conducted with two different alkali datafiles, one representing the paste with no additional alkalis, and the other representing the system with additional alkalis (either hydroxides or sulfates). It was assumed that all of the additional alkalis added to the mixing water should be considered as readily soluble alkalis in the CEMHYD3D model. A conversion factor of 0.0003 h/cycle^2 was used to convert between model cycles and real time in the CEMHYD3D v3.0 model. For the simulation with additional

alkalis, the morphology of the C-S-H gel hydration product was selected to be plate-like, as opposed to random.

A comparison of the model predictions and the experimentally measured values for degree of hydration are provided in Figure 6. It can be seen that while the CEMHYD3D v3.0 provides a good prediction of the influence of alkalis at early ages up to 7 d, at later ages, the decrease in observed hydration for the systems with additional alkalis relative to the initial cement 140 paste is only qualitatively predicted by the model. As always, future enhancements and further refinements of the model will be necessary to improve its quantitative predictive capability in specific areas.

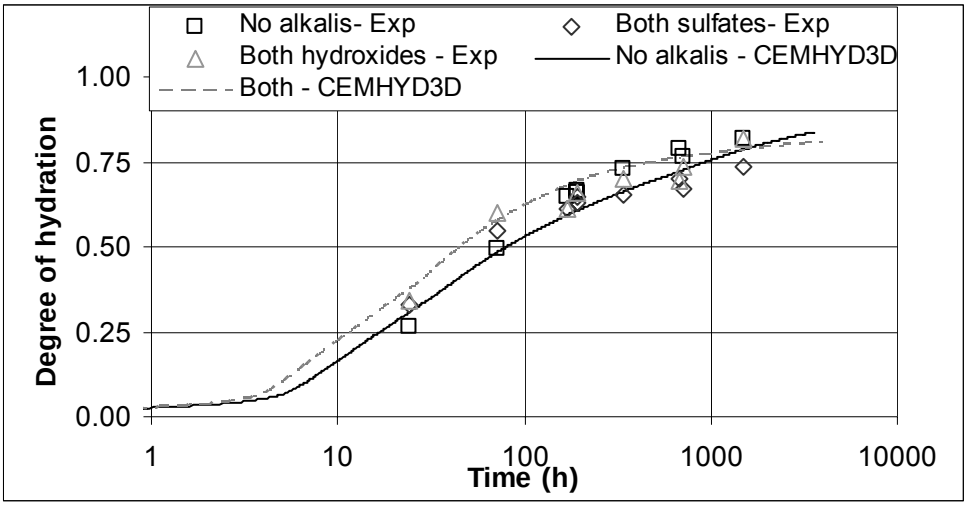

Figure 6: Influence of additional alkalis (potassium and sodium, either sulfates or hydroxides) on degree of hydration for experimental mixtures and simulations using CEMHYD3D v3.0.

7 Acknowledgements

The author would like to thank the NIST/industry Virtual Cement and Concrete Testing Laboratory (VCCTL) consortium for funding in support of several of the enhancements to the cement hydration and microstructure development computer programs comprising CEMHYD3D v3.0. He would also like to thank Dr. Xiuping Feng (formerly a guest researcher in NIST/BFRL) for her assistance with laboratory measurements in support of the pore solution and slag reactivity additions to CEMHYD3D v 3.0.

8 References

[1] Bentz, D.P., Guide to Using CEMHYD3D: A Three-Dimensional Cement Hydration and Microstructure Development Modelling Package, NISTIR **5977**, U.S. Department of Commerce, February, 1997 (available from National Technical Information Service).

[2] Bentz, D.P., CEMHYD3D: A Three-Dimensional Cement Hydration and Microstructure Development Modelling Package. Version 2.0, NISTIR **6485**, U.S. Department of Commerce, April, 2000 (available from National Technical Information Service).

[3] Stauffer, D., Introduction to Percolation Theory (Taylor and Francis, London, 1985).

[4] Gutteridge, W.A., and Dalziel, J.A., "Filler Cement: The Effect of the Secondary Component on the Hydration of Portland Cement. Part I. A Fine Non-Hydraulic Filler," *Cem Concr Res*, Vol. 20, 778-782 (1990).

[5] Taylor, H.F.W., "A Method for Predicting Alkali Ion Concentrations in Cement Pore Solutions," *Adv Cem Res*, Vol. 1 (1), 5-16 (1987).

[6] Samson, E., Lemaire, G., Marchand, J., Beaudoin, J.J., "Modeling Chemical Activity Effects in Strong Ionic Solutions," *Comp Mater Sci*, Vol. 15, 285-294 (1999).

[7] van Eijk, R.J., and Brouwers, H.J.H., "Prediction of Hydroxyl Concentrations in Cement Pore Water Using a Numerical Cement Hydration Model," *Cem Concr Res*, Vol. 30, 1801-1806 (2000).

[8] Snyder, K.A., Feng, X., Keen, B.D., and Mason, T.O., "Estimating the Electrical Conductivity of Cement Paste Pore Solutions from OH-, K+, and Na+ Concentrations," *Cem Concr Res*, Vol. 33 (6), 793-798 (2003).

[9] Reardon, E.J., "An Ion Model for the Determination of Chemical Equilibria in Cement/Water Systems," *Cem Concr Res*, Vol. 20 (2), 175-192 (1990).

[10] Gartner, E.J., Tang, F.J., and Weiss, S.J., "Saturation Factors for Calcium Hydroxide and Calcium Sulfates in Fresh Portland Cement Pastes," *J Amer Ceram Soc*, Vol. 68 (12), 667-673 (1985).

[11] Taylor, H.F.W., Cement Chemistry, 2nd edition (Thomas Telford, London, 1997).

[12] Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, W.T., Numerical Recipes in C: The Art of Scientific Computing (Cambridge University Press, Cambridge, 1990).

[13] Haecker, C.J., Bentz, D.P., Feng, X.P., and Stutzman, P.E., "Prediction of Cement Physical Properties by Virtual Testing," *Cem Inter*, Vol. 1 (3), 86-92 (2003).

[14] Hawkins, P., Tennis, P., and Detwiler, R., "The Use of Limestone in Portland Cement: A State-of-the-Art Review," EB227 (Portland Cement Association, Skokie, IL, 2003) 44 pp.

[15] Klemm, W.A., and Adams, L.D., "An Investigation of the Formation of Carboaluminates," in: P. Klieger, R.D. Hooton (Eds.), Carbonate Additions to Cement, ASTM STP 1064 (American Society for Testing and Materials, Philadelphia, PA, 1990) pp. 60-72.

[16] Kuzel, H.-J., and Pollmann, H. "Hydration of C_3A in the Presence of $Ca(OH)_2$, $CaSO_4 \cdot 2H_2O$ and $CaCO_3$," *Cem Concr Res*, Vol. 21, 885-895 (1991).

[17] Bonavetti, V.L., Rahhal, V.F., and Irassar, E.F., "Studies on the Carboaluminate Formation in Limestone Filler-Blended Cements," *Cem Concr Res*, Vol. 31, 853-859 (2001).

[18] Kakali, G., Tsivilis, S., Aggeli, E., and Bati, M. "Hydration Products of C_3A, C_3S and Portland Cement in the Presence of $CaCO_3$," *Cem Concr Res*, Vol. 30, 1073-1077 (2000).

[19] Wang, S.-D., "Alkali-Activated Slag: Hydration Process and Development of Microstructure," *Adv Cem Res*, Vol. 12 (4), 163-172 (2000).

[20] Richardson, J.R., Biernacki, J.J., Stutzman, P.E., and Bentz, D.P., "Stoichiometry of Slag Hydration in the Presence of Calcium Hydroxide," *J Amer Ceram Soc*, Vol. 85 (4), 947-953 (2002).

[21] Biernacki, J.J., Richardson, J.R., Stutzman, P.E., and Bentz, D.P., "Kinetics of Slag Hydration in the Presence of Calcium Hydroxide," *J Amer Ceram Soc*, Vol. 85 (9), 2261-2267 (2002).

[22] Feng, X., Garboczi, E.J., Bentz, D.P., Stutzman, P.E., and Mason, T.O., "Estimation of the Degree of Hydration of Blended Cement Pastes by a Scanning Electron Microscope Point-Counting Procedure," *Cem Concr Res*, Vol. 34, 1787-1793 (2004).

[23] ASTM C1074-04 "Standard Practice for Estimating Concrete Strength by the Maturity Method," ASTM Annual Book of Standards, Vol. 04.02 Concrete and Aggregates (ASTM International, West Conshohocken, PA, 2004).

[24] De Schutter, G., "Hydration and Temperature Development of Concrete Made with Blast-Furnace Slag Cement," *Cem Concr Res*, Vol. 29, 143-149 (1999).

[25] Bentz, D.P., and Stutzman, P.E., "Curing, Hydration, and Microstructure of Cement Paste," submitted to *ACI Mater J*, 2005.

[26] Mori, H., Sudoh, G., Minegishi, K., and Ohta, T., "Some Properties of C-S-H Gel Formed by C_3S Hydration in the Presence of Alkali," Proceedings 6th International Congress on the Chemistry of Cement, Moscow, V. 2, Supplementary Paper, Section 2, 2-12 (1974).

[27] Richardson, I.G., "Tobermite/Jennite- and Tobermorite/Calcium Hydroxide-Based Models for the Structure of C-S-H: Applicability to Hardened Pastes of Tricalcium Silicate, β-Dicalcium Silicate, Portland Cement, and Blends of Portland Cement with Blast-Furnace Slag, Metakaolin, or Silica Fume," *Cem Concr Res*, Vol. 34, 1733-1777 (2004).

[28] Bentz, D.P., "Influence of Alkalis on Porosity Percolation in Hydrating Cement Pastes," submitted to *Cem Concr Res*, 2005.

[29] Blinc, R., Lahajnar, G., Zumer, S., and Pintar, M.M., "NMR Study of the Time Evolution of the Fractal Geometry of Cement Gels," *Phys. Rev. B*, Vol. 38 (4), 2873-2875 (1988).

[30] Bullard, J.W., "Modeling the Hydration Kinetics of Fine Cements: Application to CEMHYD3D," submitted to *Cem Concr Res,* 2004.

[31] Bentz, D.P., and Remond, S., "Incorporation of Fly Ash into a 3-D Cement Hydration Microstructure Model," NISTIR **6050**, U.S. Department of Commerce, August 1997 (available from National Technical Information Service).

[32] Bentz, D.P., "A Three-Dimensional Cement Hydration and Microstructure Program. I. Hydration Rate, Heat of Hydration, and Chemical Shrinkage," NISTIR **5756**, U.S. Department of Commerce, November 1995 (available from National Technical Information Service).

[33] Bentz, D.P., Waller, V., and DeLarrard, F., "Prediction of Adiabatic Temperature Rise in Conventional and High-Performance Concretes Using a 3-D Microstructural Model," *Cem Concr Res*, Vol. 28 (2), 285-297 (1998).

[34] Bache, H., Idorn, G.M., Nepper-Christensen, P., and Nielsen, J., "Morphology of Calcium Hydroxide in Cement Paste," in HRB Special Report 90, Symposium on Structure of Portland Cement Paste and Concrete, Highway Research Board, 154-174 (1966).

[35] Diamond, S., and Huang, J., "The ITZ in Concrete- A Different View Based on Image Analysis and SEM Observations," *Cem Concr Comp*, Vol. 23, 179-188 (2001).

[36] Bentz, D.P., "Three-Dimensional Computer Simulation of Cement Hydration and Microstructure Development," *J Amer Ceram Soc*, Vol. 80 (1), 3-21 (1997).

[37] Waller, V., DeLarrard, F., and Roussel, P., "Modelling the Temperature Rise in Massive HPC Structures," *4th International Symposium on Utilization of High-Strength/High-Performance Concrete*, Paris, Paper 170, 415-421 (1996).

[38] Carino, N.J., Knab, L.I., and Clifton, J.R., "Applicability of the Maturity Method to High-Performance Concrete," NISTIR **4819**, U.S. Department of Commerce, May 1992 (available from National Technical Information Service).

[39] Powers, T.C., "Physical Properties of Cement Paste," in *Proceedings of the 4th International Symposium on the Chemistry of Cement*, Washington, D.C., Vol. 2, Paper V-1, 577-613 (1962).

[40] ASTM C 150-04, "Standard Specification for Portland Cement," ASTM Annual Book of Standards, Vol. 04.01 Cement; Lime; Gypsum (American Society for Testing and Materials, West Conshohocken, PA, 2004).

[41] Bentz, D.P., "Modeling the Influence of Limestone Filler on Cement Hydration Using CEMHYD3D," submitted to *Cem Concr Comp*, 2005.

[42] Cement and Concrete Reference Laboratory, "Cement and Concrete Reference Laboratory Proficiency Sample Program: Final Report on Portland Cement Proficiency Samples Number 139 and 140," Gaithersburg, MD, March 2001.

A. Computer programs for two-dimensional to three-dimensional conversion

Program genpartnew.c

```
/*********************************************************************/
/*                                                                   */
/*      Program genpartnew.c to generate three-dimensional cement    */
/*              and gypsum particles in a 3-D box with periodic      */
/*              boundaries.                                          */
/*      Particles are composed of either cement clinker, gypsum,     */
/*              slag, calcium carbonate, or inert filler,            */
/*              follow a user-specified size distribution, and can   */
/*              be either flocculated, random, or dispersed.         */
/*      Programmer: Dale P. Bentz                                    */
/*                  Building and Fire Research Laboratory            */
/*                  NIST                                             */
/*                  100 Bureau Drive Mail Stop 8615                  */
/*                  Gaithersburg, MD  20899-8615   USA               */
/*                  (301) 975-5865      FAX: 301-990-6891            */
/*                  E-mail: dale.bentz@nist.gov                      */
/*                                                                   */
/*********************************************************************/
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

/* Modified 3/97 to allow placement of pozzolanic, inert and fly ash particles */
/* Modified 9/98 to allow placement of various forms of gypsum */
/* Documented version produced 1/00 */
#include <stdio.h>
#include <math.h>

#define SYSSIZE 100     /* system size in pixels per dimension */
#define MAXTRIES 150000 /* maximum number of random tries for sphere placement */

/* phase identifiers */
#define POROSITY 0
/* Note that each particle must have a separate ID to allow for flocculation */
#define CEM 100           /* and greater */
#define CEMID 1           /* phase identifier for cement */
#define C2SID 2           /* phase identifier for C2S cement */
#define GYPID 5           /* phase identifier for gypsum */
```

```
#define HEMIHYDRATE 6     /* phase identifier for hemihydrate */
#define ANHYDRITE 7       /* phase identifier for anhydrite */
#define POZZID 8          /* phase identifier for pozzolanic material */
#define INERTID 9         /* phase identifier for inert material */
#define SLAGID 10         /* phase identifier for slag */
#define CACO3 26          /* phase identifier for calcium carbonate */
#define AGG 28            /* phase identifier for flat aggregate */
#define FLYASH 30         /* phase identifier for all fly ash components */
#define NPARTC 30000      /* maximum number of particles allowed in box*/
#define BURNT 34000       /* this value must be at least 100 > NPARTC */
#define NUMSIZES 200      /* maximum number of different particle sizes */

/* data structure for clusters to be used in flocculation */
struct cluster{
        int partid;     /* index for particle */
        int clustid;    /* ID for cluster to which this particle belongs */
        int partphase;  /* phase identifier for this particle */
        int x,y,z,r;    /* particle centroid and radius in pixels */
        struct cluster *nextpart;  /* pointer to next particle in cluster */
};

/* 3-D particle structure (each particle has own ID) stored in array cement */
/* 3-D microstructure is stored in 3-D array cemreal */
static unsigned short int cement [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
static unsigned short int cemreal [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
int npart,aggsize;     /* global number of particles and size of aggregate */
int *seed;             /* random number seed- global */
int dispdist;          /* dispersion distance in pixels */
int clusleft;          /* number of clusters in system */
/* parameters to aid in obtaining correct sulfate content */
long int n_sulfate=0,target_sulfate=0,n_total=0,target_total=0,volpart[47];
long int n_anhydrite=0,target_anhydrite=0,n_hemi=0,target_hemi=0;
float probgyp,probhem,probanh; /* prob. of gypsum particle instead of cement */
                            /* and probabilities of anhydrite and hemihydrate */
                               /* relative to total sulfate */
struct cluster *clust[NPARTC];      /* limit of NPARTC particles/clusters */

/* Random number generator ran1 from Computers in Physics */
/* Volume 6 No. 5, 1992, 522-524, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

double ran1(idum)
int *idum;
/* Calls: no routines */
/* Called by: gsphere,makefloc */
{
        int j,k;
        static int iv[NTAB],iy=0;
```

```
        void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

        if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
                        if(j < NTAB) iv[j] = *idum;
                }
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN

/* routine to add a flat plate aggregate in the microstructure */
void addagg()
/* Calls: no other routines */
/* Called by: main program */
{
        int ix,iy,iz;
        int agglo,agghi;

/* Be sure aggregate size is an even integer */
        do{
             printf("Enter thickness of aggregate to place (an even integer) \n");
             scanf("%d",&aggsize);
             printf("%d\n",aggsize);
        } while (((aggsize%2)!=0)||(aggsize>(SYSSIZE-2)));

        if(aggsize!=0){
                agglo=(SYSSIZE/2)-((aggsize-2)/2);
                agghi=(SYSSIZE/2)+(aggsize/2);

                /* Aggregate is placed in yz plane */
                for(ix=agglo;ix<=agghi;ix++){
                for(iy=1;iy<=SYSSIZE;iy++){
                for(iz=1;iz<=SYSSIZE;iz++){

        /* Mark aggregate in both particle and microstructure images */
```

```
                        cement [ix][iy][iz]=AGG;
                        cemreal [ix][iy][iz]=AGG;
                }
                }
                }
        }
}


/* routine to check or perform placement of sphere of ID phasein */
/* centered at location (xin,yin,zin) of radius radd */
/* wflg=1 check for fit of sphere */
/* wflg=2 place the sphere */
/* phasein and phase2 are phases to assign to cement and cemreal images resp. */
int chksph(xin,yin,zin,radd,wflg,phasein,phase2)
        int xin,yin,zin,radd,wflg,phasein,phase2;
/* Calls: no other routines */
/* Called by: gsphere */
{
        int nofits,xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

        nofits=0;       /* Flag indicating if placement is possible */
/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        /* Perform placement */
                        if(wflg==2){
                                cement [xp] [yp] [zp]=phasein;
                                cemreal [xp][yp][zp]=phase2;
                        }
                        /* or check placement */
                        else if((wflg==1)&&(cement [xp] [yp] [zp] !=POROSITY)){
                                nofits=1;
```

```
                        }
                }
                /* Check for overlap with aggregate */
      if((wflg==1)&&((fabs(xp-((float)(SYSSIZE+1)/2.0)))<((float)aggsize/2.0))){
                        nofits=1;
                }
        }
        }
        }

        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to place spheres of various sizes and phases at random */
/* locations in 3-D microstructure */
/* numgen is number of different size spheres to place */
/* numeach holds the number of each size class */
/* sizeeach holds the radius of each size class */
/* pheach holds the phase of each size class */
void gsphere(numgen,numeach,sizeeach,pheach)
        int numgen;
        long int numeach[NUMSIZES];
        int sizeeach[NUMSIZES],pheach[NUMSIZES];
/* Calls: makesph, ran1 */
/* Called by: create */
{
        int count,x,y,z,radius,ig,tries,phnow;
        long int jg;
        float rx,ry,rz,testgyp,typegyp;
        struct cluster *partnew;

/* Generate spheres of each size class in turn (largest first) */
        for(ig=0;ig<numgen;ig++){

            phnow=pheach[ig];        /* phase for this class */
                radius=sizeeach[ig];     /* radius for this class */
                /* loop for each sphere in this size class */
                for(jg=1;jg<=numeach[ig];jg++){

                        tries=0;
                        /* Stop after MAXTRIES random tries */
                        do{
                                tries+=1;
                        /* generate a random center location for the sphere */
                                x=(int)((float)SYSSIZE*ran1(seed))+1;
                                y=(int)((float)SYSSIZE*ran1(seed))+1;
                                z=(int)((float)SYSSIZE*ran1(seed))+1;
                            /* See if the sphere will fit at x,y,z */
                            /* Include dispersion distance when checking */
                            /* to insure requested separation between spheres */
                              count=chksph(x,y,z,radius+dispdist,1,npart+CEM,0);
                                if((tries>MAXTRIES)&&(dispdist==2)){
                              tries=0;
                              dispdist+=1;
                        }
                                if(tries>MAXTRIES){
```

```
printf("Could not place sphere %d after %d random attempts \n",npart,MAXTRIES);
                                        printf("Exitting program \n");
                                        exit(1);
                                }
                        } while(count!=0);

                        /* place the sphere at x,y,z */
                        npart+=1;
                        if(npart>=NPARTC){
                                printf("Too many spheres being generated \n");
          printf("User needs to increase value of NPARTC at top of C-code\n");
                                printf("Exitting program \n");
                                exit(1);
                        }
                        /* Allocate space for new particle info */
                  clust[npart]=(struct cluster *)malloc(sizeof(struct cluster));
                        clust[npart]->partid=npart;
                        clust[npart]->clustid=npart;
                        /* Default to cement placement */
                        clust[npart]->partphase=CEMID;
                        clust[npart]->x=x;
                        clust[npart]->y=y;
                        clust[npart]->z=z;
                        clust[npart]->r=radius;
                        clusleft+=1;
                  if(phnow==1){
                        testgyp=ran1(seed);
                  /* Do not use dispersion distance when placing particle */
                        if(((testgyp>probgyp)&&((target_sulfate-n_sulfate)
<(target_total-n_total)))||(n_sulfate>target_sulfate)||(volpart[radius] >
(target_sulfate-n_sulfate))||(numeach[ig]<=2)){
                                count=chksph(x,y,z,radius,2,npart+CEM-1,CEMID);
                                n_total+=volpart[radius];
                        }
                        else{
                                /* Place particle as gypsum */
                                typegyp=ran1(seed);
                                n_total+=volpart[radius];
                                n_sulfate+=volpart[radius];

      if((probanh>=1.0)||((typegyp<probanh)&&(n_anhydrite<target_anhydrite)&&(volpa
rt[radius]<=(target_anhydrite-n_anhydrite)))){
                              /* Place particle as anhydrite */
                                        n_anhydrite+=volpart[radius];
                              count=chksph(x,y,z,radius,2,npart+CEM-1,ANHYDRITE);
                              clust[npart]->partphase=ANHYDRITE;
                        }
                        else
if(((probanh+probhem)>=1.0)||((typegyp<(probanh+probhem))&&(n_hemi<target_hemi)&&(v
olpart[radius]<=(target_hemi-n_hemi)))){
                              /* Place particle as hemihydrate */
                                        n_hemi+=volpart[radius];
                              count=chksph(x,y,z,radius,2,npart+CEM-1,HEMIHYDRATE);
                              clust[npart]->partphase=HEMIHYDRATE;
                        }
                        else{
                                count=chksph(x,y,z,radius,2,npart+CEM-1,GYPID);
```

```
                                /* Correct phase ID of particle */
                                clust[npart]->partphase=GYPID;
                        }
                        }
                  }
                  /* place as inert, CaCO3, C2S, slag, or pozzolanic material */
                  else{
                        count=chksph(x,y,z,radius,2,npart+CEM-1,phnow);
                        /* Correct phase ID of particle */
                        clust[npart]->partphase=phnow;
                  }
                        clust[npart]->nextpart=NULL;
                }
        }
}

/* routine to obtain user input and create a starting microstructure */
void create()
/* Calls: gsphere */
/* Called by: main program */
{
        int numsize,sphrad [NUMSIZES],sphase[NUMSIZES];
        long int sphnum [NUMSIZES],inval1;
        int isph,inval;

        do{
printf("Enter number of different size spheres to use(max. is %d) \n",NUMSIZES);
                scanf("%d",&numsize);
                printf("%d \n",numsize);
        }while ((numsize>NUMSIZES)||(numsize<0));
        do{
                printf("Enter dispersion factor (separation distance in pixels) for
spheres (0-2) \n");
                printf("0 corresponds to totally random placement \n");
                scanf("%d",&dispdist);
                printf("%d \n",dispdist);
        }while ((dispdist<0)||(dispdist>2));
        do{
                printf("Enter probability for gypsum particles on a random particle
basis (0.0-1.0) \n");
                scanf("%f",&probgyp);
                printf("%f \n",probgyp);
        } while ((probgyp<0.0)||(probgyp>1.0));
      do{
            printf("Enter probabilities for hemihydrate and anhydrite forms of
gypsum (0.0-1.0) \n");
            scanf("%f %f",&probhem,&probanh);
            printf("%f %f\n",probhem,probanh);
      } while
((probhem<0.0)||(probhem>1.0)||(probanh<0.0)||(probanh>1.0)||((probanh+probhem)>1.0
01));

        if((numsize>0)&&(numsize<(NUMSIZES+1))){
       printf("Enter number, radius, and phase ID for each sphere class (largest
radius 1st) \n");
      printf("Phases are %d- Cement and (random) calcium sulfate, %d- C2S, %d-
Gypsum, %d- hemihydrate %d- anhydrite %d- Pozzolanic, %d- Inert, %d- Slag, %d-
```

```
CaCO3 %d- Fly Ash
\n",CEMID,C2SID,GYPID,HEMIHYDRATE,ANHYDRITE,POZZID,INERTID,SLAGID,CACO3,FLYASH);

        /* Obtain input for each size class of spheres */
                for(isph=0;isph<numsize;isph++){
                        printf("Enter number of spheres of class %d \n",isph+1);
                        scanf("%ld",&inval1);
                        printf("%ld \n",inval1);
                        sphnum[isph]=inval1;
                        do{
                        printf("Enter radius of spheres of class %d \n",isph+1);
                                printf("(Integer <=%d please) \n",SYSSIZE/3);
                                scanf("%d",&inval);
                                printf("%d \n",inval);
                        } while ((inval<1)||(inval>(SYSSIZE/3)));
                        sphrad[isph]=inval;
                        do{
                        printf("Enter phase of spheres of class %d \n",isph+1);
                                scanf("%d",&inval);
                                printf("%d \n",inval);
                        } while
((inval!=CEMID)&&(inval!=C2SID)&&(inval!=GYPID)&&(inval!=HEMIHYDRATE)&&(inval!=ANHY
DRITE)&&(inval!=POZZID)&&(inval!=INERTID)&&(inval!=SLAGID)&&(inval!=FLYASH)&&(inval
!=CACO3));
                        sphase[isph]=inval;
                  if(inval==CEMID){
                        target_total+=sphnum[isph]*volpart[sphrad[isph]];
                  }

                }
            /* Determine target pixel counts for calcium sulfate forms */
                target_sulfate=(int)((float)target_total*probgyp);
            target_anhydrite=(int)((float)target_total*probgyp*probanh);
            target_hemi=(int)((float)target_total*probgyp*probhem);
                gsphere(numsize,sphnum,sphrad,sphase);
        }
}

/* Routine to draw a particle during flocculation routine */
/* See routine chksph for definition of parameters */
void drawfloc(xin,yin,zin,radd,phasein,phase2)
        int xin,yin,zin,radd,phasein,phase2;
/* Calls: no other routines */
/* Called by: makefloc */
{
        int xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;(i<=xin+radd);i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;(j<=yin+radd);j++){
```

```
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;(k<=zin+radd);k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        /* Update both cement and cemreal images */
                        cement [xp] [yp] [zp]=phasein;
                        cemreal [xp] [yp] [zp]=phase2;
                }
        }
        }
        }
}

/* Routine to check particle placement during flocculation */
/* for particle of size radd centered at (xin,yin,zin) */
/* Returns flag indicating if placement is possible */
int chkfloc(xin,yin,zin,radd)
        int xin,yin,zin,radd;
/* Calls: no other routines */
/* Called by: makefloc */
{
        int nofits,xp,yp,zp,i,j,k;
        float dist,xdist,ydist,zdist,ftmp;

        nofits=0;       /* Flag indicating if placement is possible */

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
                xp=i;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                else if(xp>SYSSIZE) {xp-=SYSSIZE;}
                ftmp=(float)(i-xin);
                xdist=ftmp*ftmp;
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
                yp=j;
                /* use periodic boundary conditions for sphere placement */
                if(yp<1) {yp+=SYSSIZE;}
                else if(yp>SYSSIZE) {yp-=SYSSIZE;}
                ftmp=(float)(j-yin);
                ydist=ftmp*ftmp;
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){
                zp=k;
                /* use periodic boundary conditions for sphere placement */
```

```
                if(zp<1) {zp+=SYSSIZE;}
                else if(zp>SYSSIZE) {zp-=SYSSIZE;}
                ftmp=(float)(k-zin);
                zdist=ftmp*ftmp;

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt(xdist+ydist+zdist);
                if((dist-0.5)<=(float)radd){
                        if((cement [xp] [yp] [zp] !=POROSITY)){
                                /* Record ID of particle hit */
                                nofits=cement [xp] [yp] [zp];
                        }
                }
                /* Check for overlap with aggregate */
                if((fabs(xp-((float)(SYSSIZE+1)/2.0)))<((float)aggsize/2.0)){
                        nofits=AGG;
                }
        }
        }
        }
        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to perform flocculation of particles */
void makefloc()
/* Calls: drawfloc, chkfloc, ran1 */
/* Called by: main program */
{
        int partdo,numfloc;
        int nstart;
        int nleft,ckall;
        int xm,ym,zm,moveran;
        int xp,yp,zp,rp,clushit,valkeep;
        int iclus,cluspart[NPARTC];
        struct cluster *parttmp,*partpoint,*partkeep;

        nstart=npart; /* Counter of number of flocs remaining */
        for(iclus=1;iclus<=npart;iclus++){
                  cluspart[iclus]=iclus;
        }
        do{
          printf("Enter number of flocs desired at end of routine (>0) \n");
                scanf("%d",&numfloc);
                printf("%d\n",numfloc);
        } while (numfloc<=0);

        while(nstart>numfloc){
                nleft=0;

                /* Try to move each cluster in turn */
                for(iclus=1;iclus<=npart;iclus++){
                        if(clust[iclus]==NULL){
                                nleft+=1;
                        }
                        else{
                                xm=ym=zm=0;
```

```
/* Generate a random move in one of 6 principal directions */
                moveran=6.*ran1(seed);
switch(moveran){
                case 0:
                        xm=1;
                        break;
                case 1:
                        xm=(-1);
                        break;
                case 2:
                        ym=1;
                        break;
                case 3:
                        ym=(-1);
                        break;
                case 4:
                        zm=1;
                        break;
                case 5:
                        zm=(-1);
                        break;
                default:
                        break;
}

/* First erase all particles in cluster */
partpoint=clust[iclus];
while(partpoint!=NULL){
        xp=partpoint->x;
        yp=partpoint->y;
        zp=partpoint->z;
        rp=partpoint->r;
        drawfloc(xp,yp,zp,rp,0,0);
        partpoint=partpoint->nextpart;
}

ckall=0;
/* Now try to draw cluster at new location */
partpoint=clust[iclus];
while((partpoint!=NULL)&&(ckall==0)){
        xp=partpoint->x+xm;
        yp=partpoint->y+ym;
        zp=partpoint->z+zm;
        rp=partpoint->r;
        ckall=chkfloc(xp,yp,zp,rp);
        partpoint=partpoint->nextpart;
}

if(ckall==0){
/* Place cluster particles at new location */
        partpoint=clust[iclus];
        while(partpoint!=NULL){
                xp=partpoint->x+xm;
                yp=partpoint->y+ym;
                zp=partpoint->z+zm;
                rp=partpoint->r;
                valkeep=partpoint->partphase;
```

```
                                partdo=partpoint->partid;
                                drawfloc(xp,yp,zp,rp,partdo+CEM-1,valkeep);
                                /* Update particle location */
                                partpoint->x=xp;
                                partpoint->y=yp;
                                partpoint->z=zp;
                                partpoint=partpoint->nextpart;
                        }
                }
                else{
                        /* A cluster or aggregate was hit */
                        /* Draw particles at old location */
                        partpoint=clust[iclus];
                        /* partkeep stores pointer to last particle in list */
                        while(partpoint!=NULL){
                                xp=partpoint->x;
                                yp=partpoint->y;
                                zp=partpoint->z;
                                rp=partpoint->r;
                                valkeep=partpoint->partphase;
                                partdo=partpoint->partid;
                                drawfloc(xp,yp,zp,rp,partdo+CEM-1,valkeep);
                                partkeep=partpoint;
                                partpoint=partpoint->nextpart;
                        }
                        /* Determine the cluster hit */
                        if(ckall!=AGG){
                                clushit=cluspart[ckall-CEM+1];
      /* Move all of the particles from cluster clushit to cluster iclus */
                                parttmp=clust[clushit];
                                /* Attach new cluster to old one */
                                partkeep->nextpart=parttmp;
                                while(parttmp!=NULL){
                                        cluspart[parttmp->partid]=iclus;
                          /* Relabel all particles added to this cluster */
                                        parttmp->clustid=iclus;
                                        parttmp=parttmp->nextpart;
                                }
                                /* Disengage the cluster that was hit */
                                clust[clushit]=NULL;
                                nstart-=1;
                        }
                }
                }
        }
  printf("Number left was %d but number of clusters is %d \n",nleft,nstart);
        }      /* end of while loop */
        clusleft=nleft;
}

/* routine to assess global phase fractions present in 3-D system */
void measure()
/* Calls: no other routines */
/* Called by: main program */
{
     long int npor,nc2s,ngyp,ncem,nagg,npozz,ninert,nflyash,nanh,nhem,ncaco3,nslag;
     int i,j,k,valph;
```

```
/* counters for the various phase fractions */
      npor=0;
      ngyp=0;
      ncem=0;
      nagg=0;
      ninert=0;
      nslag=0;
      nc2s=0;
      npozz=0;
      nflyash=0;
      nanh=0;
      nhem=0;
      ncaco3=0;

/* Check all pixels in 3-D microstructure */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){
                valph=cemreal [i] [j] [k];
                if(valph==POROSITY) {npor+=1;}
                else if(valph==CEMID){ncem+=1;}
                else if(valph==C2SID){nc2s+=1;}
                else if(valph==GYPID){ngyp+=1;}
                else if(valph==ANHYDRITE){nanh+=1;}
                else if(valph==HEMIHYDRATE){nhem+=1;}
                else if(valph==AGG) {nagg+=1;}
                else if(valph==POZZID) {npozz+=1;}
                else if(valph==SLAGID) {nslag+=1;}
                else if(valph==INERTID) {ninert+=1;}
                else if(valph==FLYASH) {nflyash+=1;}
                else if(valph==CACO3) {ncaco3+=1;}
        }
        }
        }

/* Output results */
      printf("\n Phase counts are: \n");
      printf("Porosity= %ld \n",npor);
      printf("Cement= %ld \n",ncem);
      printf("C2S= %ld \n",nc2s);
      printf("Gypsum= %ld \n",ngyp);
      printf("Anhydrite= %ld \n",nanh);
      printf("Hemihydrate= %ld \n",nhem);
      printf("Pozzolan= %ld \n",npozz);
      printf("Inert= %ld \n",ninert);
      printf("Slag= %ld \n",nslag);

      printf("CaCO3= %ld \n",ncaco3);
      printf("Fly Ash= %ld \n",nflyash);
      printf("Aggregate= %ld \n",nagg);
}

/* Routine to measure phase fractions as a function of distance from */
/* aggregate surface                                                 */
void measagg()
/* Calls: no other routines */
```

```
/* Called by: main program */
{
        int phase [40],ptot;
        int icnt,ixlo,ixhi,iy,iz,phid,idist;
        FILE *aggfile;

      /* By default, results are sent to output file called agglist.out */
      aggfile=fopen("agglist.out","w");


      printf("Distance  Porosity  Cement C2S  Gypsum  Anhydrite Hemihydrate
Pozzolan  Inert   Slag CaCO3  Fly Ash\n");
      fprintf(aggfile,"Distance  Porosity  Cement  C2S Gypsum  Anhydrite
Hemihydrate Pozzolan  Inert  Slag  CaCO3   Fly Ash\n");


/* Increase distance from aggregate in increments of one */
        for(idist=1;idist<=(SYSSIZE-aggsize)/2;idist++){
                /* Pixel left of aggregate surface */
                ixlo=((SYSSIZE-aggsize+2)/2)-idist;
                /* Pixel right of aggregate surface */
                ixhi=((SYSSIZE+aggsize)/2)+idist;

        /* Initialize phase counts for this distance */
                for(icnt=0;icnt<39;icnt++){
                        phase[icnt]=0;
                }
                ptot=0;

/* Check all pixels which are this distance from aggregate surface */
                for(iy=1;iy<=SYSSIZE;iy++){
                for(iz=1;iz<=SYSSIZE;iz++){
                        phid=cemreal [ixlo] [iy] [iz];
                        ptot+=1;
                        if(phid<=FLYASH){
                                phase[phid]+=1;
                        }
                        phid=cemreal [ixhi] [iy] [iz];
                        ptot+=1;
                        if(phid<=FLYASH){
                                phase[phid]+=1;
                        }
                }
                }

                /* Output results for this distance from surface */
     printf("%d   %d   %d  %d  %d %d %d  %d %d\n",idist,phase[0],phase[CEMID],
phase[C2SID],phase[GYPID],phase[ANHYDRITE],phase[HEMIHYDRATE],phase[POZZID],
phase[INERTID],phase[SLAGID],phase[CACO3],phase[FLYASH]);
     fprintf(aggfile,"%d   %d   %d  %d  %d %d %d  %d %d\n",idist,phase[0],
phase[CEMID],phase[C2SID],phase[GYPID],phase[ANHYDRITE],phase[HEMIHYDRATE],
phase[POZZID],phase[INERTID],phase[SLAGID],phase[CACO3],phase[FLYASH]);

         }
      fclose(aggfile);
}
```

```
/* routine to assess the connectivity (percolation) of a single phase */
/* Two matrices are used here: one for the current burnt locations */
/*                the other to store the newly found burnt locations */
void connect()
/* Calls: no other routines */
/* Called by: main program */
{
     long int inew,ntop,nthrough,ncur,nnew,ntot;
     int i,j,k,nmatx[29000],nmaty[29000],nmatz[29000];
     int xcn,ycn,zcn,npix,x1,y1,z1,igood,nnewx[29000],nnewy[29000],nnewz[29000];
     int jnew,icur;

        do{
                printf("Enter phase to analyze 0) pores 1) Cement  \n");
                scanf("%d",&npix);
                printf("%d \n",npix);
        } while ((npix!=0)&&(npix!=1));

        /* counters for number of pixels of phase accessible from top surface */
        /* and number which are part of a percolated pathway */
        ntop=0;
        nthrough=0;

        /* percolation is assessed from top to bottom only */
        /* and burning algorithm is periodic in x and y directions */
        k=1;
        for(i=1;i<=SYSSIZE;i++){
                for(j=1;j<=SYSSIZE;j++){
                        ncur=0;
                        ntot=0;
                        igood=0;     /* Indicates if bottom has been reached */
if(((cement [i] [j] [k]==npix)&&((cement [i] [j] [SYSSIZE]==npix)||
(cement [i] [j] [SYSSIZE]==(npix+BURNT))))||((cement [i] [j] [SYSSIZE]>=CEM)&&
(cement [i] [j] [k]>=CEM)&&(cement [i] [j] [k]<BURNT)&&(npix==1))){
                                /* Start a burn front */
                                cement [i] [j] [k]+=BURNT;
                                ntot+=1;
                                ncur+=1;
                                /* burn front is stored in matrices nmat* */
                                /* and nnew* */
                                nmatx[ncur]=i;
                                nmaty[ncur]=j;
                                nmatz[ncur]=1;
                            /* Burn as long as new (fuel) pixels are found */
                                do{
                                        nnew=0;
                                        for(inew=1;inew<=ncur;inew++){
                                               xcn=nmatx[inew];
                                               ycn=nmaty[inew];
                                               zcn=nmatz[inew];

                                               /* Check all six neighbors */
                                               for(jnew=1;jnew<=6;jnew++){
                                                       x1=xcn;
                                                       y1=ycn;
                                                       z1=zcn;
                                                       if(jnew==1){
```

```
                                                        x1-=1;
                                                        if(x1<1){
                                                            x1+=SYSSIZE;
                                                        }
                                                    }
                                                    else if(jnew==2){
                                                        x1+=1;
                                                        if(x1>SYSSIZE){
                                                            x1-=SYSSIZE;
                                                        }
                                                    }
                                                    else if(jnew==3){
                                                        y1-=1;
                                                        if(y1<1){
                                                            y1+=SYSSIZE;
                                                        }
                                                    }
                                                    else if(jnew==4){
                                                        y1+=1;
                                                        if(y1>SYSSIZE){
                                                            y1-=SYSSIZE;
                                                        }
                                                    }
                                                    else if(jnew==5){
                                                        z1-=1;
                                                    }
                                                    else if(jnew==6){
                                                        z1+=1;
                                                    }

/* Nonperiodic in z direction so be sure to remain in the 3-D box */
                                                if((z1>=1)&&(z1<=SYSSIZE)){
if((cement [x1] [y1] [z1]==npix)||((cement [x1] [y1] [z1]>=CEM)&&
(cement [x1] [y1] [z1]<BURNT)&&(npix==1))){
                                                        ntot+=1;
                                                  cement [x1] [y1] [z1]+=BURNT;
                                                        nnew+=1;
                                                    if(nnew>=29000){
                                              printf("error in size of nnew \n");
                                                    }
                                                        nnewx[nnew]=x1;
                                                        nnewy[nnew]=y1;
                                                        nnewz[nnew]=z1;
                                    /* See if bottom of system has been reached */
                                                    if(z1==SYSSIZE){
                                                        igood=1;
                                                    }
                                                    }
                                                }
                                                }
                                            }
                                            if(nnew>0){
                                                ncur=nnew;
                                            /* update the burn front matrices */
                                                for(icur=1;icur<=ncur;icur++){
                                                    nmatx[icur]=nnewx[icur];
                                                    nmaty[icur]=nnewy[icur];
```

```
                                                        nmatz[icur]=nnewz[icur];
                                                }
                                        }
                                }while (nnew>0);

                                ntop+=ntot;
                                if(igood==1){
                                        nthrough+=ntot;
                                }
                        }
                }
        }

        printf("Phase ID= %d \n",npix);
        printf("Number accessible from top= %ld \n",ntop);
        printf("Number contained in through pathways= %ld \n",nthrough);

        /* return the burnt sites to their original phase values */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){
                if(cement [i] [j] [k]>=BURNT){
                        cement [i] [j] [k]-=BURNT;
                }
        }
        }
        }
}

/* Routine to output final microstructure to file */
void outmic()
/* Calls: no other routines */
/* Called by: main program */
{
        FILE *outfile,*partfile;
        char filen[80],filepart[80];
        int ix,iy,iz,valout;

        printf("Enter name of file to save microstructure to \n");
        scanf("%s",filen);
        printf("%s\n",filen);

        outfile=fopen(filen,"w");

        printf("Enter name of file to save particle IDs to \n");
        scanf("%s",filepart);
        printf("%s\n",filepart);

        partfile=fopen(filepart,"w");

        for(iz=1;iz<=SYSSIZE;iz++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(ix=1;ix<=SYSSIZE;ix++){
                valout=cemreal[ix][iy][iz];
                fprintf(outfile,"%1d\n",valout);
                valout=cement[ix][iy][iz];
                if(valout<0){valout=0;}
```

```
                fprintf(partfile,"%d\n",valout);
        }
        }
        }
        fclose(outfile);
        fclose(partfile);
}

main(){
      int userc;      /* User choice from menu */
      int nseed,ig,jg,kg;

      /* Initialize volume array */
      volpart[0]=1;
      volpart[1]=19;
      volpart[2]=81;
      volpart[3]=179;
      volpart[4]=389;
      volpart[5]=739;
      volpart[6]=1189;
      volpart[7]=1791;
      volpart[8]=2553;
      volpart[9]=3695;
      volpart[10]=4945;
      volpart[11]=6403;
      volpart[12]=8217;
      volpart[13]=10395;
      volpart[14]=12893;
      volpart[15]=15515;
      volpart[16]=18853;
      volpart[17]=22575;
      volpart[18]=26745;
      volpart[19]=31103;
      volpart[20]=36137;
      volpart[21]=41851;
      volpart[22]=47833;
      volpart[23]=54435;
      volpart[24]=61565;
      volpart[25]=69599;
      volpart[26]=78205;
      volpart[27]=87271;
      volpart[28]=97233;
      volpart[29]=107783;
      volpart[30]=119009;
      volpart[31]=131155;
      volpart[32]=143761;
      volpart[33]=157563;
      volpart[34]=172317;
      volpart[35]=187511;
      volpart[36]=203965;

        printf("Enter random number seed value (a negative integer) \n");
        scanf("%d",&nseed);
        printf("%d \n",nseed);
        seed=(&nseed);
```

```
        /* Initialize counters and system parameters */
        npart=0;
        aggsize=0;
        clusleft=0;

        /* clear the 3-D system to all porosity to start */
        for(ig=1;ig<=SYSSIZE;ig++){
        for(jg=1;jg<=SYSSIZE;jg++){
        for(kg=1;kg<=SYSSIZE;kg++){
                cement [ig] [jg] [kg]=POROSITY;
                cemreal [ig] [jg] [kg]=POROSITY;
        }
        }
        }

        /* present menu and execute user choice */
        do{
printf(" \n Input User Choice \n");
printf("1) Exit \n");
printf("2) Add spherical particles (cement,gypsum, pozzolans, etc.) to
microstructure \n");
printf("3) Flocculate system by reducing number of particle clusters \n");
printf("4) Measure global phase fractions \n");
printf("5) Add an aggregate to the microstructure \n");
printf("6) Measure single phase connectivity (pores or solids) \n");
printf("7) Measure phase fractions vs. distance from aggregate surface \n");
printf("8) Output current microstructure to file \n");

                scanf("%d",&userc);
                printf("%d \n",userc);
                fflush(stdout);

                switch (userc) {
                        case 2:
                                create();
                                break;
                        case 3:
                                makefloc();
                                break;
                        case 4:
                                measure();
                                break;
                        case 5:
                                addagg();
                                break;
                        case 6:
                                connect();
                                break;
                        case 7:
                                if(aggsize!=0){
                                      measagg();
                        }
                        else{
                              printf("No aggregate present. \n");
                        }
                                break;
                        case 8:
```

```
                                outmic();
                                break;
                        default:
                                break;
                }
        } while (userc!=1);
}
```

Program distrib3d.c

```
/*************************************************************************/
/*                                                                       */
/*      Program distrib3d.c to distribute phases amongst three-          */
/*              dimensional cement particles in a 3-D box with           */
/*              periodic boundaries.                                     */
/*      Programmer: Dale P. Bentz                                        */
/*                  Building and Fire Research Laboratory                */
/*                  NIST                                                 */
/*                  100 Bureau Drive Mail Stop 8615                      */
/*                  Gaithersburg, MD  20899-8615   USA                   */
/*                  (301) 975-5865      FAX: 301-990-6891                */
/*                  E-mail: dale.bentz@nist.gov                          */
/*                                                                       */
/*************************************************************************/
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

/* Random number generator */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b
#define PI 3.1415926
#define SYSIZE 100
#define SYSSIZE 100
#define MAXCYC 1000    /* maximum sintering cycles to use */
#define MAXSPH 10000   /* maximum number of elements in a spherical template */
#define C3S 1
#define C2S 2
#define C3A 3
#define C4AF 4

int *seed;
```

```
static int mask[SYSIZE+1][SYSIZE+1][SYSIZE+1];
static unsigned short int curvature [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
long int volume[50],surface[50];
int nsph,xsph[MAXSPH],ysph[MAXSPH],zsph[MAXSPH];
long int nsolid[1500],nair[1500];

double ran1(idum)
int *idum;
{
        int j,k;
      static int iv[NTAB],iy=0;
      void nrerror();
        static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
        static double RNMX = (1.0-EPS);
        static double AM = (1.0/IM);

      if ((*idum <= 0) || (iy == 0)) {
            *idum = MAX(-*idum,*idum);
                for(j=NTAB+7;j>=0;j--) {
                  k = *idum/IQ;
                  *idum = IA*(*idum-k*IQ)-IR*k;
                  if(*idum < 0) *idum += IM;
                  if(j < NTAB) iv[j] = *idum;
            }
            iy = iv[0];
      }
      k = *idum/IQ;
      *idum = IA*(*idum-k*IQ)-IR*k;
      if(*idum<0) *idum += IM;
      j = iy*NDIV;
      iy = iv[j];
      iv[j] = *idum;
      return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN


/* routine to create a template for the sphere of interest of radius size */
/* to be used in curvature evaluation */
/* Called by: runsint */
/* Calls no other routines */
int maketemp(size)
        int size;
{
        int icirc,xval,yval,zval;

        float xtmp,ytmp;
        float dist;

/* determine and store the locations of all pixels in the 3-D sphere */
```

```
        icirc=0;
        for(xval=(-size);xval<=size;xval++){
                xtmp=(float)(xval*xval);
        for(yval=(-size);yval<=size;yval++){
                ytmp=(float)(yval*yval);
        for(zval=(-size);zval<=size;zval++){
                dist=sqrt(xtmp+ytmp+(float)(zval*zval));
                if(dist<=((float)size+0.5)){
                        icirc+=1;
                        if(icirc>=MAXSPH){
                            printf("Too many elements in sphere \n");
                            printf("Must change value of MAXSPH parameter \n");
                            printf("Currently set at %d \n",MAXSPH);
                            exit(1);
                        }
                        xsph[icirc]=xval;
                        ysph[icirc]=yval;
                        zsph[icirc]=zval;
                }
        }
        }
        }

/* return the number of pixels contained in sphere of radius (size+0.5) */
        return(icirc);
}

/* routine to count phase fractions (porosity and solids) */
/* Called by main routine */
/* Calls no other routines */
void phcount()
{
        long int npore,nsolid [37];
        int ix,iy,iz;

        npore=0;
        for(ix=1;ix<37;ix++){
            nsolid[ix]=0;
      }
        /* check all pixels in the 3-D system */
        for(ix=1;ix<=SYSSIZE;ix++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(iz=1;iz<=SYSSIZE;iz++){
                if(mask [ix] [iy] [iz]==0){
                        npore+=1;
                  }
                else{
                        nsolid[mask [ix] [iy] [iz]]+=1;
                  }
        }
        }
        }

        printf("Pores are: %ld \n",npore);
        printf("Solids are: %ld %ld %ld %ld %ld %ld\n",nsolid[1],nsolid[2],
             nsolid[3],nsolid[4],nsolid[5],nsolid[6]);
}
```

```
/* routine to return number of surface faces exposed to porosity */
/* for pixel located at (xin,yin,zin) */
/* Called by rhcalc */
/* Calls no other routines */
int surfpix(xin,yin,zin)
        int xin,yin,zin;
{
        int npix,ix1,iy1,iz1;

        npix=0;

        /* check each of the six immediate neighbors */
        /* using periodic boundary conditions */
        ix1=xin-1;
        if(ix1<1){ix1+=SYSSIZE;}
        if(mask[ix1][yin][zin]==0){
                npix+=1;
        }
        ix1=xin+1;
        if(ix1>SYSSIZE){ix1-=SYSSIZE;}
        if(mask[ix1][yin][zin]==0){
                  npix+=1;
        }
        iy1=yin-1;
        if(iy1<1){iy1+=SYSSIZE;}
        if(mask[xin][iy1][zin]==0){
                npix+=1;
        }
        iy1=yin+1;
        if(iy1>SYSSIZE){iy1-=SYSSIZE;}
        if(mask[xin][iy1][zin]==0){
                npix+=1;
        }
        iz1=zin-1;
        if(iz1<1){iz1+=SYSSIZE;}
        if(mask[xin][yin][iz1]==0){
                npix+=1;
        }
            iz1=zin+1;
            if(iz1>SYSSIZE){iz1-=SYSSIZE;}
            if(mask[xin][yin][iz1]==0){
                npix+=1;
        }
        return(npix);
}

/* routine to return the current hydraulic radius for phase phin */
/* Calls surfpix */
/* Called by runsint */
float rhcalc(phin)
        int phin;
{
        int ix,iy,iz;
        long int porc,surfc;
        float rhval;
```

```
        porc=surfc=0;

        /* Check all pixels in the 3-D volume */
        for(ix=1;ix<=SYSSIZE;ix++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(iz=1;iz<=SYSSIZE;iz++){

                if(mask [ix] [iy] [iz]==phin){
                        porc+=1;
                        surfc+=surfpix(ix,iy,iz);
                }
        }
        }
        }

        printf("Phase area count is %ld \n",porc);
        printf("Phase surface count is %ld \n",surfc);
        rhval=(float)porc*6./(4.*(float)surfc);
        printf("Hydraulic radius is %f \n",rhval);
        return(rhval);
}

/* routine to return count of pixels in a spherical template which are phase */
/* phin or porosity (phase=0) */
/* Calls no other routines */
/* Called by sysinit */
int countem(xp,yp,zp,phin)
        int xp,yp,zp,phin;
{
        int xc,yc,zc;
        int cumnum,ic;

        cumnum=0;
        for(ic=1;ic<=nsph;ic++){
                xc=xp+xsph[ic];
                yc=yp+ysph[ic];
                zc=zp+zsph[ic];
                /* Use periodic boundaries */
                if(xc<1){xc+=SYSSIZE;}
                else if(xc>SYSSIZE){xc-=SYSSIZE;}
                if(yc<1){yc+=SYSSIZE;}
                else if(yc>SYSSIZE){yc-=SYSSIZE;}
                if(zc<1){zc+=SYSSIZE;}
                else if(zc>SYSSIZE){zc-=SYSSIZE;}

                if((xc!=xp)||(yc!=yp)||(zc!=zp)){

                if((mask [xc] [yc] [zc]==phin)||(mask [xc] [yc] [zc]==0)){
                        cumnum+=1;
                }
                  }
        }
        return(cumnum);
}

/* routine to initialize system by determining local curvature */
/* of all phase 1 and phase 2 pixels */
```

```
/* Calls countem */
/* Called by runsint */
void sysinit(ph1,ph2)
        int ph1,ph2;
{
        int count,xl,yl,zl;

            count=0;
            /* process all pixels in the 3-D box */
            for(xl=1;xl<=SYSSIZE;xl++){
        for(yl=1;yl<=SYSSIZE;yl++){
        for(zl=1;zl<=SYSSIZE;zl++){

                /* determine local curvature */
                /* For phase 1 want to determine number of porosity pixels */
                /* (phase=0) in immediate neighborhood */
                if(mask [xl] [yl] [zl]==ph1){
                        count=countem(xl,yl,zl,0);
                }
                /* For phase 2 want to determine number of porosity or phase */
                /* 2 pixels in immediate neighborhood */
                if(mask [xl] [yl] [zl]==ph2){
                        count=countem(xl,yl,zl,ph2);
                }
                if((count<0)||(count>=nsph)){
                        printf("Error count is %d \n",count);
                        printf("xl %d  yl %d  zl %d \n",xl,yl,zl);
                }

                /* case where we have a phase 1 surface pixel */
                /* with non-zero local curvature */
                if((count>=0)&&(mask [xl] [yl] [zl]==ph1)){

                        curvature [xl] [yl] [zl]=count;
                        /* update solid curvature histogram */
                        nsolid[count]+=1;
                }

                /* case where we have a phase 2 surface pixel */
                if((count>=0)&&(mask [xl] [yl] [zl]==ph2)){

                        curvature [xl] [yl] [zl]=count;
                        /* update air curvature histogram */
                        nair[count]+=1;
                }

        }
        }
            }     /* end of xl loop */
}

/* routine to scan system and determine nsolid (ph2) and nair (ph1) */
/* histograms based on values in phase and curvature arrays */
/* Calls no other routines */
/* Called by runsint */
void sysscan(ph1,ph2)
        int ph1,ph2;
```

```
{
        int xd,yd,zd,curvval;

        /* Scan all pixels in 3-D system */
        for(xd=1;xd<=SYSSIZE;xd++){
        for(yd=1;yd<=SYSSIZE;yd++){
        for(zd=1;zd<=SYSSIZE;zd++){

                curvval=curvature [xd] [yd] [zd];

                if(mask [xd] [yd] [zd]==ph2){
                        nair[curvval]+=1;
                }
                else if (mask [xd] [yd] [zd]==ph1){
                        nsolid[curvval]+=1;
                }
        }
        }
        }
}

/* routine to return how many cells of solid curvature histogram to use */
/* to accomodate nsearch pixels moving */
/* want to use highest values first */
/* Calls no other routines */
/* Called by movepix */
int procsol(nsearch)
        int nsearch;
{
        int valfound,i,stop;
        long int nsofar;

        /* search histogram from top down until cumulative count */
        /* exceeds nsearch */
        valfound=nsph-1;
        nsofar=0;
        stop=0;
        for(i=(nsph-1);((i>=0)&&(stop==0));i--){
                nsofar+=nsolid[i];
                if(nsofar>nsearch){
                        valfound=i;
                        stop=1;
                }
        }
        return(valfound);
}

/* routine to determine how many cells of air curvature histogram to use */
/* to accomodate nsearch moving pixels */
/* want to use lowest values first */
/* Calls no other routines */
/* Called by movepix */
int procair(nsearch)
        int nsearch;
{
        int valfound,i,stop;
        long int nsofar;
```

```
        /* search histogram from bottom up until cumulative count */
        /* exceeds nsearch */
        valfound=0;
        nsofar=0;
        stop=0;
        for(i=0;((i<nsph)&&(stop==0));i++){
                nsofar+=nair[i];
                  if(nsofar>nsearch){
                        valfound=i;
                        stop=1;
                }
        }
            return(valfound);
}

/* routine to move requested number of pixels (ntomove) from highest */
/* curvature phase 1 (ph1) sites to lowest curvature phase 2 (ph2) sites */
/* Calls procsol and procair */
/* Called by runsint */
int movepix(ntomove,ph1,ph2)
        int ntomove,ph1,ph2;
{
        int xloc[2100],yloc[2100],zloc[2100];
        int count1,count2,ntot,countc,i,xp,yp,zp;
        int cmin,cmax,cfg;
        int alldone;
        long int nsolc,nairc,nsum,nsolm,nairm,nst1,nst2,next1,next2;
        float pck,plsol,plair;

        alldone=0;
        /* determine critical values for removal and placement */
        count1=procsol(ntomove);
        nsum=0;
        cfg=0;
        cmax=count1;
        for(i=nsph;i>count1;i--){
                if((nsolid[i]>0)&&(cfg==0)){
                        cfg=1;
                        cmax=i;
                }
                  nsum+=nsolid[i];
        }
        /* Determine movement probability for last cell */
        plsol=(float)(ntomove-nsum)/(float)nsolid[count1];
        next1=ntomove-nsum;
        nst1=nsolid[count1];

        count2=procair(ntomove);
        nsum=0;
        cmin=count2;
        cfg=0;
        for(i=0;i<count2;i++){
                if((nair[i]>0)&&(cfg==0)){
                        cfg=1;
                        cmin=i;
                }
```

```
          nsum+=nair[i];
}
/* Determine movement probability for last cell */
plair=(float)(ntomove-nsum)/(float)nair[count2];
next2=ntomove-nsum;
nst2=nair[count2];

/* Check to see if equilibrium has been reached --- */
/* no further increase in hydraulic radius is possible */
if(cmin>=cmax){
        alldone=1;
        printf("Stopping - at equilibrium \n");
        printf("cmin- %d  cmax- %d \n",cmin,cmax);
        return(alldone);
}

/* initialize counters for performing sintering */
ntot=0;
nsolc=0;
nairc=0;
nsolm=0;
nairm=0;

/* Now process each pixel in turn */
for(xp=1;xp<=SYSSIZE;xp++){
for(yp=1;yp<=SYSSIZE;yp++){
for(zp=1;zp<=SYSSIZE;zp++){

        countc=curvature [xp] [yp] [zp];
        /* handle phase 1 case first */
        if(mask [xp] [yp] [zp]==ph1){
                if(countc>count1){
                        /* convert from phase 1 to phase 2 */
                        mask [xp] [yp] [zp]=ph2;

                        /* update appropriate histogram cells */
                        nsolid[countc]-=1;
                        nair[countc]+=1;
                        /* store the location of the modified pixel */
                        ntot+=1;
                        xloc[ntot]=xp;
                        yloc[ntot]=yp;
                        zloc[ntot]=zp;
                }
                if(countc==count1){
                        nsolm+=1;
                     /* generate probability for pixel being removed */
                        pck=ran1(seed);
                        if((pck<0)||(pck>1.0)){pck=1.0;}

         if(((pck<plsol)&&(nsolc<next1))||((nst1-nsolm)<(next1-nsolc))){
                                nsolc+=1;
                                /* convert phase 1 pixel to phase 2 */
                                mask [xp] [yp] [zp]=ph2;

                                /* update appropriate histogram cells */
                                nsolid[count1]-=1;
```

```
                                        nair[count1]+=1;
                                    /* store the location of the modified pixel */
                                        ntot+=1;
                                        xloc[ntot]=xp;
                                        yloc[ntot]=yp;
                                        zloc[ntot]=zp;
                                    }
                                    }
                            }

                            /* handle phase 2 case here */
                            else if (mask [xp] [yp] [zp]==ph2){
                                    if(countc<count2){
                                        /* convert phase 2 pixel to phase 1 */
                                        mask [xp] [yp] [zp]=ph1;

                                        nsolid[countc]+=1;
                                        nair[countc]-=1;
                                        ntot+=1;
                                        xloc[ntot]=xp;
                                        yloc[ntot]=yp;
                                        zloc[ntot]=zp;
                                    }
                                    if(countc==count2){
                                        nairm+=1;
                                        pck=ran1(seed);
                                        if((pck<0)||(pck>1.0)){pck=1.0;}

                if(((pck<plair)&&(nairc<next2))||((nst2-nairm)<(next2-nairc))){
                                                nairc+=1;
                                                /* convert phase 2 to phase 1 */
                                                mask [xp] [yp] [zp]=ph1;

                                                nsolid[count2]+=1;
                                                nair[count2]-=1;
                                                ntot+=1;
                                                xloc[ntot]=xp;
                                                yloc[ntot]=yp;
                                                zloc[ntot]=zp;
                                        }
                                    }
                            }

                } /* end of zp loop */
                } /* end of yp loop */
                    } /* end of xloop */
                printf("ntot is %d \n",ntot);
                    return(alldone);
}

/* routine to execute user input number of cycles of sintering algorithm */
/* Calls maketemp, rhcalc, sysinit, sysscan, and movepix */
/* Called by main routine */

void sinter3d(ph1id,ph2id,rhtarget)
      int ph1id,ph2id;
      float rhtarget;
```

```
{
        int natonce,ncyc,i,rade,j,rflag;
        int keepgo;
        long int curvsum1,curvsum2,pixsum1,pixsum2;
        float rhnow,avecurv1,avecurv2;

        /* initialize the solid and air count histograms */
        for(i=0;i<=499;i++){
                nsolid[i]=0;
                nair[i]=0;
        }

        /* Obtain needed user input */
      natonce=200;
      rade=3;
        rflag=0;   /* always initialize system */

        nsph=maketemp(rade);
        printf("nsph is %d \n",nsph);
        if(rflag==0){
                sysinit(ph1id,ph2id);
        }
            else{
                sysscan(ph1id,ph2id);
        }
        i=0;
        rhnow=rhcalc(ph1id);
        while((rhnow<rhtarget)&&(i<MAXCYC)){
                printf("Now: %f  Target: %f \n",rhnow,rhtarget);
                i+=1;
                printf("Cycle: %d \n",i);
                keepgo=movepix(natonce,ph1id,ph2id);
                /* If equilibrium is reached, then return to calling routine */
                if(keepgo==1){
                        return;
                }
                curvsum1=0;
                curvsum2=0;
                pixsum1=0;
                pixsum2=0;
                /* Determine average curvatures for phases 1 and 2 */
                for(j=0;j<=nsph;j++){
                        pixsum1+=nsolid[j];
                        curvsum1+=(j*nsolid[j]);
                        pixsum2+=nair[j];
                        curvsum2+=(j*nair[j]);
                }
                avecurv1=(float)curvsum1/(float)pixsum1;
                avecurv2=(float)curvsum2/(float)pixsum2;
                printf("Ave. solid curvature: %f \n",avecurv1);
                printf("Ave. air curvature: %f \n",avecurv2);
                rhnow=rhcalc(ph1id);
        }
}

void stat3d(){
        int valin,ix,iy,iz;
```

```
        int ix1,iy1,iz1,k;
        long int voltot,surftot;

        for(ix=0;ix<=42;ix++){
                volume[ix]=surface[ix]=0;
        }

/* Read in image and accumulate volume totals */
        for(iz=1;iz<=SYSIZE;iz++){
        for(iy=1;iy<=SYSIZE;iy++){
        for(ix=1;ix<=SYSIZE;ix++){
                valin=mask[ix][iy][iz];
                volume[valin]+=1;
        }
        }
        }


        for(iz=1;iz<=SYSIZE;iz++){
        for(iy=1;iy<=SYSIZE;iy++){
        for(ix=1;ix<=SYSIZE;ix++){
                if(mask [ix] [iy] [iz]!=0){
                        valin=mask [ix] [iy] [iz];
                /* Check six neighboring pixels for porosity */
                        for(k=1;k<=6;k++){

                                switch (k){
                                       case 1:
                                                ix1=ix-1;
                                                if(ix1<1){ix1+=SYSIZE;}
                                                iy1=iy;
                                                iz1=iz;
                                                break;
                                       case 2:
                                                ix1=ix+1;
                                                if(ix1>SYSIZE){ix1-=SYSIZE;}
                                                iy1=iy;
                                                iz1=iz;
                                                break;
                                       case 3:
                                                iy1=iy-1;
                                                if(iy1<1){iy1+=SYSIZE;}
                                                ix1=ix;
                                                iz1=iz;
                                                break;
                                       case 4:
                                                iy1=iy+1;
                                                if(iy1>SYSIZE){iy1-=SYSIZE;}
                                                ix1=ix;
                                                iz1=iz;
                                                break;
                                       case 5:
                                                iz1=iz-1;
                                                if(iz1<1){iz1+=SYSIZE;}
                                                iy1=iy;
                                                ix1=ix;
                                                break;
```

```
                                        case 6:
                                                iz1=iz+1;
                                                if(iz1>SYSIZE){iz1-=SYSIZE;}
                                                iy1=iy;
                                                ix1=ix;
                                                break;
                                        default:
                                                break;
                                }
        if((ix1<1)||(iy1<1)||(iz1<1)||(ix1>SYSIZE)||(iy1>SYSIZE)||(iz1>SYSIZE)){
                                        printf("%d %d %d \n",ix1,iy1,iz1);
                                        exit(1);
                                }
                                if(mask[ix1] [iy1] [iz1]==0){
                                        surface[valin]+=1;
                                }
                        }
                }
        }
        }
        }

        printf("Phase    Volume      Surface     Volume    Surface \n");
        printf(" ID      count        count      fraction  fraction \n");
/* Only include clinker phases in surface area fraction calculation */
        surftot=surface[1]+surface[2]+surface[3]+surface[4];
        voltot=volume[1]+volume[2]+volume[3]+volume[4];
        k=0;
        printf("  %d    %8ld     %8ld  \n",k,volume[0],surface[0]);
        for(k=1;k<=4;k++){
        printf("  %d    %8ld     %8ld     %.5f   %.5f\n",k,volume[k],surface[k],

(float)volume[k]/(float)voltot,(float)surface[k]/(float)surftot);
        }
        printf("Total  %8ld     %8ld\n\n\n\n",voltot,surftot);
        for(k=5;k<=11;k++){
            printf("  %d    %8ld     %8ld\n",k,volume[k],surface[k]);
        }
      printf(" 20    %8ld     %8ld\n",volume[20],surface[20]);
        for(k=24;k<=27;k++){
            printf(" %d    %8ld     %8ld\n",k,volume[k],surface[k]);
        }
      printf(" 28    %8ld     %8ld\n",volume[28],surface[28]);
}

void rand3d(phasein,phaseout,filecorr,xpt)
      int phasein,phaseout;
      char filecorr[80];
      float xpt;
{
        int syssize,ires;
        float snew,s2,ss,sdiff,xtmp,ytmp;
        static float normm[SYSIZE+1][SYSIZE+1][SYSIZE+1];
        static float res[SYSIZE+1][SYSIZE+1][SYSIZE+1];
        static float filter [32][32][32];
        int done,r[61];
        static float s[61],xr[61],sum[502];
```

```
        float val2;
        float t1,t2,x1,x2,u1,u2,xrad,resmax,resmin;
        float xtot,filval,radius,sect,sumtot,vcrit;
        int valin,r1,r2,i1,i2,i3,i,j,k,j1,k1;
        int ido,iii,jjj,ix,iy,iz,index;
      FILE *corrfile;

/* Create the Gaussian noise image */
       i1=i2=i3=1;
       for(i=1;i<=((SYSIZE*SYSIZE*SYSIZE)/2);i++){
             u1=ran1(seed);
             u2=ran1(seed);
             t1=2.*PI*u2;
             t2=sqrt(-2.*log(u1));
             x1=cos(t1)*t2;
             x2=sin(t1)*t2;
             normm[i1][i2][i3]=x1;
             i1+=1;
             if(i1>SYSIZE){
                  i1=1;
                  i2+=1;
                  if(i2>SYSIZE){
                        i2=1;
                        i3+=1;
                  }
             }
             normm[i1][i2][i3]=x2;
             i1+=1;
             if(i1>SYSIZE){
                  i1=1;
                  i2+=1;
                  if(i2>SYSIZE){
                        i2=1;
                        i3+=1;
                  }
             }
       }

/* Now perform the convolution */
      corrfile=fopen(filecorr,"r");

      fscanf(corrfile,"%d",&ido);
      printf("Number of points in correlation file is %d \n",ido);
      for(i=1;i<=ido;i++){
            fscanf(corrfile,"%d %f",&valin,&val2);
            r[i]=valin;
            s[i]=val2;
            xr[i]=(float)r[i];
       }
       fclose(corrfile);
       ss=s[1];
       s2=ss*ss;
/* Load up the convolution matrix */
       sdiff=ss-s2;
       for(i=0;i<31;i++){
           iii=i*i;
           for(j=0;j<31;j++){
```

```
                jjj=j*j;
                for(k=0;k<31;k++){
                   xtmp=(float)(iii+jjj+k*k);
                   radius=sqrt(xtmp);
                   r1=(int)radius+1;
                   r2=r1+1;
                   if(s[r1]<0.0){
                       printf("%d and %d %f and %f with xtmp of
%f\n",r1,r2,s[r1],s[r2],xtmp);
                       fflush(stdout);
                       exit(1);
                   }
                   xrad=radius+1-r1;
                   filval=s[r1]+(s[r2]-s[r1])*xrad;
                   filter[i+1][j+1][k+1]=(filval-s2)/sdiff;
                }
            }
        }
/* Now filter the image maintaining periodic boundaries */
        resmax=0.0;
        resmin=1.0;
        for(i=1;i<=SYSIZE;i++){
        for(j=1;j<=SYSIZE;j++){
        for(k=1;k<=SYSIZE;k++){
             res[i][j][k]=0.0;
             if((float)mask[i][j][k]==phasein){
                   for(ix=1;ix<=31;ix++){
                       i1=i+ix-1;
                       if(i1<1){i1+=SYSIZE;}
                       else if(i1>SYSIZE){i1-=SYSIZE;}
                       for(iy=1;iy<=31;iy++){
                            j1=j+iy-1;
                            if(j1<1){j1+=SYSIZE;}
                            else if(j1>SYSIZE){j1-=SYSIZE;}
                            for(iz=1;iz<=31;iz++){
                                 k1=k+iz-1;
                                 if(k1<1){k1+=SYSIZE;}
                                 else if(k1>SYSIZE){k1-=SYSIZE;}
                            res[i][j][k]+=normm[i1][j1][k1]*filter[ix][iy][iz];
                            }
                       }
                   }
                   if(res[i][j][k]>resmax){resmax=res[i][j][k];}
                   if(res[i][j][k]<resmin){resmin=res[i][j][k];}
             }
        }
        }
        }

/* Now threshold the image */
        sect=(resmax-resmin)/500.;
        for(i=1;i<=500;i++){
            sum[i]=0.0;
        }
        xtot=0.0;
        for(i=1;i<=SYSIZE;i++){
        for(j=1;j<=SYSIZE;j++){
```

```
        for(k=1;k<=SYSIZE;k++){
            if((float)mask[i][j][k]==phasein){
                 xtot+=1.0;
                 index=1+(int)((res[i][j][k]-resmin)/sect);
                 if(index>500){index=500;}
                 sum[index]+=1.0;
            }
        }
        }
        }
/* Determine which bin to choose for correct thresholding */
        sumtot=vcrit=0.0;
        done=0;
        for(i=1;((i<=500)&&(done==0));i++){
            sumtot+=sum[i]/xtot;
            if(sumtot>xpt){
                 ytmp=(float)i;
                 vcrit=resmin+(resmax-resmin)*(ytmp-0.5)/500.;
                 done=1;
            }
        }
        printf("Critical volume fraction is %f\n",vcrit);
        ires=0;

        for(k=1;k<=SYSIZE;k++){
        for(j=1;j<=SYSIZE;j++){
        for(i=1;i<=SYSIZE;i++){
             if((float)mask[i][j][k]==phasein){
                 if(res[i][j][k]>vcrit){
                        mask[i][j][k]=phaseout;
                 }
             }
        }
        }
        }

}

main(){
       int i,j,k,iseed,alumflag,alumval,alum2,valin;
       float volin,volf[5],surff[5],rhtest,rdesire;
       char filen[80],fileout[80],filecem[80],filec3s[80],filesil[80],filealum[80];
       FILE *infile,*outfile,*testfile;

       /* Seed the random number generator */
         printf("Enter random number seed (negative integer) \n");
         scanf("%d",&iseed);
         printf("%d\n",iseed);
         seed=(&iseed);

       /* Read in the parameters to use */
         printf("Enter name of cement microstructure image file\n");
         scanf("%s",filen);
         printf("%s\n",filen);

       /* Set up the correlation filenames */
         printf("Enter root name of cement correlation files\n");
```

```
        scanf("%s",filecem);

        printf("%s\n",filecem);
      sprintf(filesil,"%s",filecem);
      strcat(filesil,".sil");
      sprintf(filec3s,"%s",filecem);
      strcat(filec3s,".c3s");
      sprintf(filealum,"%s",filecem);
      alumflag=1;
      alumval=4;
      strcat(filealum,".c4f");
      testfile=fopen(filealum,"r");
      if(testfile==NULL){
            alumflag=0;
            sprintf(filealum,"%s",filecem);
            strcat(filealum,".c3a");
            alumval=3;
      }
      else{
            fclose(testfile);
      }

       printf("Enter name of new cement microstructure image file\n");
       scanf("%s",fileout);
       printf("%s\n",fileout);
      for(i=1;i<=4;i++){
            scanf("%f",&volin);
            volf[i]=volin;
            printf("%f\n",volf[i]);
            scanf("%f",&volin);
            surff[i]=volin;
            printf("%f\n",surff[i]);
      }

/* Read in the original microstructure image file */
        infile=fopen(filen,"r");

        for(k=1;k<=SYSIZE;k++){
        for(j=1;j<=SYSIZE;j++){
        for(i=1;i<=SYSIZE;i++){
             fscanf(infile,"%d",&valin);
             mask[i][j][k]=valin;
           curvature[i][j][k]=0;
        }
        }
        }
        fclose(infile);

      /* First filtering */
      volin=volf[1]+volf[2];
        if(volin<1.0){
            rand3d(1,alumval,filesil,volin);

            /* First sintering */
            stat3d();
            rdesire=(surff[1]+surff[2])*(float)(surface[1]+surface[alumval]);
                if(rdesire!=0.0){
```

```
      if((int)rdesire<surface[1]){
            rhtest=(6./4.)*(float)(volume[1])/rdesire;
            sinter3d(1,alumval,rhtest);
      }
      else{
            rdesire=(surff[3]+surff[4])*(float)(surface[1]+surface[alumval]);
            if(rdesire!=0.0){
                  rhtest=(6./4.)*(float)(volume[alumval])/rdesire;
                  sinter3d(alumval,1,rhtest);
            }
      }
      }
}

/* Second filtering */
  if((volf[1]+volf[2])>0.0){
volin=volf[1]/(volf[1]+volf[2]);
  if(volin<1.0){
      rand3d(1,2,filec3s,volin);

      /* Second sintering */
      stat3d();
      rdesire=(surff[1]/(surff[1]+surff[2]))*(float)(surface[1]+surface[2]);
            if(rdesire!=0.0){
      if((int)rdesire<surface[1]){
            rhtest=(6./4.)*(float)(volume[1])/rdesire;
            sinter3d(1,2,rhtest);
      }
      else{

rdesire=(surff[2]/(surff[1]+surff[2]))*(float)(surface[1]+surface[2]);
            if(rdesire!=0.0){
                  rhtest=(6./4.)*(float)(volume[2])/rdesire;
                  sinter3d(2,1,rhtest);
            }
      }
      }
}
  }

/* Third (final) filtering */
if(alumval==4){
      volin=volf[4]/(volf[4]+volf[3]);
      alum2=3;
}
else{
      volin=volf[3]/(volf[4]+volf[3]);
      alum2=4;
}
  if(volin<1.0){
rand3d(alumval,alum2,filealum,volin);

/* Third (final) sintering */
stat3d();
if(alumval==4){
      rdesire=(surff[4]/(surff[3]+surff[4]))*(float)(surface[3]+surface[4]);
  if(rdesire!=0.0){
```

```
      if((int)rdesire<surface[4]){
            rhtest=(6./4.)*(float)(volume[4])/rdesire;
            sinter3d(alumval,alum2,rhtest);
      }
      else{
            rdesire=(surff[3]/(surff[3]+surff[4]))*(float)(surface[3]+surface[4]);
            if(rdesire!=0.0){
                  rhtest=(6./4.)*(float)(volume[3])/rdesire;
                  sinter3d(alum2,alumval,rhtest);
            }
      }
      }
      }
      else{
            rdesire=(surff[3]/(surff[3]+surff[4]))*(float)(surface[3]+surface[4]);
        if(rdesire!=0.0){
            if((int)rdesire<surface[3]){
                  rhtest=(6./4.)*(float)(volume[3])/rdesire;
                  sinter3d(alumval,alum2,rhtest);
            }
            else{

      rdesire=(surff[4]/(surff[3]+surff[4]))*(float)(surface[3]+surface[4]);
                        if(rdesire!=0.0){
                        rhtest=(6./4.)*(float)(volume[4])/rdesire;
                        sinter3d(alum2,alumval,rhtest);
                  }
            }
      }
      }
      }

      /* Output final microstructure */
       outfile=fopen(fileout,"w");

       for(k=1;k<=SYSIZE;k++){
       for(j=1;j<=SYSIZE;j++){
       for(i=1;i<=SYSIZE;i++){
            fprintf(outfile,"%2d\n",mask[i][j][k]);
       }
       }
       }

}
```

Program stat3d.c

```
/*********************************************************************/
/*                                                                   */
/*      Program: stat3d.c                                            */
/*      Purpose: To read in a 3-D image and output phase volumes     */
/*               and report the volume and pore-exposed surface area */
/*               fractions                                           */
/*      Programmer: Dale P. Bentz                                    */
/*                  NIST                                             */
/*                  100 Bureau Drive Mail Stop 8615                  */
/*                  Gaithersburg, MD  20899-8615                     */
/*                  Phone: (301) 975-5865                            */
/*                  E-mail: dale.bentz@.nist.gov                     */
/*                                                                   */
/*********************************************************************/
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

#include <stdio.h>
#include <math.h>

#define ISIZE 100
#define NPHASES 50

main(){
        static int mic [ISIZE] [ISIZE] [ISIZE];
        int valin,ix,iy,iz;
        int ix1,iy1,iz1,k;
        long int voltot,surftot,volume[NPHASES],surface [NPHASES];
        FILE *infile,*statfile;
        char filen[80],fileout[80];

        printf("Enter name of file to open \n");
        scanf("%s",filen);
        printf("%s \n",filen);
        printf("Enter name of file to write statistics to \n");
        scanf("%s",fileout);
        printf("%s \n",fileout);

        for(ix=0;ix<=(NPHASES-1);ix++){
                volume[ix]=surface[ix]=0;
        }
```

```
        infile=fopen(filen,"r");
        statfile=fopen(fileout,"w");

/* Read in image and accumulate volume totals */
        for(iz=0;iz<ISIZE;iz++){
        for(iy=0;iy<ISIZE;iy++){
        for(ix=0;ix<ISIZE;ix++){
                fscanf(infile,"%d",&valin);
                mic [ix] [iy] [iz]=valin;
            if(valin<NPHASES){
                        volume[valin]+=1;
            }
            else{
                  volume[0]+=1;
            }
        }
        }
        }
        fclose(infile);

        for(iz=0;iz<ISIZE;iz++){
        for(iy=0;iy<ISIZE;iy++){
        for(ix=0;ix<ISIZE;ix++){
                if((mic [ix] [iy] [iz]!=0)&&(mic[ix][iy][iz]<=49)){
                        valin=mic [ix] [iy] [iz];
                /* Check six neighboring pixels for porosity */
                        for(k=1;k<=6;k++){

                                switch (k){
                                       case 1:
                                               ix1=ix-1;
                                               if(ix1<0){ix1+=ISIZE;}
                                               iy1=iy;
                                               iz1=iz;
                                               break;
                                       case 2:
                                               ix1=ix+1;
                                               if(ix1>=ISIZE){ix1-=ISIZE;}
                                               iy1=iy;
                                               iz1=iz;
                                               break;
                                       case 3:
                                               iy1=iy-1;
                                               if(iy1<0){iy1+=ISIZE;}
                                               ix1=ix;
                                               iz1=iz;
                                               break;
                                       case 4:
                                               iy1=iy+1;
                                               if(iy1>=ISIZE){iy1-=ISIZE;}
                                               ix1=ix;
                                               iz1=iz;
                                               break;
                                       case 5:
                                               iz1=iz-1;
                                               if(iz1<0){iz1+=ISIZE;}
                                               iy1=iy;
```

```
                                        ix1=ix;
                                        break;
                                case 6:
                                        iz1=iz+1;
                                        if(iz1>=ISIZE){iz1-=ISIZE;}
                                        iy1=iy;
                                        ix1=ix;
                                        break;
                                default:
                                        break;
                                }
        if((ix1<0)||(iy1<0)||(iz1<0)||(ix1>=ISIZE)||(iy1>=ISIZE)||(iz1>=ISIZE)){
                                        printf("%d %d %d \n",ix1,iy1,iz1);
                                        exit(1);
                                }
                                if((mic[ix1] [iy1]
[iz1]==0)||(mic[ix1][iy1][iz1]>44)){
                                        surface[valin]+=1;
                                }
                        }
                }
        }
        }
        }

        printf("Phase    Volume      Surface     Volume    Surface \n");
        printf(" ID      count        count      fraction  fraction \n");
        fprintf(statfile,"Phase    Volume      Surface     Volume    Surface \n");
        fprintf(statfile," ID      count        count      fraction  fraction \n");
/* Only include clinker phases in surface area fraction calculation */
        surftot=surface[1]+surface[2]+surface[3]+surface[4];
        voltot=volume[1]+volume[2]+volume[3]+volume[4];
        k=0;
        printf("  %d    %8ld     %8ld  \n",k,volume[0],surface[0]);
        fprintf(statfile,"  %d    %8ld     %8ld  \n",k,volume[0],surface[0]);
        for(k=1;k<=4;k++){
        printf("  %d    %8ld     %8ld     %.5f   %.5f\n",k,volume[k],surface[k],

(float)volume[k]/(float)voltot,(float)surface[k]/(float)surftot);
        fprintf(statfile,"  %d    %8ld     %8ld     %.5f
%.5f\n",k,volume[k],surface[k],

(float)volume[k]/(float)voltot,(float)surface[k]/(float)surftot);

        }
        printf("Total  %8ld     %8ld\n\n\n\n",voltot,surftot);
        fprintf(statfile,"Total  %8ld     %8ld\n\n\n\n",voltot,surftot);
        for(k=5;k<=30;k++){
            printf("  %d    %8ld     %8ld\n",k,volume[k],surface[k]);
            fprintf(statfile,"  %d    %8ld     %8ld\n",k,volume[k],surface[k]);
        }
      printf("  35    %8ld     %8ld\n",volume[35],surface[35]);
      fprintf(statfile,"  35    %8ld     %8ld\n",volume[35],surface[35]);
      printf("  45    %8ld     %8ld\n",volume[45],surface[45]);
      fprintf(statfile,"  45    %8ld     %8ld\n",volume[45],surface[45]);
        fclose(statfile);
}
```

B Computer Program Listings for Cement Hydration Model

Program disrealnew.c

```
/*************************************************************************/
/*                                                                       */
/*      Program disrealnew.c to hydrate three-dimensional cement         */
/*              particles in a 3-D box with periodic boundaries.         */
/*              Uses cellular automata techniques and preserves correct  */
/*              hydration volume stoichiometry.                          */
/*      Programmer: Dale P. Bentz                                        */
/*                  Building and Fire Research Laboratory                */
/*                  NIST                                                 */
/*                  100 Bureau Drive Mail Stop 8615                      */
/*                  Gaithersburg, MD  20899   USA                        */
/*                  (301) 975-5865      FAX: 301-990-6891                */
/*                  E-mail: dale.bentz@nist.gov                          */
/*                                                                       */
/*************************************************************************/
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

/* Heat of formation data added: 12/92 */
/* Three-dimensional version created 7/94 */
/* General C version created 8/94 */
/* Modified to have pseudo-continuous dissolution 11/94 */
/* In this program, dissolved silicates are located close */
/* to dissolution source within a 17*17*17 box centered at dissolution source */
/* Hydration under self-desiccating conditions included 9/95 */
/* Additions for adiabatic hydration conditions included 9/96 */
/* Adjustments for pozzolanic reaction included 11/96 */
/* Additions for calcium chloride to Freidel's salt added 4/97 */
/* Additions for stratlingite formation added 4/97 */
/* Additions for anhydrite to gypsum conversion added 5/97 */
/* Additions for calcium aluminodisilicate added 7/97 */
/* Temperature-variable C-S-H properties added 8/98 */
/* molar volumes and water consumption based on values */
/* given in Geiker thesis on chemical shrinkage */
/* Modelling of induction period based on an impermeable layer */
/* theory */
/* Ettringite unstable above 70 C added 9/98 */
```

```
/* Hemihydrate to gypsum conversion added 9/98 */
/* Decided to base alpha only on four major clinker phases 9/98 */
/* Updated dissolution to allow next nearest neighbors, etc. 9/98 */
/* Created stable iron-rich ettringite 3/99 */
/* Phases renumbered 1/00 */
/* Sulfate reactions modified by Claus-Jochen Haecker 6/00 */
/* Reactions between CaCO3 and Afm phase added 8/00 */
/* Slag incorporation 2/01 */
/* pH and pore solution concentration added 8/01 */
/* Influence of pH on hydration added 2/02 */
/* Induction and gypsum acceleration - influence of w/c modified 11/04 */
/* Possibility of resaturation added 12/04 */
/* C-S-H precipitating as plates added 01/05 */
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>

#define MAXCYC 30000    /* Maximum number of cycles of hydration */
                        /* For hydration under sealed conditions: */
#define CUBEMAX 7       /* Maximum cube size for checking pore size */
#define CUBEMIN 3       /* Minimum cube size for checking pore size */
#define SYSIZE 100      /* System size in pixels per dimension */
#define SYSIZEM1 99     /* System size -1 */
#define DISBIAS 30.0    /* Dissolution bias- to change all dissolution rates */
#define DISMIN 0.001    /* Minimum dissolution for C3S dissolution */
#define DISMIN2 0.00025  /* Minimum dissolution for C2S dissolution */
#define DISMINSLAG 0.0001  /* Minimum dissolution for SLAG dissolution */
#define DISMINASG 0.0005  /* Minimum dissolution for ASG dissolution */
#define DISMINCAS2 0.0005  /* Minimum dissolution for CAS2 dissolution */
#define DISMIN_C3A_0 0.002  /* Minimum dissolution for C3A dissolution */
#define DISMIN_C4AF_0 0.0005  /* Minimum dissolution for C4AF dissolution */
#define DETTRMAX 1200   /* Maximum allowed # of ettringite diffusing species */
#define DGYPMAX 2000    /*  Maximum allowed # of gypsum diffusing species */
#define DCACO3MAX 1000  /*  Maximum allowed # of CaCO3 diffusing species */
#define DCACL2MAX 2000  /* Maximum allowed # of CaCl2 diffusing species */
#define DCAS2MAX 2000   /* Maximum allowed # of CAS2 diffusing species */
#define CHCRIT 50.0     /* Scale parameter to adjust CH dissolution probability */
#define C3AH6CRIT 10.0  /* Scale par. to adjust C3AH6 dissolution prob. */
#define C3AH6GROW 0.01  /* Probability for C3AH6 growth */
#define CHGROW 1.0      /* Probability for CH growth */
#define CHGROWAGG 1.0   /* Probability for CH growth on aggregate surface */
#define ETTRGROW 0.002  /* Probability for ettringite growth */
#define C3AETTR 0.001   /* Probability for reaction of diffusing C3A
                             with ettringite  */
#define C3AGYP 0.001    /*  Probability for diffusing C3A to react with diffusing
gypsum */
                        /* diffusing anhydrite, and diffusing hemihydrate */
#define SOLIDC3AGYP 0.5 /* Prob. of solid C3A to react with diffusing sulfate */
#define SOLIDC4AFGYP 0.1 /* Prob. of solid C4AF to react with diffusing sulfate */
#define PPOZZ 0.05      /* base probability for pozzolanic reaction */
#define PCSH2CSH 0.002  /* probability for CSH dissolution */
                        /* for conversion of C-S-H to pozz. C-S-H */
#define A0_CHSOL 1.325  /* Parameters for variation of CH solubility with */
#define A1_CHSOL 0.008162   /* temperature (data from Taylor- Cement Chemistry) */
/* changed CSHSCALE to 70000 6/15/01  to better model induction CS */
#define CSHSCALE 70000. /*scale factor for CSH controlling induction */
```

```
#define WCSCALE 0.4     /* scale factor for influence of w/c on induction */
#define WCSULFSCALE 0.5 /* scale factor for influence of w/c on sulfate
acceleration of silicates and aluminates */
#define C3AH6_SCALE 2000. /*scale factor for C3AH6 controlling induction of
aluminates */
#define NEIGHBORS 26    /* number of neighbors to consider (6, 18, or 26) */
/* define IDs for each phase used in model */
/* To add a solid phase, insert new phase before ABSGYP */
/*  and increment all subsequent IDs */
/* To add a diffusing species, insert just before DIFFCACL2 */
/*  and increment all subsequent IDs */
#define POROSITY 0
#define C3S 1
#define C2S 2
#define C3A 3
#define C4AF 4
#define GYPSUM 5
#define HEMIHYD 6
#define ANHYDRITE 7
#define POZZ 8
#define INERT 9
#define SLAG 10
#define ASG 11          /* aluminosilicate glass */
#define CAS2 12
#define CH 13
#define CSH 14
#define C3AH6 15
#define ETTR 16
#define ETTRC4AF 17     /* Iron-rich stable ettringite phase */
#define AFM 18
#define FH3 19
#define POZZCSH 20
#define SLAGCSH 21      /* Slag gel-hydration product */
#define CACL2 22
#define FREIDEL 23      /* Freidel's salt */
#define STRAT 24        /* stratlingite (C2ASH8) */
#define GYPSUMS 25      /* Gypsum formed from hemihydrate and anhydrite */
#define CACO3 26
#define AFMC 27
#define INERTAGG 28
#define ABSGYP 29
#define DIFFCSH 30
#define DIFFCH 31
#define DIFFGYP 32
#define DIFFC3A 33
#define DIFFC4A 34
#define DIFFFH3 35
#define DIFFETTR 36
#define DIFFCACO3 37
#define DIFFAS 38
#define DIFFANH 39
#define DIFFHEM 40
#define DIFFCAS2 41
#define DIFFCACL2 42
#define EMPTYP 45         /* Empty porosity due to self desiccation */
#define OFFSET 50         /* Offset for highlighted potentially soluble pixel */
/* define heat capacities for all components in J/g/C */
```

```
/* including free and bound water */
#define Cp_cement 0.75
#define Cp_pozz 0.75
#define Cp_agg 0.84
#define Cp_CH 0.75
#define Cp_h2o 4.18     /* Cp for free water */
#define Cp_bh2o 2.2     /* Cp for bound water */
#define WN 0.23        /* water bound per gram of cement during hydration */
#define WCHSH 0.06     /* water imbibed per gram of cement during chemical
                  shrinkage (estimate) */

/* data structure for diffusing species - to be dynamically allocated */
/* Use of a doubly linked list to allow for easy maintenance */
/* (i.e. insertion and deletion) */
/* Added 11/94 */
/* Note that if SYSIZE exceeds 256, need to change x, y, and z to */
/* int variables */
struct ants{
        unsigned char x,y,z,id;
        int cycbirth;
        struct ants *nextant;
        struct ants *prevant;
};

/* data structure for elements to remove to simulate self-desiccation */
/* once again a doubly linked list */
struct togo{
        int x,y,z,npore;
        struct togo *nexttogo;
        struct togo *prevtogo;
};

/* Global variables */
/* Microstructure stored in array mic of type char to minimize storage */
/* Initial particle IDs stored in array micpart (for assessing set point) */
static char mic [SYSIZE] [SYSIZE] [SYSIZE];
static char micorig [SYSIZE] [SYSIZE] [SYSIZE];
static short int micpart [SYSIZE] [SYSIZE] [SYSIZE];
static short int cshage [SYSIZE] [SYSIZE] [SYSIZE];
static short int faces [SYSIZE] [SYSIZE] [SYSIZE];
/* counts for dissolved and solid species */
long int discount[EMPTYP+1], count[EMPTYP+1];
long int ncshplategrow=0,ncshplateinit=0;
/* Counts for pozzolan reacted, initial pozzolan, gypsum, ettringite,
initial porosity, and aluminosilicate reacted */
long int npr,nfill,ncsbar,netbar,porinit,nasr,nslagr,slagemptyp=0;
/* Initial clinker phase counts */
long int c3sinit,c2sinit,c3ainit,c4afinit,anhinit,heminit,chold,chnew;
long int nmade,ngoing,gypready,poregone,poretodo,countpore=0;
long int countkeep,water_left,water_off,pore_off;
int ncyc,cyccnt,cubesize,sealed,outfreq;
int icyc,burnfreq,setfreq,setflag,sf1,sf2,sf3,porefl1,porefl2,porefl3;
int xoff[27]={1,0,0,-1,0,0,1,1,-1,-1,0,0,0,0,1,1,-1,-1,1,1,1,1,-1,-1,-1,-1,0};
int yoff[27]={0,1,0,0,-1,0,1,-1,1,-1,1,-1,1,-1,0,0,0,0,1,-1,1,-1,1,1,-1,-1,0};
int zoff[27]={0,0,1,0,0,-1,0,0,0,0,1,1,-1,-1,1,-1,1,-1,1,1,-1,-1,1,-1,1,-1,0};
/* Parameters for kinetic modelling ---- maturity approach */
float ind_time,temp_0,temp_cur,time_step=0.0,time_cur,E_act,beta,heat_cf;
```

```
float w_to_c=0.0,s_to_c,krate,totfract=1.0,tfractw04=0.438596,fractwithfill=1.0;
float tfractw05=0.384615,surffract=0.0,pfract,pfractw05=0.615385,sulf_conc;
long int scntcement=0,scnttotal=0;
float U_coeff=0.0,T_ambient=25.;
float alpha_cur,heat_old,heat_new,cemmass,mass_agg,mass_water,mass_fill,Cp_now;
float alpha,CH_mass,mass_CH,mass_fill_pozz,E_act_pozz,chs_new,cemmasswgyp;
float flyashmass,alpha_fa_cur;
float E_act_slag;
/* Arrays for variable CSH molar volume and water consumption */
float molarvcsh[MAXCYC],watercsh[MAXCYC],heatsum,molesh2o,saturation=1.0;
/* Arrays for dissolution probabilities for each phase */
float disprob[EMPTYP+1],disbase[EMPTYP+1],gypabsprob,ppozz;
/* Arrays for specific gravities, molar volumes, heats of formation, and */
/* molar water consumption for each phase */
float specgrav[EMPTYP+1],molarv[EMPTYP+1],heatf[EMPTYP+1],waterc[EMPTYP+1];
/* Solubility flags and diffusing species created for each phase */
/* Also flag for C1.7SH4.0 to C1.1SH3.9 conversion */
int soluble[EMPTYP+1],creates[EMPTYP+1],csh2flag,adiaflag,chflag,nummovsl;
float cs_acc=1.0;     /* increases disprob[C3S] and disprob[C2S] if gypsum is
present */
float ca_acc=1.0;    /* increases disprob[C3A] and disprob[C4AF] if gypsum is
present */
float dismin_c3a=DISMIN_C3A_0;
float dismin_c4af=DISMIN_C4AF_0;
float gsratio2=0.0,onepixelbias=1.0;
/* Slag probabilities */
float p1slag;   /*  probability SLAG is converted to SLAGCSH */
float p2slag;   /*  probability SLAG is converted to POROSITY or EMPTYP */
float p3slag;   /*  probability adjoining pixel is converted to SLAGCSH */
float p4slag;   /*  probability CH is consumed during SLAG reaction */
float p5slag;   /*  probability a C3A diffusing species is created */
float slagcasi,slaghydcasi;   /* Ca/Si ratios for SLAG and SLAGCSH */
float slagh2osi;     /* H/S ratio of SLAGCSH */
float slagc3a;       /* C3A/slag molar ratio */
float siperslag;     /* S ratio of SLAG (per mole) */
float slagreact;     /* Base dissolution reactivity factor for SLAG */
long int DIFFCHdeficit=0,slaginit=0;   /* Deficit in CH due to SLAG reaction */
long int slagcum=0,chgone=0;
long int nch_slag=0;    /* number of CH consumed by SLAG reaction */
long int sulf_cur=0;
long int sulf_solid;
int *seed;       /* Random number seed */
char heatname[80],adianame[80],phname[80],ppsname[80],ptsname[80],phrname[80];
char chshrname[80],moviename[80],parname[80],micname[80];
char cmdnew[120],pHname[80],fileroot[80];
struct ants *headant,*tailant;
FILE *heatfile,*chsfile,*ptmpfile,*movfile,*pHfile,*micfile;
/* Variables for alkali predictions */
float pH_cur,totsodium,totpotassium,rssodium,rspotassium;
/* Array for whether pH influences phase solubility  -- added 2/12/02 */
float pHeffect[EMPTYP+1],pHfactor=0.0;
int pHactive,resatcyc,cshgeom;
/* Make conccaplus global to speed up execution and moles_syn_precip */
/* global to accumulate properly */
double conccaplus=0.0,moles_syn_precip=0.0,concsulfate=0.0;
int primevalues[6]={2,3,5,7,11,13};
int cshboxsize;              /* Box size for location of extra diffusing C-S-H */
```

```
/* Supplementary programs */
#include "ran1.c"             /* random number generation */
#include "burn3d.c"           /* percolation of porosity assessment */
#include "burnset.c"          /* set point assessment */
#include "parthyd.c"          /* particle hydration assessment */
#include "hydrealnew.c"       /* hydration execution */
#include "pHpred.c"           /* pore solution pH prediction */

/* routine to initialize values for solubilities, molar volumes, etc. */
/* Called by main program */
/* Calls no other routines */
void init()
{
        int i;
        float slagin,CHperslag;
        FILE *slagfile,*alkalifile;

        for(i=0;i<=EMPTYP;i++){

                creates[i]=0;
                soluble[i]=0;
                disprob[i]=0.0;
                disbase[i]=0.0;
                pHeffect[i]=0.0;
        }

        /* soluble [x] - flag indicating if phase x is soluble */
        /* disprob [x] - probability of dissolution (relative diss. rate) */
        gypabsprob=0.01;      /* One sulfate absorbed per 100 CSH units */
        /* Source is from Taylor's Cement Chemistry  for K and Na absorption */
        /* Note that for the first cycle, of the clinker phases only the */
        /* aluminates and gypsum are soluble */
        soluble[C4AF]=1;
        disprob[C4AF]=disbase[C4AF]=0.067/DISBIAS;
        creates[C4AF]=POROSITY;
        pHeffect[C4AF]=1.0;
        soluble[C3S]=0;
        disprob[C3S]=disbase[C3S]=0.7/DISBIAS;
        creates[C3S]=DIFFCSH;
        pHeffect[C3S]=1.0;
        soluble[C2S]=0;
        disprob[C2S]=disbase[C2S]=0.1/DISBIAS;
        creates[C2S]=DIFFCSH;
        pHeffect[C2S]=1.0;
        soluble[C3A]=1;
        /* increased back to 0.4 from 0.25 7/8/99 */
        disprob[C3A]=disbase[C3A]=0.4/DISBIAS;
        creates[C3A]=POROSITY;
        pHeffect[C3A]=1.0;
        soluble[GYPSUM]=1;
        /* Changed from 0.05 to 0.015  9/29/98 */
        /* Changed to 0.040 10/15/98 */
        /* back to 0.05 from 0.10 7/8/99 */
        /* from 0.05 to 0.02 4/4/00 */
        /* from 0.02 to 0.025 8/13/01 */
        disprob[GYPSUM]=disbase[GYPSUM]=0.025;
```

```
        /* dissolved gypsum distributed at random throughout microstructure */
        creates[GYPSUM]=POROSITY;
        soluble[GYPSUMS]=1;
        pHeffect[GYPSUM]=0.0;
        /* Changed from 0.05 to 0.015  9/29/98 */
        /* Changed to 0.020 10/15/98 */
        /* and also changed all sulfate based dissolution rates */
        disprob[GYPSUMS]=disbase[GYPSUMS]=disprob[GYPSUM];
        creates[GYPSUMS]=POROSITY;
        pHeffect[GYPSUMS]=0.0;
        soluble[ANHYDRITE]=1;
        /* set anhydrite dissolution at 4/5ths of that of gypsum */
        /* Source: Uchikawa et al., Cement and Concrete Research, 1984 */
        disprob[ANHYDRITE]=disbase[ANHYDRITE]=0.8*disprob[GYPSUM];
        /* dissolved anhydrite distributed at random throughout microstructure */
        creates[ANHYDRITE]=POROSITY;
        pHeffect[ANHYDRITE]=0.0;
        soluble[HEMIHYD]=1;
        /* set hemihydrate dissolution at 3 times that of gypsum */
        /* Source: Uchikawa et al., Cement and Concrete Research, 1984 */
        /* Changed to 1.5 times that of gypsum 6/1/00 */
        disprob[HEMIHYD]=disbase[HEMIHYD]=1.5*disprob[GYPSUM];
        /* dissolved hemihydrate distributed at random throughout microstructure */
        creates[HEMIHYD]=POROSITY;
        pHeffect[HEMIHYD]=0.0;
        /* CH soluble to allow for Ostwald ripening of crystals */
        soluble[CH]=1;
        disprob[CH]=disbase[CH]=0.5/DISBIAS;
        creates[CH]=DIFFCH;
        pHeffect[CH]=0.0;
        /* CaCO3 is only mildly soluble */
        soluble[CACO3]=1;
        disprob[CACO3]=disbase[CACO3]=0.10/DISBIAS;
        creates[CACO3]=DIFFCACO3;
        pHeffect[CACO3]=0.0;
        /* Slag is not truly soluble, but use its dissolution probability for
reaction probability */
        soluble[SLAG]=0;
        disprob[SLAG]=disbase[SLAG]=0.005/DISBIAS;
        creates[SLAG]=0;
        pHeffect[SLAG]=1.0;
        soluble[C3AH6]=1;
        disprob[C3AH6]=disbase[C3AH6]=0.01/DISBIAS; /* changed from 0.5 to 0.01
06.09.00 */
        creates[C3AH6]=POROSITY;
        pHeffect[C3AH6]=0.0;
        /* Ettringite is initially insoluble */
        soluble[ETTR]=0;
        /* Changed to 0.008 from 0.020  3/11/99 */
        disprob[ETTR]=disbase[ETTR]=0.008/DISBIAS;
        creates[ETTR]=DIFFETTR;
        pHeffect[ETTR]=0.0;
        /* Iron-rich ettringite is always insoluble */
        soluble[ETTRC4AF]=0;
        disprob[ETTRC4AF]=disbase[ETTRC4AF]=0.0;
        creates[ETTRC4AF]=ETTRC4AF;
        pHeffect[ETTRC4AF]=0.0;
```

```
/* calcium chloride is soluble */
soluble[CACL2]=1;
disprob[CACL2]=disbase[CACL2]=0.1/DISBIAS;
creates[CACL2]=DIFFCACL2;
pHeffect[CACL2]=0.0;
/* aluminosilicate glass is soluble */
soluble[ASG]=1;
disprob[ASG]=disbase[ASG]=0.2/DISBIAS;
creates[ASG]=DIFFAS;
pHeffect[ASG]=1.0;
/* calcium aluminodisilicate is soluble */
soluble[CAS2]=1;
disprob[CAS2]=disbase[CAS2]=0.2/DISBIAS;
creates[CAS2]=DIFFCAS2;
pHeffect[CAS2]=1.0;

/* establish molar volumes and heats of formation */
/* molar volumes are in cm^3/mole */
/* heats of formation are in kJ/mole */
/* See paper by Fukuhara et al., Cem. & Conc. Res., 11, 407-14, 1981. */
/* values for Porosity are those of water */
molarv[POROSITY]=18.068;
heatf[POROSITY]=(-285.83);
waterc[POROSITY]=1.0;
specgrav[POROSITY]=0.99707;

molarv[C3S]=71.129;
heatf[C3S]=(-2927.82);
waterc[C3S]=0.0;
specgrav[C3S]=3.21;
/* For improvement in chemical shrinkage correspondence */
/* Changed molar volume of C(1.7)-S-H(4.0) to 108 5/24/95 */
molarv[CSH]=108.;
heatf[CSH]=(-3283.0);
waterc[CSH]=4.0;
specgrav[CSH]=2.11;

molarv[CH]=33.1;
heatf[CH]=(-986.1);
waterc[CH]=1.0;
specgrav[CH]=2.24;

/* Assume that calcium carbonate is in the calcite form */
molarv[CACO3]=36.93;
waterc[CACO3]=0.0;
specgrav[CACO3]=2.71;
heatf[CACO3]=(-1206.92);

molarv[AFMC]=261.91;
waterc[AFMC]=11.0;
specgrav[AFMC]=2.17;
/* Need to fill in heat of formation at a later date */
heatf[AFMC]=(0.0);

molarv[C2S]=52.513;
heatf[C2S]=(-2311.6);
waterc[C2S]=0.0;
```

```
        specgrav[C2S]=3.28;

        molarv[C3A]=88.94;
        heatf[C3A]=(-3587.8);
        waterc[C3A]=0.0;
        specgrav[C3A]=3.038;

        molarv[GYPSUM]=74.21;
        heatf[GYPSUM]=(-2022.6);
        waterc[GYPSUM]=0.0;
        specgrav[GYPSUM]=2.32;
        molarv[GYPSUMS]=74.21;
        heatf[GYPSUMS]=(-2022.6);
        waterc[GYPSUMS]=0.0;
        specgrav[GYPSUMS]=2.32;

        molarv[ANHYDRITE]=52.16;
        heatf[ANHYDRITE]=(-1424.6);
        waterc[ANHYDRITE]=0.0;
        specgrav[ANHYDRITE]=2.61;

        molarv[HEMIHYD]=52.973;
        heatf[HEMIHYD]=(-1574.65);
        waterc[HEMIHYD]=0.0;
        specgrav[HEMIHYD]=2.74;

        molarv[C4AF]=130.29;
        heatf[C4AF]=(-5090.3);
        waterc[C4AF]=0.0;
        specgrav[C4AF]=3.73;

        molarv[C3AH6]=150.12;
        heatf[C3AH6]=(-5548.);
        waterc[C3AH6]=6.0;
        specgrav[C3AH6]=2.52;

/* Changed molar volume of FH3 to 69.8 (specific gravity of 3.06) 5/23/95 */
        molarv[FH3]=69.803;
        heatf[FH3]=(-823.9);
        waterc[FH3]=3.0;
        specgrav[FH3]=3.062;

        molarv[ETTRC4AF]=735.01;
        heatf[ETTRC4AF]=(-17539.0);
        waterc[ETTRC4AF]=26.0;
        specgrav[ETTRC4AF]=1.7076;
/* Changed molar volue of ettringite to 735 (spec. gr.=1.7076)  5/24/95 */
        molarv[ETTR]=735.01;
        heatf[ETTR]=(-17539.0);
        waterc[ETTR]=26.0;
        specgrav[ETTR]=1.7076;

        molarv[AFM]=312.82;
        heatf[AFM]=(-8778.0);
        /* Each mole of AFM which forms requires 12 moles of water, */
        /* two of which are supplied by gypsum in forming ETTR  */
        /* leaving 10 moles to be incorporated from free water */
```

```
        waterc[AFM]=10.0;
        specgrav[AFM]=1.99;

        molarv[CACL2]=51.62;
        heatf[CACL2]=(-795.8);
        waterc[CACL2]=0;
        specgrav[CACL2]=2.15;

        molarv[FREIDEL]=296.662;
        /* No data available for heat of formation */
        heatf[FREIDEL]=(0.0);
        /* 10 moles of H2O per mole of Freidel's salt */
        waterc[FREIDEL]=10.0;
        specgrav[FREIDEL]=1.892;

        /* Basic reaction is 2CH + ASG + 6H --> C2ASH8 (Stratlingite) */
        molarv[ASG]=49.9;
        /* No data available for heat of formation */
        heatf[ASG]=0.0;
        waterc[ASG]=0.0;
        specgrav[ASG]=3.247;

        molarv[CAS2]=100.62;
        /* No data available for heat of formation */
        heatf[CAS2]=0.0;
        waterc[CAS2]=0.0;
        specgrav[CAS2]=2.77;

        molarv[STRAT]=215.63;
        /* No data available for heat of formation */
        heatf[STRAT]=0.0;
        /* 8 moles of water per mole of stratlingite */
        waterc[STRAT]=8.0;
        specgrav[STRAT]=1.94;

        molarv[POZZ]=27.0;
        /* Use heat of formation of SiO2 (quartz) for unreacted pozzolan */
        /* Source- Handbook of Chemistry and Physics */
        heatf[POZZ]=-907.5;
        waterc[POZZ]=0.0;
        specgrav[POZZ]=2.22;

        /* Data for Pozzolanic CSH based on work of Atlassi, DeLarrard, */
        /* and Jensen */
        /* gives a chemical shrinkage of 0.2 g H2O/g CSF */
        /* heat of formation estimated based on heat release of */
        /* 780 J/g Condensed Silica Fume */
        /* MW is 191.8 g/mole */
        /* Changed molar volume to 101.81  3/10/97 */
        molarv[POZZCSH]=101.81;
        waterc[POZZCSH]=3.9;
        specgrav[POZZCSH]=1.884;
        heatf[POZZCSH]=(-2299.1);

/* Assume inert filler has same specific gravity and molar volume as SiO2 */
        molarv[INERT]=27.0;
        heatf[INERT]=0.0;
```

```
        waterc[INERT]=0.0;
        specgrav[INERT]=2.2;

        molarv[ABSGYP]=74.21;
        heatf[ABSGYP]=(-2022.6);
        waterc[ABSGYP]=0.0;
        specgrav[ABSGYP]=2.32;

        molarv[EMPTYP]=18.068;
        heatf[EMPTYP]=(-285.83);
        waterc[EMPTYP]=0.0;
        specgrav[EMPTYP]=0.99707;

        /* Read in values for alkali characteristics and */
        /* convert them to fractions from percentages */
        alkalifile=fopen("alkalichar.dat","r");
        fscanf(alkalifile,"%f",&totsodium);
        fscanf(alkalifile,"%f",&totpotassium);
        fscanf(alkalifile,"%f",&rssodium);
        fscanf(alkalifile,"%f",&rspotassium);
        fclose(alkalifile);
        totsodium/=100.;
        totpotassium/=100.;
        rssodium/=100.;
        rspotassium/=100.;
        /* Read in values for slag characteristics */
        slagfile=fopen("slagchar.dat","r");
        fscanf(slagfile,"%f",&slagin);
        fscanf(slagfile,"%f",&slagin);
        fscanf(slagfile,"%f",&slagin);
        specgrav[SLAG]=slagin;
        fscanf(slagfile,"%f",&slagin);
        specgrav[SLAGCSH]=slagin;
        fscanf(slagfile,"%f",&slagin);
        molarv[SLAG]=slagin;
        fscanf(slagfile,"%f",&slagin);
        molarv[SLAGCSH]=slagin;
        fscanf(slagfile,"%f",&slagcasi);
        fscanf(slagfile,"%f",&slaghydcasi);
        fscanf(slagfile,"%f",&siperslag);
        fscanf(slagfile,"%f",&slagin);
        waterc[SLAGCSH]=slagin*siperslag;
        fscanf(slagfile,"%f",&slagc3a);
        fscanf(slagfile,"%f",&slagreact);
        waterc[SLAG]=0.0;
        heatf[SLAG]=0.0;
        heatf[SLAGCSH]=0.0;
        fclose(slagfile);
        /* Compute slag probabilities as defined above */
        CHperslag=siperslag*(slaghydcasi-slagcasi)+3.*slagc3a;
        if(CHperslag<0.0){CHperslag=0.0;}

p2slag=(molarv[SLAG]+molarv[CH]*CHperslag+molarv[POROSITY]*(waterc[SLAGCSH]-
CHperslag+waterc[C3AH6]*slagc3a)-molarv[SLAGCSH]-
molarv[C3AH6]*slagc3a)/molarv[SLAG];
        p1slag=1.0-p2slag;
        p3slag=(molarv[SLAGCSH]/molarv[SLAG])-p1slag;
```

```
        p4slag=CHperslag*molarv[CH]/molarv[SLAG];
        p5slag=slagc3a*molarv[C3A]/molarv[SLAG];
        if(p5slag>1.0){
           p5slag=1.0;
           printf("Error in range of C3A/slag value...  reset to 1.0 \n");
        }
}

/* routine to check if a pixel located at (xck,yck,zck) is on an edge */
/* (in contact with pore space) in 3-D system */
/* Called by passone */
/* Calls no other routines */
int chckedge(xck,yck,zck)
        int xck,yck,zck;
{
        int edgeback,x2,y2,z2;
        int ip;

        edgeback=0;

        /* Check all six neighboring pixels */
        /* with periodic boundary conditions */
        for(ip=0;((ip<NEIGHBORS)&&(edgeback==0));ip++){

                x2=xck+xoff[ip];
                y2=yck+yoff[ip];
                z2=zck+zoff[ip];
                if(x2>=SYSIZE){x2=0;}
                if(x2<0){x2=SYSIZEM1;}
                if(y2>=SYSIZE){y2=0;}
                if(y2<0){y2=SYSIZEM1;}
                if(z2>=SYSIZE){z2=0;}
                if(z2<0){z2=SYSIZEM1;}
                if(mic [x2] [y2] [z2]==POROSITY){
                        edgeback=1;
                }
        }
        return(edgeback);
}

/* routine for first pass through microstructure during dissolution */
/* low and high indicate phase ID range to check for surface sites */
/* Called by dissolve */
/* Calls chckedge */
void passone(low,high,cycid,cshexflag)
        int low,high,cycid,cshexflag;
{
        int i,xid,yid,zid,phid,iph,edgef,phread,cshcyc;

        /* gypready used to determine if any soluble gypsum remains */
        if((low<=GYPSUM)&&(GYPSUM<=high)){
                gypready=0;
        }
        /* Zero out count for the relevant phases */
        for(i=low;i<=high;i++){
            count[i]=0;
        }
```

```
        /* Scan the entire 3-D microstructure */
        for(xid=0;xid<SYSIZE;xid++){
        for(yid=0;yid<SYSIZE;yid++){
        for(zid=0;zid<SYSIZE;zid++){

             phread=mic[xid][yid][zid];
             /* Update heat data and water consumed for solid CSH */
             if((cshexflag==1)&&(phread==CSH)){
                   cshcyc=cshage[xid][yid][zid];
                   heatsum+=heatf[CSH]/molarvcsh[cshcyc];
                   molesh2o+=watercsh[cshcyc]/molarvcsh[cshcyc];
             }
        /* Identify phase and update count */
        phid=60;
        for(i=low;((i<=high)&&(phid==60));i++){

                if(mic [xid][yid][zid]==i){
                        phid=i;
                        /* Update count for this phase */
                        count[i]+=1;
                        if((i==GYPSUM)||(i==GYPSUMS)){
                                gypready+=1;
                        }
                        /* If first cycle, then accumulate initial counts */
                        if((cycid==1)||((cycid==0)&&(ncyc==0))){
                                if(i==POROSITY){porinit+=1;}
                        /* Ordered in terms of likely volume fractions */
                        /* (largest to smallest) to speed execution */
                                else if(i==C3S){c3sinit+=1;}
                                else if(i==C2S){c2sinit+=1;}
                                else if(i==C3A){c3ainit+=1;}
                                else if(i==C4AF){c4afinit+=1;}

                                else if(i==GYPSUM){ncsbar+=1;}
                                else if(i==GYPSUMS){ncsbar+=1;}
                                else if(i==ANHYDRITE){anhinit+=1;}
                                else if(i==HEMIHYD){heminit+=1;}
                                else if(i==POZZ){nfill+=1;}
                                else if(i==SLAG){slaginit+=1;}
                                else if(i==ETTR){netbar+=1;}
                                else if(i==ETTRC4AF){netbar+=1;}
                        }
                }
        }

        if(phid!=60){
                /* If phase is soluble, see if it is in contact with porosity */
                if((cycid!=0)&&(soluble[phid]==1)){
                        edgef=chckedge(xid,yid,zid);
                        if(edgef==1){
/* Surface eligible species has an ID OFFSET greater than its original value */
                                mic [xid][yid][zid]+=OFFSET;
                        }
                }
        }
        }  /* end of zid */
        }  /* end of yid */
```

```
        }  /* end of xid */
}

/* routine to locate a diffusing CSH species near dissolution source */
/* at (xcur,ycur,zcur) */
/* Called by dissolve */
/* Calls no other routines */
int loccsh(xcur,ycur,zcur,extent)
        int xcur,ycur,zcur,extent;
{
            int xpmax,ypmax,effort,tries,xmod,ymod,zmod;
            struct ants *antnew;

            effort=0;    /* effort indicates if appropriate location found */
            tries=0;
            /* 500 tries in immediate vicinity */
            while((effort==0)&&(tries<500)){
                tries+=1;
                xmod=(-extent)+(int)((2.*(float)extent+1.)*ran1(seed));
                ymod=(-extent)+(int)((2.*(float)extent+1.)*ran1(seed));
                zmod=(-extent)+(int)((2.*(float)extent+1.)*ran1(seed));
                if(xmod>extent){xmod=extent;}
                if(ymod>extent){ymod=extent;}
                if(zmod>extent){zmod=extent;}
                xmod+=xcur;
                ymod+=ycur;
                zmod+=zcur;
                /* Periodic boundaries */
                if(xmod>=SYSIZE){xmod-=SYSIZE;}
                else if(xmod<0){xmod+=SYSIZE;}
                if(zmod>=SYSIZE){zmod-=SYSIZE;}
                else if(zmod<0){zmod+=SYSIZE;}
                if(ymod<0){ymod+=SYSIZE;}
                else if(ymod>=SYSIZE){ymod-=SYSIZE;}

                if(mic[xmod][ymod][zmod]==POROSITY){
                        effort=1;
                        mic[xmod][ymod][zmod]=DIFFCSH;
                        nmade+=1;
                        ngoing+=1;
                        /* Add this diffusing species to the linked list */
                        antnew=(struct ants *)malloc(sizeof(struct ants));
                        antnew->x=xmod;
                        antnew->y=ymod;
                        antnew->z=zmod;
                        antnew->id=DIFFCSH;
                        antnew->cycbirth=cyccnt;
                     /* Now connect this ant structure to end of linked list */
                        antnew->prevant=tailant;
                        tailant->nextant=antnew;
                        antnew->nextant=NULL;
                        tailant=antnew;
                }
        }
            return(effort);
}
```

```
/* routine to count number of pore pixels in a cube of size boxsize */
/* centered at (qx,qy,qz) */
/* Called by makeinert */
/* Calls no other routines */
int countbox(boxsize,qx,qy,qz)
        int boxsize,qx,qy,qz;
{
        int nfound,ix,iy,iz,qxlo,qxhi,qylo,qyhi,qzlo,qzhi;
        int hx,hy,hz,boxhalf;

        boxhalf=boxsize/2;
        nfound=0;
        qxlo=qx-boxhalf;
        qxhi=qx+boxhalf;
        qylo=qy-boxhalf;
        qyhi=qy+boxhalf;
        qzlo=qz-boxhalf;
        qzhi=qz+boxhalf;
        /* Count the number of requisite pixels in the 3-D cube box */
        /* using periodic boundaries */
        for(ix=qxlo;ix<=qxhi;ix++){
                hx=ix;
                if(hx<0){hx+=SYSIZE;}
                else if(hx>=SYSIZE){hx-=SYSIZE;}
        for(iy=qylo;iy<=qyhi;iy++){
                hy=iy;
                if(hy<0){hy+=SYSIZE;}
                else if(hy>=SYSIZE){hy-=SYSIZE;}
        for(iz=qzlo;iz<=qzhi;iz++){
                hz=iz;
                if(hz<0){hz+=SYSIZE;}
                else if(hz>=SYSIZE){hz-=SYSIZE;}
                /* Count if porosity, diffusing species, or empty porosity */
                if((mic [hx] [hy] [hz]<C3S)||(mic[hx] [hy] [hz]>ABSGYP)){
                        nfound+=1;
                  }
        }
        }
        }

        return(nfound);
}

/* routine to count number of special pixels in a cube of size boxsize */
/* centered at (qx,qy,qz) */
/* special pixels are those not belonging to one of the cement clinker, */
/* calcium sulfate, or pozzolanic mineral admixture phases */
/* Called by addrand */
/* Calls no other routines */
int countboxc(boxsize,qx,qy,qz)
        int boxsize,qx,qy,qz;
{
        int nfound,ix,iy,iz,qxlo,qxhi,qylo,qyhi,qzlo,qzhi;
        int hx,hy,hz,boxhalf;

        boxhalf=boxsize/2;
        nfound=0;
```

```
        qxlo=qx-boxhalf;
        qxhi=qx+boxhalf;
        qylo=qy-boxhalf;
        qyhi=qy+boxhalf;
        qzlo=qz-boxhalf;
        qzhi=qz+boxhalf;
        /* Count the number of requisite pixels in the 3-D cube box */
        /* using periodic boundaries */
        for(ix=qxlo;ix<=qxhi;ix++){
                hx=ix;
                if(hx<0){hx+=SYSIZE;}
                else if(hx>=SYSIZE){hx-=SYSIZE;}
        for(iy=qylo;iy<=qyhi;iy++){
                hy=iy;
                if(hy<0){hy+=SYSIZE;}
                else if(hy>=SYSIZE){hy-=SYSIZE;}
        for(iz=qzlo;iz<=qzhi;iz++){
                hz=iz;
                if(hz<0){hz+=SYSIZE;}
                else if(hz>=SYSIZE){hz-=SYSIZE;}
                /* Count if not cement clinker */
                if((mic [hx] [hy] [hz]<C3S)||(mic[hx] [hy] [hz]>POZZ)){
                        nfound+=1;
                  }
        }
        }
        }
        return(nfound);
}

/* routine to create ndesire pixels of empty pore space to simulate */
/* self-desiccation */
/* Called by dissolve */
/* Calls countbox */
void makeinert(ndesire)
        long int ndesire;
{
        struct togo *headtogo,*tailtogo,*newtogo,*lasttogo,*onetogo;
        long int idesire;
        int px,py,pz,placed,cntpore,cntmax;

        /* First allocate the first element of the linked list */
        headtogo=(struct togo *)malloc(sizeof(struct togo));
        headtogo->x=headtogo->y=headtogo->z=(-1);
        headtogo->npore=0;
        headtogo->nexttogo=NULL;
        headtogo->prevtogo=NULL;
        tailtogo=headtogo;
        cntmax=0;

        printf("In makeinert with %ld needed elements \n",ndesire);
        fflush(stdout);
        /* Add needed number of elements to the end of the list */
        for(idesire=2;idesire<=ndesire;idesire++){
                newtogo=(struct togo *)malloc(sizeof(struct togo));
                newtogo->npore=0;
                newtogo->x=newtogo->y=newtogo->z=(-1);
```

```
        tailtogo->nexttogo=newtogo;
        newtogo->prevtogo=tailtogo;
        tailtogo=newtogo;
}

/* Now scan the microstructure and rank the sites */
for(px=0;px<SYSIZE;px++){
for(py=0;py<SYSIZE;py++){
for(pz=0;pz<SYSIZE;pz++){
        if(mic[px][py][pz]==POROSITY){
                cntpore=countbox(cubesize,px,py,pz);
                if(cntpore>cntmax){cntmax=cntpore;}
                /* Store this site value at appropriate place in */
                /* sorted linked list */
                if(cntpore>(tailtogo->npore)){
                        placed=0;
                        lasttogo=tailtogo;
                        while(placed==0){
                                newtogo=lasttogo->prevtogo;
                                if(newtogo==NULL){
                                        placed=2;
                                }
                                else{
                                        if(cntpore<=(newtogo->npore)){
                                                placed=1;
                                        }
                                }
                                if(placed==0){
                                        lasttogo=newtogo;
                                }
                        }
                        onetogo=(struct togo *)malloc(sizeof(struct togo));
                        onetogo->x=px;
                        onetogo->y=py;
                        onetogo->z=pz;
                        onetogo->npore=cntpore;
                        /* Insertion at the head of the list */
                        if(placed==2){
                                onetogo->prevtogo=NULL;
                                onetogo->nexttogo=headtogo;
                                headtogo->prevtogo=onetogo;
                                headtogo=onetogo;
                        }
                            if(placed==1){
                                onetogo->nexttogo=lasttogo;
                                onetogo->prevtogo=newtogo;
                                lasttogo->prevtogo=onetogo;
                                newtogo->nexttogo=onetogo;
                        }
                        /* Eliminate the last element */
                        lasttogo=tailtogo;
                        tailtogo=tailtogo->prevtogo;
                        tailtogo->nexttogo=NULL;
                        free(lasttogo);
                }
        }
}
```

```
        }
        }

        /* Now remove the sites */
        /* starting at the head of the list */
        /* and deallocate all of the used memory */
        for(idesire=1;idesire<=ndesire;idesire++){
                px=headtogo->x;
                py=headtogo->y;
                pz=headtogo->z;
                if(px!=(-1)){
                        mic [px] [py] [pz]=EMPTYP;
                        count[POROSITY]-=1;
                        count[EMPTYP]+=1;
                }
                lasttogo=headtogo;
                headtogo=headtogo->nexttogo;
                free(lasttogo);
         }
        /* If only small cubes of porosity were found, then adjust */
        /* cubesize to have a more efficient search in the future */
        if(cubesize>CUBEMIN){
                if((2*cntmax)<(cubesize*cubesize*cubesize)){
                       cubesize-=2;
                }
           }
}


/* routine to add extra SLAG CSH when SLAG reacts */
/* SLAG located at (xpres,ypres,zpres) */
/* Called by dissolve */
/* Calls moveone and edgecnt */
void extslagcsh(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,action,numnear;
        long int tries;
        int mstest,mstest2;

/* first try 6 neighboring locations until  */
/*    a) successful                         */
/*    b) all 6 sites are tried or           */
/*    c) 100 tries are made                 */
/* try to grow slag C-S-H as plates         */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                action=0;
                sump*=moveone(&xchr,&ychr,&zchr,&action,sump);
                if(action==0){printf("Error in value of action in extpozz \n");}
                check=mic[xchr][ychr][zchr];
```

```
                /* Determine the direction of the neighbor selected and */
                /* the plates possible for growth */
                if(xchr!=xpres){
                  mstest=1;
                  mstest2=2;
                }
                if(ychr!=ypres){
                  mstest=2;
                  mstest2=3;
                }
                if(zchr!=zpres){
                  mstest=3;
                  mstest2=1;
                }

                /* if neighbor is porosity, locate the SLAG CSH there */
                if(check==POROSITY){

      if((faces[xpres][ypres][zpres]==0)||(mstest==faces[xpres][ypres][zpres])||(ms
test2==faces[xpres][ypres][zpres])){
                        mic[xchr][ychr][zchr]=SLAGCSH;
                        faces[xchr][ychr][zchr]=faces[xpres][ypres][zpres];
                        count[SLAGCSH]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                  }
                }
        }

        /* if no neighbor available, locate SLAG CSH at random location */
        /* in pore space */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the extra SLAG CSH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,SLAG,CSH,SLAGCSH);
                        /* Be sure that one neighboring species is CSH or */
                        /* SLAG material */
                        if((tries>5000)||(numnear<26)){
                              mic[xchr][ychr][zchr]=SLAGCSH;
                              count[SLAGCSH]+=1;
                              count[POROSITY]-=1;
                              fchr=1;
                        }
                }
        }
}
```

```
/* routine to implement a cycle of dissolution */
/* Called by main program */
/* Calls passone, loccsh, and makeinert */
void dissolve(cycle)
        int cycle;
{
        int nc3aext,ncshext,nchext,nfh3ext,ngypext,nanhext,plok,edgef;
        int nsum5,nsum4,nsum3,nsum2,nhemext,nsum6,nc4aext;
        int xpmax,ypmax,phid,phnew,plnew,cread;
        int i,xloop,yloop,zloop,ngood,ix1,iy1,xc,yc,valid,xc1,yc1;
        int iz1,zc,zc1,cycnew;
        long int ctest;
        int placed,cshrand,ntrycsh,maxsulfate,msface;
        long int ncshgo,nsurf,suminit;
        long int xext,nhgd,npchext,nslagc3a=0;
        float pdis,plfh3,fchext,fc3aext,fanhext,mass_now,mass_fa_now,tot_mass;
        float dfact,dfact1,molesdh2o,h2oinit,heat4,fhemext,fc4aext,heatfill;
        float pconvert,pc3scsh,pc2scsh,calcx,calcy,calcz,tdisfact;
        float frafm,frettr,frhyg,frtot,mc3ar,mc4ar,p3init;
        FILE *phfile,*difffile;
        struct ants *antadd;

        /* Initialize variables */
        nmade=0;
        npchext=ncshgo=cshrand=0; /* counter for number of csh diffusing species */
                        /* to be located at random locations in microstructure */

        heat_old=heat_new; /* new and old values for heat released */

        /* Initialize dissolution and phase counters */
        nsurf=0;
        for(i=0;i<=EMPTYP;i++){
                discount[i]=0;
                count[i]=0;
        }

        /* Pass one- highlight all edge points which are soluble */
        soluble[C3AH6]=0;
        heatsum=molesh2o=0.0;
        passone(0,EMPTYP,cycle,1);
        printf("Returned from passone \n");
        fflush(stdout);

        sulf_solid=count[GYPSUM]+count[GYPSUMS]+count[HEMIHYD]+count[ANHYDRITE];
        /* If first cycle, then determine all mixture proportions based */
        /* on user input and original microstructure */
        if(cycle==1){
                /* Mass of cement in system */
                cemmass=(specgrav[C3S]*(float)count[C3S]+specgrav[C2S]*
                (float)count[C2S]+specgrav[C3A]*(float)count[C3A]+
                specgrav[C4AF]*(float)count[C4AF]);
      /*    +specgrav[GYPSUM]*
                (float)count[GYPSUM]+specgrav[ANHYDRITE]*(float)
            count[ANHYDRITE]+specgrav[HEMIHYD]*(float)count[HEMIHYD]); */
                cemmasswgyp=(specgrav[C3S]*(float)count[C3S]+specgrav[C2S]*
                (float)count[C2S]+specgrav[C3A]*(float)count[C3A]+
                specgrav[C4AF]*(float)count[C4AF]+specgrav[GYPSUM]*
```

```
            (float)count[GYPSUM]+specgrav[ANHYDRITE]*(float)
            count[ANHYDRITE]+specgrav[HEMIHYD]*(float)count[HEMIHYD]);
            flyashmass=(specgrav[ASG]*(float)count[ASG]+specgrav[CAS2]*
            (float)count[CAS2]+specgrav[POZZ]*(float)count[POZZ]);
            CH_mass=specgrav[CH]*(float)count[CH];
            /* Total mass in system neglecting single aggregate */
            tot_mass=cemmass+(float)count[POROSITY]+specgrav[INERT]*
            (float)count[INERT]+specgrav[CACL2]*(float)count[CACL2]+
            specgrav[ASG]*(float)count[ASG]+
            specgrav[SLAG]*(float)count[SLAG]+
            specgrav[HEMIHYD]*(float)count[HEMIHYD]+
            specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
            specgrav[CAS2]*(float)count[CAS2]+
            specgrav[CSH]*(float)count[CSH]+
            specgrav[GYPSUM]*(float)count[GYPSUM]+
            specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
            specgrav[HEMIHYD]*(float)count[HEMIHYD]+
            specgrav[GYPSUMS]*(float)count[GYPSUMS]+
            specgrav[POZZ]*(float)count[POZZ]+CH_mass;
            /* water-to-cement ratio */
            if(cemmass!=0.0){
                  w_to_c=(float)count[POROSITY]/(cemmass+
                  specgrav[GYPSUM]*(float)count[GYPSUM]+
                  specgrav[ANHYDRITE]*(float)count[ANHYDRITE]+
                  specgrav[HEMIHYD]*(float)count[HEMIHYD]);
            }
            else{
                  w_to_c=0.0;
            }
/* totfract is the total cement volume count including calcium sulfates */
/* fractwithfill is the total count of cement and solid fillers and */
/*    mineral admixtures   DPB- 10/04 */
       totfract=count[C3S]+count[C2S];
       totfract+=(count[C3A]+count[C4AF]);
       totfract+=(count[GYPSUM]+count[ANHYDRITE]+count[HEMIHYD]);
       fractwithfill=totfract+count[CACO3]+count[SLAG]+count[INERT];
       fractwithfill+=count[POZZ]+count[CAS2]+count[ASG]+count[CACL2];
       totfract/=(float)SYSIZE;
       totfract/=(float)SYSIZE;
       totfract/=(float)SYSIZE;
       fractwithfill/=(float)SYSIZE;
       fractwithfill/=(float)SYSIZE;
       fractwithfill/=(float)SYSIZE;
       /* Adjust masses for presence of aggregates in concrete */
       mass_water=(1.-mass_agg)*(float)count[POROSITY]/tot_mass;
       mass_CH=(1.-mass_agg)*CH_mass/tot_mass;
       /* pozzolan-to-cement ratio */
       if(cemmass!=0.0){
             s_to_c=(float)(count[INERT]*specgrav[INERT]+
             count[CACL2]*specgrav[CACL2]+count[ASG]*specgrav[ASG]+
             count[CAS2]*specgrav[CAS2]+
             count[SLAG]*specgrav[SLAG]+
             count[POZZ]*specgrav[POZZ])/cemmass;
       }
       else{
             s_to_c=0.0;
       }
```

```
                /* Conversion factor to kJ/kg for heat produced */
            if(cemmass!=0.0){

      heatfill=(float)(count[INERT]+count[SLAG]+count[POZZ]+count[CACL2]+count[ASG]
+count[CAS2])/cemmass;
            }
            else{
                  heatfill=0.0;
            }
            if(w_to_c>0.01){
                      heat_cf=0.001*(0.3125+w_to_c+heatfill);
            }
            else{
                  /* Need volume per 1 gram of silica fume */

      heat_cf=0.001*((1./specgrav[POZZ])+(float)(count[POROSITY]+count[CH]+count[IN
ERT])/(specgrav[POZZ]*(float)count[POZZ]));
            }
                mass_fill_pozz=(1.-
                mass_agg)*(float)(count[POZZ]*specgrav[POZZ])/tot_mass;
                mass_fill=(1.-mass_agg)*(float)(count[INERT]*specgrav[INERT]+
                count[ASG]*specgrav[ASG]+count[SLAG]*specgrav[SLAG]+
                count[CAS2]*specgrav[CAS2]+count[POZZ]*specgrav[POZZ]+
                count[CACL2]*specgrav[CACL2])/tot_mass;
                printf("Calculated w/c is %.4f\n",w_to_c);
                printf("Calculated s/c is %.4f \n",s_to_c);
                printf("Calculated heat conversion factor is %f \n",heat_cf);
  printf("Calculated mass fractions of water and filler are %.4f  and %.4f \n",
                mass_water,mass_fill);
        }

        molesdh2o=0.0;
        alpha=0.0;      /* degree of hydration */
        /* heat4 contains measured heat release for C4AF hydration from  */
        /* Fukuhara et al., Cem. and Conc. Res. article */
        heat4=0.0;
        mass_now=0.0;    /* total cement mass corrected for hydration */
        suminit=c3sinit+c2sinit+c3ainit+c4afinit;
        /* suminit=c3sinit+c2sinit+c3ainit+c4afinit+ncsbar+anhinit+heminit; */
        /* ctest is number of gypsum likely to form ettringite */
        /* 1 unit of C3A can react with 2.5 units of Gypsum */
        ctest=count[DIFFGYP];
        printf("ctest is %ld\n",ctest);
        fflush(stdout);
        if((float)ctest>(2.5*(float)(count[DIFFC3A]+count[DIFFC4A]))){
                 ctest=2.5*(float)(count[DIFFC3A]+count[DIFFC4A]);
        }
        for(i=0;i<=EMPTYP;i++){
                if((i!=0)&&(i<=ABSGYP)&&(i!=INERTAGG)&&(i!=CSH)){
                        heatsum+=(float)count[i]*heatf[i]/molarv[i];
                        /* Tabulate moles of H2O consumed by reactions so far */
                        molesh2o+=(float)count[i]*waterc[i]/molarv[i];
                }
                /* assume that all C3A which can, does form ettringite */
                if(i==DIFFC3A){
                      heatsum+=((float)count[DIFFC3A]-
                      (float)ctest/2.5)*heatf[C3A]/molarv[C3A];
```

```
    }
    /* assume that all C4A which can, does form ettringite */
    if(i==DIFFC4A){
          heatsum+=((float)count[DIFFC4A]-
          (float)ctest/2.5)*heatf[C4AF]/molarv[C4AF];
    }
    /* assume all gypsum which can, does form ettringite */
    /* rest will remain as gypsum */
    if(i==DIFFGYP){
           heatsum+=(float)(count[DIFFGYP]-
           ctest)*heatf[GYPSUM]/molarv[GYPSUM];
      /* 3.3 is the molar expansion from GYPSUM to ETTR */
           heatsum+=(float)ctest*3.30*heatf[ETTR]/molarv[ETTR];
           molesdh2o+=(float)ctest*3.30*waterc[ETTR]/molarv[ETTR];
    }
      else if(i==DIFFCH){
           heatsum+=(float)count[DIFFCH]*heatf[CH]/molarv[CH];
           molesdh2o+=(float)count[DIFFCH]*waterc[CH]/molarv[CH];
    }
    else if(i==DIFFFH3){
           heatsum+=(float)count[DIFFFH3]*heatf[FH3]/molarv[FH3];
           molesdh2o+=(float)count[DIFFFH3]*waterc[FH3]/molarv[FH3];
    }
    else if(i==DIFFCSH){
      /* use current CSH properties */
            heatsum+=(float)count[DIFFCSH]*heatf[CSH]/molarvcsh[cycle];
     molesdh2o+=(float)count[DIFFCSH]*watercsh[cycle]/molarvcsh[cycle];
    }
    else if(i==DIFFETTR){
            heatsum+=(float)count[DIFFETTR]*heatf[ETTR]/molarv[ETTR];
           molesdh2o+=(float)count[DIFFETTR]*waterc[ETTR]/molarv[ETTR];
    }
    else if(i==DIFFCACL2){
           heatsum+=(float)count[DIFFCACL2]*heatf[CACL2]/molarv[CACL2];
        molesdh2o+=(float)count[DIFFCACL2]*waterc[CACL2]/molarv[CACL2];
    }
    else if(i==DIFFAS){
            heatsum+=(float)count[DIFFAS]*heatf[ASG]/molarv[ASG];
            molesdh2o+=(float)count[DIFFAS]*waterc[ASG]/molarv[ASG];
    }
    else if(i==DIFFCAS2){
            heatsum+=(float)count[DIFFCAS2]*heatf[CAS2]/molarv[CAS2];
           molesdh2o+=(float)count[DIFFCAS2]*waterc[CAS2]/molarv[CAS2];
    }
/* assume that all diffusing anhydrite leads to gypsum formation */
    else if(i==DIFFANH){
         heatsum+=(float)count[DIFFANH]*heatf[GYPSUMS]/molarv[GYPSUMS];
         /* 2 moles of water per mole of gypsum formed */
         molesdh2o+=(float)count[DIFFANH]*2.0/molarv[GYPSUMS];
    }
/* assume that all diffusing hemihydrate leads to gypsum formation */
    else if(i==DIFFHEM){
         heatsum+=(float)count[DIFFHEM]*heatf[GYPSUMS]/molarv[GYPSUMS];
         /* 1.5 moles of water per mole of gypsum formed */
         molesdh2o+=(float)count[DIFFHEM]*1.5/molarv[GYPSUMS];
    }
    else if(i==C3S){
```

```
     alpha+=(float)(c3sinit-count[C3S]);
     mass_now+=specgrav[C3S]*(float)count[C3S];
     heat4+=.517*(float)(c3sinit-count[C3S])*specgrav[C3S];
}
else if(i==C2S){
     alpha+=(float)(c2sinit-count[C2S]);
     mass_now+=specgrav[C2S]*(float)count[C2S];
     heat4+=.262*(float)(c2sinit-count[C2S])*specgrav[C2S];
}
else if(i==C3A){
     alpha+=(float)(c3ainit-count[C3A]);
     mass_now+=specgrav[C3A]*(float)count[C3A];
     mc3ar=(c3ainit-(float)count[C3A])/molarv[C3A];
     mc4ar=(c4afinit-(float)count[C4AF])/molarv[C4AF];
     if((mc3ar+mc4ar)>0.0){
  frhyg=(mc3ar/(mc3ar+mc4ar))*(float)count[C3AH6]/molarv[C3AH6];
     }
     else{
              frhyg=0.0;
     }
     frettr=(float)count[ETTR]/molarv[ETTR];
     frafm=3*(float)count[AFM]/molarv[AFM];
     frtot=frafm+frettr+frhyg;
     if(frtot>0.0){
        frettr/=frtot;
        frafm/=frtot;
        frhyg/=frtot;
        heat4+=frafm*1.144*(float)(c3ainit-
              count[C3A])*specgrav[C3A];
        heat4+=frhyg*0.908*(float)(c3ainit-
              count[C3A])*specgrav[C3A];
        heat4+=frettr*1.672*(float)(c3ainit-
              count[C3A])*specgrav[C3A];
     }
}
else if(i==C4AF){
        alpha+=(float)(c4afinit-count[C4AF]);
        mass_now+=specgrav[C4AF]*(float)count[C4AF];
        mc3ar=(c3ainit-(float)count[C3A])/molarv[C3A];
        mc4ar=(c4afinit-(float)count[C4AF])/molarv[C4AF];
        if((mc3ar+mc4ar)>0.0){
  frhyg=(mc4ar/(mc3ar+mc4ar))*(float)count[C3AH6]/molarv[C3AH6];
        }
        else{
              frhyg=0.0;
        }
        frettr=(float)count[ETTRC4AF]/molarv[ETTRC4AF];
        frtot=frettr+frhyg;
        if(frtot>0.0){
        frettr/=frtot;
        frhyg/=frtot;
        heat4+=frhyg*.418*(float)(c4afinit-
              count[C4AF])*specgrav[C4AF];
        heat4+=frettr*.725*(float)(c4afinit-
              count[C4AF])*specgrav[C4AF];
        }
}
```

```
/*              else if(i==GYPSUM){
                        alpha+=(float)(ncsbar-count[GYPSUM]);
                        mass_now+=specgrav[GYPSUM]*(float)count[GYPSUM];
                }   */
                  /* 0.187 kJ/g anhydrite for anhydrite --> gypsum conversion */
                  else if(i==ANHYDRITE){
                       /*  alpha+=(float)(anhinit-count[ANHYDRITE]);
                       mass_now+=specgrav[ANHYDRITE]*(float)count[ANHYDRITE]; */
                       heat4+=.187*(float)(anhinit-
                        count[ANHYDRITE])*specgrav[ANHYDRITE];
                  /* 2 moles of water consumed per mole of anhydrite reacted */
                       molesh2o+=(float)(anhinit-
                        count[ANHYDRITE])*2.0/molarv[ANHYDRITE];
                }
            /* 0.132 kJ/g hemihydrate for hemihydrate-->gypsum conversion */
                else if(i==HEMIHYD){
/*                     alpha+=(float)(heminit-count[HEMIHYD]);
                       mass_now+=specgrav[HEMIHYD]*(float)count[HEMIHYD]; */
                       heat4+=.132*(float)(heminit-
                        count[HEMIHYD])*specgrav[HEMIHYD];
                  /* 1.5 moles of water consumed per mole of anhydrite reacted */
                       molesh2o+=(float)(heminit-
                        count[HEMIHYD])*1.5/molarv[HEMIHYD];
                }
          }
      mass_fa_now = specgrav[ASG]*(float)count[ASG];
      mass_fa_now += specgrav[CAS2]*(float)count[CAS2];
      mass_fa_now += specgrav[POZZ]*(float)count[POZZ];
      if(suminit!=0){
              alpha=alpha/(float)suminit;
      }
      else{
            alpha=0.0;
      }
        /* Current degree of hydration on a mass basis */
      if(cemmass!=0.0){
              alpha_cur=1.0-(mass_now/cemmass);
      }
      else{
            alpha_cur=0.0;
      }
      if(flyashmass!=0.0){
              alpha_fa_cur=1.0-(mass_fa_now/flyashmass);
      }
      else{
            alpha_fa_cur=0.0;
      }
      h2oinit=(float)porinit/molarv[POROSITY];

      /* Assume 780 J/g S for pozzolanic reaction */
      /* Each unit of silica fume consumes 1.35 units of CH, */
      /* so divide npr by 1.35 to get silca fume which has reacted */
      heat4+=0.78*((float)npr/1.35)*specgrav[POZZ];

      /* Assume 800 J/g S for slag reaction */
      /* Seems reasonable with measurements of Biernacki and Richardson */
      heat4+=0.8*((float)nslagr)*specgrav[SLAG];
```

```
      /* Assume 800 J/g AS for stratlingite formation (DeLarrard) */
      /* Each unit of AS consumes 1.3267 units of CH, */
      /* so divide nasr by 1.3267 to get ASG which has reacted */
      heat4+=0.80*((float)nasr/1.3267)*specgrav[ASG];

      /* Should be additional code here for heat release due to CAS2 to */
      /* stratlingite conversion, but data unavailable at this time */

      /* Adjust heat sum for water left in system */
      water_left=(long int)((h2oinit-molesh2o)*molarv[POROSITY]+0.5);
      countkeep=count[POROSITY];
      heatsum+=(h2oinit-molesh2o-molesdh2o)*heatf[POROSITY];
      if(cyccnt==0){
           heatfile=fopen(heatname,"w");
  fprintf(heatfile,"Cycle time(h) alpha_vol alpha_mass heat4(kJ/kg_solid) Gsratio2
G-s_ratio\n");
           fclose(heatfile);
      }
      heat_new=heat4;     /* use heat4 for all adiabatic calculations */
                            /* due to best agreement with calorimetry data */
      if(cyccnt==0){
        chsfile=fopen(chshrname,"w");
  fprintf(chsfile,"Cycle  time(h) alpha_mass  Chemical shrinkage (ml/g cement)\n");
        fclose(chsfile);
      }
      chs_new=((float)(count[EMPTYP]+count[POROSITY]-water_left)*heat_cf/1000.);
/*    if((molesh2o>h2oinit)&&(sealed==1)){  */
      if(((water_left+water_off)<0)&&(sealed==1)){
                printf("All water consumed at cycle %d \n",cyccnt);
                fflush(stdout);
                exit(1);
      }
      /* Attempt to create empty porosity to account for self-desiccation */
      if((sealed==1)&&((count[POROSITY]-water_left)>0)){
                poretodo=(count[POROSITY]-pore_off)-(water_left-water_off);
                poretodo-=slagemptyp;
                if(poretodo>0){
                      makeinert(poretodo);
                      poregone+=poretodo;
                }
      }
      /* Output phase counts */
      /* phfile for reactant and product phases */
      if(cyccnt==0){
            phfile=fopen(phname,"w");
         fprintf(phfile,"Cycle Porosity C3S C2S C3A C4AF GYPSUM HEMIHYD ANHYDRITE
POZZ INERT SLAG ASG CAS2 CH CSH C3AH6 ETTR ETTRC4AF AFM FH3 POZZCSH SLAGCSH CACL2
FREIDEL STRAT GYPSUMS CACO3 AFMC AGG ABSGYP EMPTYP water_left \n");
      }
      else{
            phfile=fopen(phname,"a");
      }
      fprintf(phfile,"%d ",cyccnt);
            for(i=0;i<=EMPTYP;i++){
            if((i<DIFFCSH)||(i>=EMPTYP)){
                      fprintf(phfile,"%ld ",count[i]);
```

```
      }
      printf("%ld ",count[i]);
}
printf("\n");
fprintf(phfile,"%ld\n",water_left);
fclose(phfile);

if(cycle==0){
          return;
}
cyccnt+=1;
printf("Cycle %d \n",cyccnt);
fflush(stdout);
/* Update current volume count for CH */
chold=chnew;
chnew=count[CH];

/* See if ettringite is soluble yet */
/* Gypsum 80% consumed, changed 06.09.00 from 90% to 80% */
/* Gypsum 75% consumed, changed 09.09.01 from 80% to 75% */
/* or system temperature exceeds 70 C */
if(((ncsbar+anhinit+heminit)!=0.0)||(temp_cur>=70.0)){
/* Account for all sulfate sources and forms */
if((soluble[ETTR]==0)&&((temp_cur>=70.0)||(count[AFM]!=0)||
((((float)count[GYPSUM]+1.42*(float)count[ANHYDRITE]+1.4*
(float)count[HEMIHYD]+(float)count[GYPSUMS])/((float)ncsbar+
1.42*(float)anhinit+1.4*(float)heminit+((float)netbar/3.30)))<0.25))){
          soluble[ETTR]=1;
          printf("Ettringite is soluble beginning at cycle %d \n",cycle);
          /* identify all new soluble ettringite */
          passone(ETTR,ETTR,2,0);
}
} /* end of soluble ettringite test */

/* Adjust ettringite solubility */
/* if too many ettringites already in solution */
if(count[DIFFETTR]>DETTRMAX){
          disprob[ETTR]=0.0;
}
else{
          disprob[ETTR]=disbase[ETTR];
}
/* Adjust CaCl2 solubility */
/* if too many CaCl2 already in solution */
if(count[DIFFCACL2]>DCACL2MAX){
          disprob[CACL2]=0.0;
}
else{
          disprob[CACL2]=disbase[CACL2];
}
/* Adjust CaCO3 solubility */
/* if too many CaCO3 already in solution */
if((count[DIFFCACO3]>DCACO3MAX)&&(soluble[ETTR]==0)){
          disprob[CACO3]=0.0;
}
else if(count[DIFFCACO3]>(4*DCACO3MAX)){
          disprob[CACO3]=0.0;
```

```
	}
	else{
		disprob[CACO3]=disbase[CACO3];
	}
	/* Adjust solubility of CH */
	/* based on amount of CH currently diffusing */
	/* Note that CH is always soluble to allow some */
	/* Ostwald ripening of the CH crystals */
	if((float)count[DIFFCH]>=CHCRIT){
		disprob[CH]=disbase[CH]*CHCRIT/(float)count[DIFFCH];
	}
	else{
		disprob[CH]=disbase[CH];
	}
	/* Adjust solubility of CH for temperature */
	/* Fit to data provided in Taylor, Cement Chemistry */
	/* Scale to a reference temperature of 25 C */
	/* and adjust based on availability of pozzolan */
	printf("CH dissolution probability goes from %f ",disprob[CH]);
	disprob[CH]*=((A0_CHSOL-A1_CHSOL*temp_cur)/(A0_CHSOL-A1_CHSOL*25.0));
	if((ppozz>0.0)&&(nfill>0)){
	      disprob[CH]*=ppozz/PPOZZ;

	}
	printf("to %f \n",disprob[CH]);

	/* Adjust solubility of ASG and CAS2 phases */
	/* based on pH rise during hydration */
	/* To be added at a later date */
	disprob[ASG]=disbase[ASG];
	disprob[CAS2]=disbase[CAS2];
	/* Address solubility of C3AH6 */
	/* If lots of gypsum or reactive ettringite, allow C3AH6 to dissolve */
	/* to generate diffusing C3A species */
	if(((count[GYPSUM]+count[GYPSUMS])>(int)(((float)ncsbar+
	1.42*(float)anhinit+1.4*(float)heminit)*0.05))
	||(count[ETTR]>500)){
		soluble[C3AH6]=1;
		passone(C3AH6,C3AH6,2,0);
		/* Base C3AH6 solubility on maximum sulfate in solution */
		/* from gypsum or ettringite available for dissolution */

		/* The more the sulfate, the higher this solubility should be */
		maxsulfate=count[DIFFGYP];
		if((maxsulfate<count[DIFFETTR])&&(soluble[ETTR]==1)){
			maxsulfate=count[DIFFETTR];
		}
/* Adjust C3AH6 solubility based on potential gypsum which will dissolve */
		if(maxsulfate<(int)((float)gypready*disprob[GYPSUM]*
		(float)count[POROSITY]/1000000.)){
			maxsulfate=(int)((float)gypready*disprob[GYPSUM]*
			(float)count[POROSITY]/1000000.);
		}
		if(maxsulfate>0){
		      disprob[C3AH6]=disbase[C3AH6]*(float)maxsulfate/C3AH6CRIT;
		      if(disprob[C3AH6]>0.5){disprob[C3AH6]=0.5;}
		}
```

```
                else{
                        disprob[C3AH6]=disbase[C3AH6];
                }
        }
        else{
                soluble[C3AH6]=0;
        }

        /* See if silicates are soluble yet */

if((soluble[C3S]==0)&&((cycle>1)||(count[ETTR]>0)||(count[AFM]>0)||(count[ETTRC4AF]
>0))){
                soluble[C2S]=1;
                soluble[C3S]=1;
                /* identify all new soluble silicates */
                passone(C3S,C2S,2,0);
      } /* end of soluble silicate test */
      /* Adjust solubility of C3S and C2S with CSH concentration */
      /* for simulation of induction period */

      tdisfact=A0_CHSOL-temp_cur*A1_CHSOL;
/*    printf("tdisfact is %f\n",tdisfact);
      fflush(stdout); */

   /* Calculation of cs_acc; acceleration of C3S and C2S reaction by CaSO4 */
   /* Calculation of ca_acc; acceleration of C3A and C4AF reaction by CaSO4 */
   /* November 2004 --- modified to be on a sulfate per unit initial cement */
   /*    per unit porosity basis */
   /*  Try using current porosity count to see if this helps in initial */
   /*   hydration rates of sealed vs. saturated  */
   pfract=(float)count[POROSITY]/(float)(SYSIZE*SYSIZE*SYSIZE);
   if((ncsbar+anhinit+heminit)==0.0){
                cs_acc=1.0;
                ca_acc=1.0;
                dismin_c3a=5.0*DISMIN_C3A_0;
                dismin_c4af=5.0*DISMIN_C4AF_0;
   }
   else{
        sulf_conc=sulf_cur*tfractw05*pfractw05/totfract/pfract;
        if(sulf_conc<10.0){
                cs_acc=1.0;
                ca_acc=1.0;
                dismin_c3a=DISMIN_C3A_0;
                dismin_c4af=DISMIN_C4AF_0;
        }
        else if(sulf_conc<20.0){
                cs_acc=1.0+((sulf_conc)-10.0)/10.0;
                ca_acc=1.0;
                dismin_c3a=DISMIN_C3A_0;
                dismin_c4af=DISMIN_C4AF_0;
      }
      else{
                cs_acc=1.0+(double)log10(sulf_conc-10.0);
                ca_acc=1.0;
                dismin_c3a=(6.0-(double)log10(sulf_conc))*DISMIN_C3A_0;
                dismin_c4af=(6.0-(double)log10(sulf_conc))*DISMIN_C4AF_0;
                if(dismin_c3a<DISMIN_C3A_0){dismin_c3a=DISMIN_C3A_0;}
```

```
                if(dismin_c4af<DISMIN_C4AF_0){dismin_c4af=DISMIN_C4AF_0;}
      }
   }

      /* Suggest change WCSCALE/w_to_c to (0.3125+WCSCALE)/(0.3125+w_to_c) */
      /* to have the induction period scaling depend on volume of CSH */
      /* produced per volume (not mass) of cement */
/*
dfact=tdisfact*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(w_to_c+0.3125)
))*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(w_to_c+0.3125)))*cs_acc;*/
      /* October 2004 --- changed to truly scale with volume of cement in */
      /* system for both plain portland cements and filled systems */

dfact=tdisfact*((float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*(
(float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*cs_acc;
       disprob[C3S]=DISMIN+dfact*disbase[C3S];
       disprob[C2S]=DISMIN2+dfact*disbase[C2S];
       if(disprob[C3S]>(1.*disbase[C3S])){disprob[C3S]=(1.*disbase[C3S]);}
       if(disprob[C2S]>(1.*disbase[C2S])){disprob[C2S]=(1.*disbase[C2S]);}

       /* Also adjust slag and fly ash dissolution rates here */
       /* Really slow down initial slag and fly ash dissolutions */
       /* Ultimately should be linked to pH of pore solution, most likely */
       disprob[SLAG]=slagreact*(DISMINSLAG+dfact*disbase[SLAG])/10.0;

if(disprob[SLAG]>(slagreact*disbase[SLAG])){disprob[SLAG]=(slagreact*disbase[SLAG])
;}
       if(disprob[C3S]==disbase[C3S]){disprob[SLAG]=slagreact*disbase[SLAG];}
       disprob[ASG]=DISMINASG+dfact*disbase[ASG]/5.0;
       if(disprob[ASG]>(1.*disbase[ASG])){disprob[ASG]=(1.*disbase[ASG]);}
       if(disprob[C3S]==disbase[C3S]){disprob[ASG]=disbase[ASG];}
       disprob[CAS2]=DISMINCAS2+dfact*disbase[CAS2]/5.0;
       if(disprob[CAS2]>(1.*disbase[CAS2])){disprob[CAS2]=(1.*disbase[CAS2]);}
       if(disprob[C3S]==disbase[C3S]){disprob[CAS2]=disbase[CAS2];}
       /* Adjust CAS2 solubility */
       /* if too many CAS2 already in solution */
       if(count[DIFFCAS2]>DCAS2MAX){
                disprob[CAS2]=0.0;
       }
    printf("Silicate probabilities: %f %f\n",disprob[C3S],disprob[C2S]);
    fflush(stdout);
       /* Assume that aluminate dissolution controlled by formation */
       /* of impermeable layer proportional to CSH concentration */
       /* if sulfates are present in the system */

/*
dfact1=tdisfact*((float)count[CSH]/CSHSCALE)*((float)count[CSH]/CSHSCALE)*ca_acc;
*/
    if((ncsbar+heminit+anhinit)>1000){
/*
dfact1=tdisfact*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(0.3125+w_to_c
)))*((float)count[CSH]/((float)CSHSCALE*(0.3125+WCSCALE)/(0.3125+w_to_c)))*ca_acc;
*/
      /* October 2004 --- changed to truly scale with volume of cement in */
      /* system for both plain portland cements and filled systems */
```

```
dfact1=tdisfact*((float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*
((float)count[CSH]/((float)CSHSCALE*surffract*totfract/tfractw04))*ca_acc;
    disprob[C3A]=dismin_c3a+dfact1*disbase[C3A];
    disprob[C4AF]=dismin_c4af+dfact1*disbase[C4AF];
    if(disprob[C3A]>(1.*disbase[C3A])){disprob[C3A]=(1.*disbase[C3A]);}
    if(disprob[C4AF]>(1.*disbase[C4AF])){disprob[C4AF]=(1.*disbase[C4AF]);}

    /* Location to add in dissolution reduction in calcium sulfate phases */
    /* if needed */
    disprob[GYPSUM]=(disbase[GYPSUM]/15.)+dfact1*disbase[GYPSUM];
    if(disprob[GYPSUM]>(disbase[GYPSUM])){disprob[GYPSUM]=(disbase[GYPSUM]);}
    disprob[GYPSUMS]=(disbase[GYPSUMS]/15.)+dfact1*disbase[GYPSUMS];
    if(disprob[GYPSUMS]>(disbase[GYPSUMS])){disprob[GYPSUMS]=(disbase[GYPSUMS]);}
    /* Adjust gypsum solubility */
    /* if too many diffusing gypsums already in solution */
    if(count[DIFFGYP]>DGYPMAX){
           disprob[GYPSUM]=disprob[GYPSUMS]=0.0;
    }
    disprob[HEMIHYD]=(disbase[HEMIHYD]/15.)+dfact1*disbase[HEMIHYD];
    if(disprob[HEMIHYD]>(disbase[HEMIHYD])){disprob[HEMIHYD]=(disbase[HEMIHYD]);}
    disprob[ANHYDRITE]=(disbase[ANHYDRITE]/15.)+dfact1*disbase[ANHYDRITE];

if(disprob[ANHYDRITE]>(disbase[ANHYDRITE])){disprob[ANHYDRITE]=(disbase[ANHYDRITE])
;}

    }
    else{
      /* Cause flash set by increasing dissolution rates of C3A and C4AF */
      /* each by a factor of four */
      disprob[C3A]=4.*disbase[C3A];
      disprob[C4AF]=4.*disbase[C4AF];
      disprob[GYPSUM]=disbase[GYPSUM];
      disprob[HEMIHYD]=disbase[HEMIHYD];
      disprob[ANHYDRITE]=disbase[ANHYDRITE];
    }
    /* Reduce dissolution probabilities based on saturation of system */
    if((count[EMPTYP]>0)&&((count[POROSITY]+count[EMPTYP])<220000)){
        if(countpore==0){countpore=count[EMPTYP];}
        saturation=(float)(count[POROSITY])/(float)(count[POROSITY]+(count[EMPTYP]-
countpore));
        /* Roughly according to results of Jensen in CCR, powers for RH
           sensitivity are:
            C3S-19
            C2S-29
            C3A, C4AF-6    */
        /* Adjust fly ash silicates (ASG and CAS2) and pozzolanic reactivity */
        /* by same factor as C3S (also CH) */
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
        disprob[C3S]*=(saturation*saturation);
```

```
disprob[C3S]*=(saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation*saturation);
disprob[SLAG]*=(saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation*saturation);
disprob[CH]*=(saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation*saturation);
disprob[ASG]*=(saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation*saturation);
disprob[CAS2]*=(saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation*saturation);
ppozz*=(saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
disprob[C2S]*=(saturation*saturation);
```

```
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation*saturation);
        disprob[C2S]*=(saturation);
        disprob[C3A]*=(saturation*saturation);
        disprob[C3A]*=(saturation*saturation);
        disprob[C3A]*=(saturation*saturation);
        disprob[C4AF]*=(saturation*saturation);
        disprob[C4AF]*=(saturation*saturation);
        disprob[C4AF]*=(saturation*saturation);
    }
      printf("Silicate and aluminate probabilities: %f %f %f
%f\n",disprob[C3S],disprob[C2S],disprob[C3A],disprob[C4AF]);
      printf("cs_acc is %f  and ca_acc is %f sulf_cur is
%ld\n",cs_acc,ca_acc,sulf_cur);
      fflush(stdout);
      /* Pass two- perform the dissolution of species */
      /* Determine the pH factor to use */
      pHfactor=0.0;

if((pHactive==1)&&(count[CSH]>((CSHSCALE*surffract*surffract*totfract*totfract/tfra
ctw04/tfractw04)/8.0))){
         pHfactor=1.5;
         if(pH_cur>12.5){pHfactor=1.0;}
            /* 2/02*/
/*          pHfactor=0.30*(((14.1-pH_cur)/0.3)-1.0); */
          /* 3/02*/
          /*  pHfactor=0.60*(((14.1-pH_cur)/0.5)-1.0); */
          if(pH_cur>12.75){pHfactor=0.667;}
          if(pH_cur>13.00){pHfactor=0.333;}
          if(pH_cur>13.25){pHfactor=(0.0);}
          if(pH_cur>13.75){pHfactor=(-0.25);}
          pHfactor+=concsulfate;   /* influence of sulfate on reactivity */
        }
        nhgd=0;
      /* Update molar volume ratios for CSH formation */
      pc3scsh=molarvcsh[cyccnt]/molarv[C3S]-1.0;
      pc2scsh=molarvcsh[cyccnt]/molarv[C2S]-1.0;
      /* Once again, scan all pixels in microstructure */
      slagemptyp=0;
      for(xloop=0;xloop<SYSIZE;xloop++){
      for(yloop=0;yloop<SYSIZE;yloop++){
      for(zloop=0;zloop<SYSIZE;zloop++){
                if(mic[xloop][yloop][zloop]>OFFSET){
                        phid=mic[xloop][yloop][zloop]-OFFSET;
                        /* attempt a one-step random walk to dissolve */
                        plnew=(int)((float)NEIGHBORS*ran1(seed));
                        if((plnew<0)||(plnew>=NEIGHBORS)){ plnew=NEIGHBORS-1;}
                        xc=xloop+xoff[plnew];
                        yc=yloop+yoff[plnew];
                        zc=zloop+zoff[plnew];
                        if(xc<0){xc=(SYSIZEM1);}
```

```
                        if(yc<0){yc=(SYSIZEM1);}
                        if(xc>=SYSIZE){xc=0;}
                        if(yc>=SYSIZE){yc=0;}
                        if(zc<0){zc=(SYSIZEM1);}
                        if(zc>=SYSIZE){zc=0;}

                       /* Generate probability for dissolution */
                       pdis=ran1(seed);
                       /* Bias dissolution for one pixel particles as */
                       /* indicated by a pixel value of zero in the */
                       /* particle microstructure image */

if(((pdis<=(disprob[phid]/(1.+pHfactor*pHeffect[phid])))||((pdis<=(onepixelbias*dis
prob[phid]/(1.+pHfactor*pHeffect[phid])))&&(micpart[xloop][yloop][zloop]==0)))&&(mi
c[xc][yc][zc]==POROSITY)){
                                discount[phid]+=1;
                                cread=creates[phid];
                                count[phid]-=1;
                                mic[xloop][yloop][zloop]=POROSITY;
                                if(phid==C3AH6){nhgd+=1;}
                                /* Special dissolution for C4AF */
                                if(phid==C4AF){
                                        plfh3=ran1(seed);
                                        if((plfh3<0.0)||(plfh3>1.0)){
                                                plfh3=1.0;
                                        }
                                /* For every C4AF that dissolves, 0.5453 */
                                /* diffusing FH3 species should be created */
                                        if(plfh3<=0.5453){
                                                cread=DIFFFH3;
                                        }
                                }
                                if(cread==POROSITY){
                                count[POROSITY]+=1;
                         }
                         if(cread!=POROSITY){
                                        nmade+=1;
                                        ngoing+=1;
                                        phnew=cread;
                                        count[phnew]+=1;
                                        mic[xc][yc][zc]=phnew;
                            antadd=(struct ants *)malloc(sizeof(struct ants));
                                        antadd->x=xc;
                                        antadd->y=yc;
                                        antadd->z=zc;
                                        antadd->id=phnew;
                                        antadd->cycbirth=cyccnt;
                       /* Now connect this ant structure to end of linked list */
                                        antadd->prevant=tailant;
                                        tailant->nextant=antadd;
                                        antadd->nextant=NULL;
                                        tailant=antadd;
                                 }
                        /* Extra CSH diffusing species based on current temperature
*/
                                 if((phid==C3S)||(phid==C2S)){
                                        plfh3=ran1(seed);
```

```
if(((phid==C2S)&&(plfh3<=pc2scsh))||(plfh3<=pc3scsh)){
                                        cshboxsize=int(2.+6.*(40.-temp_cur)/20.);
                                        if(cshboxsize<1){cshboxsize=1;}
                                              placed=loccsh(xc,yc,zc,cshboxsize);
                                              if(placed!=0){
                                                      count[DIFFCSH]+=1;
                                                      count[POROSITY]-=1;
                                              }
                                              else{
                                                      cshrand+=1;
                                              }
                                        }
                                  }

                                  if((phid==C2S)&&(pc2scsh>1.0)){
                                        plfh3=ran1(seed);
                                        if(plfh3<=(pc2scsh-1.0)){
                                          cshboxsize=int(2.+6.*(40.-temp_cur)/20.);
                                          if(cshboxsize<1){cshboxsize=1;}
                                              placed=loccsh(xc,yc,zc,cshboxsize);
                                              if(placed!=0){
                                                      count[DIFFCSH]+=1;
                                                      count[POROSITY]-=1;
                                              }
                                              else{
                                                      cshrand+=1;
                                              }
                                        }
                                  }
                            }
                            else{
                                  mic[xloop][yloop][zloop]-=OFFSET;
                            }

      } /* end of if edge loop */
      /* Now check if CSH to pozzolanic CSH conversion is possible */
      /* Only if CH is less than 15% in volume */
      /* Only if CSH is in contact with at least one porosity */
      /* and user wishes to use this option */

if((count[POZZ]>=13000)&&(chnew<(0.15*SYSIZE*SYSIZE*SYSIZE))&&(csh2flag==1)){
            if(mic[xloop][yloop][zloop]==CSH){
            if((countbox(3,xloop,yloop,zloop))>=1){
                  pconvert=ran1(seed);
                  if(pconvert<PCSH2CSH){
                        count[CSH]-=1;
                        plfh3=ran1(seed);
                        /* molarvcsh units of C1.7SHx goes to */
                        /* 101.81 units of C1.1SH3.9 */
                        /* with 19.86 units of CH */
                        /* so p=calcy */
                        calcz=0.0;
                        cycnew=cshage[xloop][yloop][zloop];
                        calcy=molarv[POZZCSH]/molarvcsh[cycnew];
                        if(calcy>1.0){
                              calcz=calcy-1.0;
```

```
                                    calcy=1.0;
                                    printf("Problem of not creating enough
pozzolanic CSH during CSH conversion \n");
                                    printf("Current temperature is %f
C\n",temp_cur);
                              }

                              if(plfh3<=calcy){
                                    mic[xloop][yloop][zloop]=POZZCSH;
                                    count[POZZCSH]+=1;
                              }
                              else{
                                    mic[xloop][yloop][zloop]=DIFFCH;
                                    nmade+=1;
                                    ncshgo+=1;
                                    ngoing+=1;
                                    count[DIFFCH]+=1;
                                 antadd=(struct ants *)malloc(sizeof(struct ants));
                                    antadd->x=xloop;
                                    antadd->y=yloop;
                                    antadd->z=zloop;
                                    antadd->id=DIFFCH;
                                    antadd->cycbirth=cyccnt;
                        /* Now connect this ant structure to end of linked list */
                                    antadd->prevant=tailant;
                                    tailant->nextant=antadd;
                                    antadd->nextant=NULL;
                                    tailant=antadd;
                              }
                              /* Possibly need even more pozzolanic CSH */
                              /* Would need a diffusing pozzolanic
CSH species??? */
/*                            if(calcz>0.0){
                                    plfh3=ran1(seed);
                                    if(plfh3<=calcz){
                                          cshrand+=1;
                                    }
                              }    */


                              plfh3=ran1(seed);
                              calcx=(19.86/molarvcsh[cycnew])-(1.-calcy);
                              /* Ex. 0.12658=(19.86/108.)-(1.-0.94269) */
                              if(plfh3<calcx){
                                    npchext+=1;
                              }
                        }
                  }
                  }
            }
                /* See if slag can react --- in contact with at least one porosity
*/
            if(mic[xloop][yloop][zloop]==SLAG){
                  if((countbox(3,xloop,yloop,zloop))>=1){
                        pconvert=ran1(seed);
                        if(pconvert<(disprob[SLAG]/(1.+pHfactor*pHeffect[SLAG]))){
                                    nslagr+=1;
```

```
                                        count[SLAG]-=1;
                                        discount[SLAG]+=1;
                                        /* Check on extra C3A generation */
                                        plfh3=ran1(seed);
                                        if(plfh3<p5slag){
                                            nslagc3a+=1;
                                        }
                                        /* Convert slag to reaction products */
                                        plfh3=ran1(seed);
                                        if(plfh3<p1slag){
                                             mic[xloop][yloop][zloop]=SLAGCSH;
                                  /* Assign a plate axes identifier to this slag C-S-H
voxel */
                                             msface=(int)(3.*ran1(seed)+1.);
                                             if(msface>3){msface=1;}
                                               faces[xloop][yloop][zloop]=msface;
                                               count[SLAGCSH]+=1;
                                             }
                                             else{
                                             if(sealed==1){
                                           /* Create empty porosity at slag site */
                                                  slagemptyp+=1;
                                                  mic[xloop][yloop][zloop]=EMPTYP;
                                                  count[EMPTYP]+=1;
                                  }
                                  else{
                                        mic[xloop][yloop][zloop]=POROSITY;
                                        count[POROSITY]+=1;
                                  }
                                }
                                        /* Add in extra SLAGCSH as needed */
                                        p3init=p3slag;
                                        while(p3init>1.0){
                                             extslagcsh(xloop,yloop,zloop);
                                             p3init-=1.0;
                                        }
                                        plfh3=ran1(seed);
                                        if(plfh3<p3init){
                                             extslagcsh(xloop,yloop,zloop);
                                        }
                                    }
                        }
                    }
          } /* end of zloop */
          } /* end of yloop */
          } /* end of xloop */

      if(ncshgo!=0){printf("CSH dissolved is %ld \n",ncshgo);}

      if(npchext>0){printf("npchext is %ld at cycle %d \n",npchext,cycle);}
          /* Now add in the extra diffusing species for dissolution */
          /* Expansion factors from Young and Hansen and */
          /* Mindess and Young (Concrete) */
          ncshext=cshrand;
          if(cshrand!=0){
                     printf("cshrand is %d \n",cshrand);
```

```
        }
        /* CH, Gypsum, and diffusing C3A are added at totally random */
        /* locations as opposed to at the dissolution site */
        fchext=0.61*(float)discount[C3S]+0.191*(float)discount[C2S]+
        0.2584*(float)discount[C4AF];

        nchext=fchext;
        if(fchext>(float)nchext){
                pdis=ran1(seed);
                  if((fchext-(float)nchext)>pdis){
                        nchext+=1;
                }
        }
        nchext+=npchext;
        /* Adjust CH addition for slag consumption and maintain deficit as needed
*/
        slagcum+=discount[SLAG];
        chgone=(int)(p4slag*(float)slagcum);
        nchext-=chgone;
        slagcum-=(int)((float)chgone/p4slag);
        nchext-=DIFFCHdeficit;
        DIFFCHdeficit=0;
        if(nchext<0){
               DIFFCHdeficit-=nchext;
               nchext=0;
        }
        fc3aext=discount[C3A]+0.5917*(float)discount[C3AH6];
        nc3aext=fc3aext+nslagc3a;
        if(fc3aext>(float)nc3aext){
                pdis=ran1(seed);
                if((fc3aext-(float)nc3aext)>pdis){
                        nc3aext+=1;
                }
        }
        fc4aext=0.696*(float)discount[C4AF];
        nc4aext=fc4aext;
        if(fc4aext>(float)nc4aext){
            pdis=ran1(seed);
            if((fc4aext-(float)nc4aext)>pdis){
                  nc4aext+=1;
            }
        }
        /* both forms of GYPSUM form same DIFFGYP species */
        ngypext=discount[GYPSUM]+discount[GYPSUMS];
        /* Convert to diffusing anhydrite at volume necessary for final */
        /* gypsum formation  (1 anhydrite --> 1.423 gypsum) */
        /* Since hemihydrate can now react with C3A, etc., can't */
        /* do expansion here any longer  7/99 */
/*    fanhext=1.423*(float)discount[ANHYDRITE]; */
        fanhext=(float)discount[ANHYDRITE];
        nanhext=fanhext;
        if(fanhext>(float)nanhext){
                pdis=ran1(seed);
                if((fanhext-(float)nanhext)>pdis){
                        nanhext+=1;
                }
        }
```

```
        /* Convert to diffusing hemiydrate at volume necessary for final */
        /* gypsum formation  (1 hemihydrate --> 1.4 gypsum) */
        /* Since hemihydrate can now react with C3A, etc., can't */
        /* do expansion here any longer  7/99 */
        fhemext=(float)discount[HEMIHYD];
/*      fhemext=1.3955*(float)discount[HEMIHYD];  */

        nhemext=fhemext;
        if(fhemext>(float)nhemext){
                pdis=ran1(seed);
                  if((fhemext-(float)nhemext)>pdis){
                        nhemext+=1;
                }
        }
        count[DIFFGYP]+=ngypext;
        count[DIFFANH]+=nanhext;
        count[DIFFHEM]+=nhemext;
        count[DIFFCH]+=nchext;
        count[DIFFCSH]+=ncshext;
        count[DIFFC3A]+=nc3aext;
        count[DIFFC4A]+=nc4aext;

        nsum2=nchext+ncshext;
        nsum3=nsum2+nc3aext;
        nsum4=nsum3+nc4aext;
        nsum5=nsum4+ngypext;
        nsum6=nsum5+nhemext;
        fflush(stdout);
        for(xext=1;xext<=(nsum6+nanhext);xext++){
        plok=0;
        do{
                xc=(int)((float)SYSIZE*ran1(seed));
                yc=(int)((float)SYSIZE*ran1(seed));
                zc=(int)((float)SYSIZE*ran1(seed));
                if(xc>=SYSIZE){xc=0;}
                if(yc>=SYSIZE){yc=0;}
                if(zc>=SYSIZE){zc=0;}

                if(mic[xc][yc][zc]==POROSITY){
                        plok=1;
                        phid=DIFFCH;
                        count[POROSITY]-=1;
                        if(xext>nsum6){phid=DIFFANH;}
                        else if(xext>nsum5){phid=DIFFHEM;}
                        else if(xext>nsum4){phid=DIFFGYP;}
                        else if(xext>nsum3){phid=DIFFC4A;}
                        else if(xext>nsum2){phid=DIFFC3A;}
                        else if(xext>nchext){phid=DIFFCSH;}
                        mic[xc][yc][zc]=phid;
                        nmade+=1;
                        ngoing+=1;
                        antadd=(struct ants *)malloc(sizeof(struct ants));
                        antadd->x=xc;
                        antadd->y=yc;
                        antadd->z=zc;
                        antadd->id=phid;
                        antadd->cycbirth=cyccnt;
```

```
                    /* Now connect this ant structure to end of linked list */
                       antadd->prevant=tailant;
                       tailant->nextant=antadd;
                       antadd->nextant=NULL;
                       tailant=antadd;
                }
        } while (plok==0);

        } /* end of xext for extra species generation */

        printf("Dissolved- %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld %ld
%ld\n",count[DIFFCSH],
      count[DIFFCH],count[DIFFGYP],count[DIFFC3A],count[DIFFFH3],
      count[DIFFETTR],count[DIFFAS],count[DIFFANH],count[DIFFHEM],
      count[DIFFCAS2],count[DIFFCACL2],count[DIFFCACO3]);
      sulf_cur=count[DIFFGYP]+count[DIFFANH]+count[DIFFHEM];
/*    difffile=fopen("diffuse.out","a");
      fprintf(difffile,"%d %ld %f %f %f %f %f %f %f\n",cycle, sulf_cur, cs_acc,
ca_acc, disprob[C3S], disprob[C3A], disprob[C4AF], dfact, dfact1);

      fclose(difffile); */

            /* if too many diffusing gypsums already in solution */
      if(sulf_cur>DGYPMAX){
                disprob[GYPSUM]=disprob[GYPSUMS]=0.0;
      }
      else{
                disprob[GYPSUM]=disbase[GYPSUM];
                disprob[ANHYDRITE]=disbase[ANHYDRITE];
                disprob[HEMIHYD]=disbase[HEMIHYD];
                disprob[GYPSUMS]=disbase[GYPSUMS];
      }


      printf("C3AH6 dissolved- %ld with prob. of %f \n",nhgd,disprob[C3AH6]);
      fflush(stdout);
}

/* routine to add nneed one pixel elements of phase randid at random */
/* locations in microstructure */
/* Special features for addition of 1-pixel CACO3 and INERT particles */
/* added 5/26/2004 */
/* Called by main program */
/* Calls no other routines */
void addrand(randid,nneed)
        int randid;
        long int nneed;
{
        int ix,iy,iz;
        long int ic;
        int success,cpores;

        /* Add number of requested phase pixels at random pore locations */
        for(ic=1;ic<=nneed;ic++){
                success=0;
                while(success==0){
                        ix=(int)((float)SYSIZE*ran1(seed));
```

```
                        iy=(int)((float)SYSIZE*ran1(seed));
                        iz=(int)((float)SYSIZE*ran1(seed));
                        if(ix==SYSIZE){ix=0;}
                        if(iy==SYSIZE){iy=0;}
                        if(iz==SYSIZE){iz=0;}
                        if(mic[ix][iy][iz]==POROSITY){
                        if((randid!=CACO3)&&(randid!=INERT)){
                                mic[ix][iy][iz]=randid;
                                micorig[ix][iy][iz]=randid;
                                success=1;
                      }
                      else{
                                cpores=countboxc(3,ix,iy,iz);
                                if(cpores>=26){
                                    mic[ix][iy][iz]=randid;
                                    micorig[ix][iy][iz]=randid;
                                    success=1;
                                }
                       }
                     }
                  }
       }
}

/* Routine measuresurf to measure initial surface counts for cement */
/* and for all phases (cement= C3S, C2S, C3A, C4AF, and calcium sulfates */
void measuresurf()
{
      int sx,sy,sz,jx,jy,jz,faceid;

      for(sx=0;sx<SYSIZE;sx++){
      for(sy=0;sy<SYSIZE;sy++){
      for(sz=0;sz<SYSIZE;sz++){
           if(mic[sx][sy][sz]==POROSITY){
           for(faceid=0;faceid<6;faceid++){
              if(faceid==1){
                   jx=sx-1;
                   if(jx<0){jx=SYSIZE-1;}
                   jy=sy;
                   jz=sz;
             }
             else if(faceid==0){
                   jx=sx+1;
                   if(jx>(SYSIZE-1)){jx=0;}
                   jy=sy;
                   jz=sz;
             }
             else if(faceid==2){
                   jy=sy+1;
                   if(jy>(SYSIZE-1)){jy=0;}
                   jx=sx;
                   jz=sz;
             }
             else if(faceid==3){
                   jy=sy-1;
                   if(jy<0){jy=SYSIZE-1;}
                   jx=sx;
```

```
                    jz=sz;
               }
               else if(faceid==4){
                    jz=sz+1;
                    if(jz>(SYSIZE-1)){jz=0;}
                    jx=sx;
                    jy=sy;
               }
               else if(faceid==5){
                    jz=sz-1;
                    if(jz<0){jz=SYSIZE-1;}
                    jx=sx;
                    jy=sy;
               }
              /* If the neighboring pixel is solid, update surface counts */

      if((mic[jx][jy][jz]==C3S)||(mic[jx][jy][jz]==C2S)||(mic[jx][jy][jz]==C3A)||(m
ic[jx][jy][jz]==C4AF)||(mic[jx][jy][jz]==INERT)||(mic[jx][jy][jz]==CACO3)){
            scnttotal+=1;

      if((mic[jx][jy][jz]==C3S)||(mic[jx][jy][jz]==C2S)||(mic[jx][jy][jz]==C3A)||(m
ic[jx][jy][jz]==C4AF)){
                    scntcement+=1;
             }
             }
             }
            }

      }
      }
      }
      printf("Cement surface count is %ld \n",scntcement);
      printf("Total surface count is %ld \n",scnttotal);
      surffract=(float)scntcement/(float)scnttotal;
      printf("Surface fraction is %f \n",surffract);
      fflush(stdout);
}

/* Routine resaturate to resaturate all empty porosity */
/* and continue with hydration under saturated conditions */
void resaturate()
{
      int sx,sy,sz;
      long int nresat=0;

      for(sx=0;sx<SYSIZE;sx++){
      for(sy=0;sy<SYSIZE;sy++){
      for(sz=0;sz<SYSIZE;sz++){
            if(mic[sx][sy][sz]==EMPTYP){
                  mic[sx][sy][sz]=POROSITY;
                  nresat++;
            }
      }
      }
      }
      if(nresat>0){
            porefl1=porefl2=porefl3=1;
```

```
      }
      printf("Number resaturated is %ld \n",nresat);
      fflush(stdout);
}

/* Calls init, dissolve and addrand */
main()
{
        int ntimes,valin,nmovstep,stopflag=0;
        int cycflag,ix,iy,iz,phtodo;
        int iseed,phydfreq,oflag;
        long int nadd;
        int xpl,xph,ypl,yph,fidc3s,fidc2s,fidc3a,fidc4af,fidgyp,fidagg,ffac3a;
        int fidhem,fidanh,fidcaco3,nlen,pixtmp;
        float pnucch,pscalech,pnuchg,pscalehg,pnucfh3,pscalefh3;
        float pnucgyp,pscalegyp;
        float thtimelo,thtimehi,thtemplo,thtemphi;
        float mass_cement,mass_cem_now,mass_cur,kpozz,kslag;
        FILE *infile,*outfile,*adiafile,*thfile;
        char filei[80],fileo[80],filetemp[80];

        ngoing=0;
        porefl1=porefl2=porefl3=1;
        pore_off=water_off=0;
        cycflag=0;
        heat_old=heat_new=0.0;
        chold=chnew=0;     /* Current and previous cycle CH counts */
        time_cur=0.0;      /* Elapsed time according to maturity principles */
        cubesize=CUBEMAX;
        ppozz=PPOZZ;
        poregone=poretodo=0;
        /* Get random number seed */
        printf("Enter random number seed \n");
        scanf("%d",&iseed);
        printf("%d\n",iseed);
        seed=(&iseed);

        printf("Dissolution bias is set at %f \n",DISBIAS);
        /* Open file and read in original cement particle microstructure */
        printf("Enter name of file to read initial microstructure from \n");
        scanf("%s",filei);
        printf("%s\n",filei);
        nlen=strcspn(filei,".");
        sprintf(fileroot,"");
        strncat(fileroot,filei,nlen);
        printf("nlen is %d and fileroot is now %s \n",nlen,fileroot);
        fflush(stdout);
        /* Get phase assignments for original microstructure */
        /* to transform to needed ID values */
     printf("Enter IDs in file for C3S, C2S, C3A, C4AF, Gypsum, Hemihydrate,
Anhydrite, Aggregate CaCO3\n");
    scanf("%d %d %d %d %d %d %d %d
%d",&fidc3s,&fidc2s,&fidc3a,&fidc4af,&fidgyp,&fidhem,&fidanh,&fidagg,&fidcaco3);
       printf("%d %d %d %d %d %d %d %d
%d\n",fidc3s,fidc2s,fidc3a,fidc4af,fidgyp,fidhem,fidanh,fidagg,fidcaco3);
        printf("Enter ID in file for C3A in fly ash  (default=35)\n");
        scanf("%d",&ffac3a);
```

```
printf("%d\n",ffac3a);
fflush(stdout);

infile=fopen(filei,"r");

for(ix=0;ix<SYSIZE;ix++){
for(iy=0;iy<SYSIZE;iy++){
for(iz=0;iz<SYSIZE;iz++){
    cshage[ix][iy][iz]=0;
    faces[ix][iy][iz]=0;

    fscanf(infile,"%d",&valin);
    mic[ix][iy][iz]=valin;
    if(valin==fidc3s){
                mic[ix][iy][iz]=C3S;
    }
    else if(valin==fidc2s){
                mic[ix][iy][iz]=C2S;
    }
    else if((valin==fidc3a)||(valin==ffac3a)){
                mic[ix][iy][iz]=C3A;
    }
    else if(valin==fidc4af){
                mic[ix][iy][iz]=C4AF;
    }
    else if(valin==fidgyp){
                mic[ix][iy][iz]=GYPSUM;
    }
    else if(valin==fidanh){
                mic[ix][iy][iz]=ANHYDRITE;
    }
    else if(valin==fidhem){
                mic[ix][iy][iz]=HEMIHYD;
    }
    else if(valin==fidcaco3){
                mic[ix][iy][iz]=CACO3;
    }
    else if(valin==fidagg){
                mic[ix][iy][iz]=INERTAGG;
    }
    micorig[ix][iy][iz]=mic[ix][iy][iz];
}
}
}
fclose(infile);
fflush(stdout);

/* Now read in particle IDs from file */
printf("Enter name of file to read particle IDs from \n");
scanf("%s",filei);
printf("%s\n",filei);
infile=fopen(filei,"r");

for(ix=0;ix<SYSIZE;ix++){
for(iy=0;iy<SYSIZE;iy++){
for(iz=0;iz<SYSIZE;iz++){
        fscanf(infile,"%d",&valin);
```

```
            micpart[ix][iy][iz]=valin;
    }
    }
    }

    fclose(infile);
    fflush(stdout);

    /* Initialize counters, etc. */
    npr=nasr=nslagr=0;
    nfill=0;
    ncsbar=0;
    netbar=0;
    porinit=0;
    cyccnt=0;
    setflag=0;
    c3sinit=c2sinit=c3ainit=c4afinit=anhinit=heminit=slaginit=0;

    /* Initialize structure for ants */
    headant=(struct ants *)malloc(sizeof(struct ants));
    headant->prevant=NULL;
    headant->nextant=NULL;
    headant->x=0;
    headant->y=0;
    headant->z=0;
    headant->id=100;       /* special ID indicating first ant in list */
    headant->cycbirth=0;
    tailant=headant;

  /* Allow user to iteratively add one pixel particles of various phases */
  /* Typical application would be for addition of silica fume */
    printf("Enter number of one pixel particles to add (0 to quit) \n");
    scanf("%ld",&nadd);
    printf("%ld\n",nadd);
    while(nadd>0){
            printf("Enter phase to add \n");
            printf(" C3S 1 \n");
            printf(" C2S 2 \n");
            printf(" C3A 3 \n");
            printf(" C4AF 4 \n");
            printf(" GYPSUM 5 \n");
            printf(" HEMIHYD 6 \n");
            printf(" ANHYDRITE 7 \n");
            printf(" POZZ 8 \n");
            printf(" INERT 9 \n");
            printf(" SLAG 10 \n");
            printf(" ASG 11 \n");
            printf(" CAS2 12 \n");
            printf(" CH 13 \n");
            printf(" CSH 14 \n");
            printf(" C3AH6 15 \n");
            printf(" Ettringite 16 \n");
            printf(" Stable Ettringite from C4AF 17 \n");
            printf(" AFM 18 \n");
            printf(" FH3 19 \n");
            printf(" POZZCSH 20 \n");
            printf(" SLAGCSH 21 \n");
```

```
            printf(" CACL2 22 \n");
            printf(" Friedels salt 23 \n");
            printf(" Stratlingite 24 \n");
            printf(" Calcium carbonate 26 \n");
            scanf("%d",&phtodo);
            printf("%d \n",phtodo);
            if((phtodo<0)||(phtodo>CACO3)){
                  printf("Error in phase input for one pixel particles \n");
                  exit(1);
            }
            addrand(phtodo,nadd);
           printf("Enter number of one pixel particles to add (0 to quit) \n");
            scanf("%ld",&nadd);
            printf("%ld\n",nadd);
      }
      fflush(stdout);

      init();
      printf("After init routine \n");
      printf("Enter number of cycles to execute \n");
      scanf("%d",&ncyc);
      printf("%d \n",ncyc);
printf("Do you wish hydration under 0) saturated or 1) sealed conditions \n");
      scanf("%d",&sealed);
      printf("%d \n",sealed);
      printf("Enter max. # of diffusion steps per cycle (500) \n");
      scanf("%d",&ntimes);
      printf("%d \n",ntimes);
      printf("Enter nuc. prob. and scale factor for CH nucleation \n");
      scanf("%f %f",&pnucch,&pscalech);
      printf("%f %f \n",pnucch,pscalech);
      printf("Enter nuc. prob. and scale factor for gypsum nucleation \n");
      scanf("%f %f",&pnucgyp,&pscalegyp);
      printf("%f %f \n",pnucgyp,pscalegyp);
      printf("Enter nuc. prob. and scale factor for C3AH6 nucleation \n");
      scanf("%f %f",&pnuchg,&pscalehg);
      printf("%f %f \n",pnuchg,pscalehg);
      printf("Enter nuc. prob. and scale factor for FH3 nucleation \n");
      scanf("%f %f",&pnucfh3,&pscalefh3);
      printf("%f %f \n",pnucfh3,pscalefh3);
      printf("Enter cycle frequency for checking pore space percolation \n");
      scanf("%d",&burnfreq);
      printf("%d\n",burnfreq);
printf("Enter cycle frequency for checking percolation of solids (set) \n");
      scanf("%d",&setfreq);
      printf("%d\n",setfreq);
printf("Enter cycle frequency for checking hydration of particles \n");
      scanf("%d",&phydfreq);
      printf("%d\n",phydfreq);
printf("Enter cycle frequency for outputting hydrating microstructure \n");
      scanf("%d",&outfreq);
      printf("%d\n",outfreq);
      /* Parameters for adiabatic temperature rise calculation */
      printf("Enter the induction time in hours \n");
      scanf("%f",&ind_time);
      printf("%f \n",ind_time);
      time_cur+=ind_time;
```

```
        printf("Enter the initial temperature in degrees Celsius \n");
        scanf("%f",&temp_0);
        printf("%f \n",temp_0);
        temp_cur=temp_0;
        printf("Enter the ambient temperature in degrees Celsius \n");
        scanf("%f",&T_ambient);
        printf("%f \n",T_ambient);
        printf("Enter the overall heat transfer coefficient in J/g/C/s \n");
        scanf("%f",&U_coeff);
        printf("%f \n",U_coeff);
        printf("Enter apparent activation energy for hydration in kJ/mole \n");
        scanf("%f",&E_act);
        printf("%f \n",E_act);
        printf("Enter apparent activation energy for pozzolanic reactions in
kJ/mole \n");
        scanf("%f",&E_act_pozz);
        printf("%f \n",E_act_pozz);
        printf("Enter apparent activation energy for slag reactions in kJ/mole
\n");
        scanf("%f",&E_act_slag);
        printf("%f \n",E_act_slag);
        printf("Enter kinetic factor to convert cycles to time for 25 C \n");
        scanf("%f",&beta);
        printf("%f \n",beta);
        printf("Enter mass fraction of aggregate in concrete \n");
        scanf("%f",&mass_agg);
        printf("%f \n",mass_agg);
        printf("Hydration under 0) isothermal, 1) adiabatic or 2) programmed
temperature history conditions \n");
        scanf("%d",&adiaflag);
        printf("%d \n",adiaflag);
        if(adiaflag==2){
            thfile=fopen("temphist.dat","r");
            fscanf(thfile,"%f %f %f %f",&thtimelo,&thtimehi,&thtemplo,&thtemphi);
            printf("%f %f %f %f\n",thtimelo,thtimehi,thtemplo,thtemphi);
        }
        printf("CSH to pozzolanic CSH 0) prohibited or 1) allowed \n");
        scanf("%d",&csh2flag);
        printf("%d \n",csh2flag);
      printf("CH precipitation on aggregate surfaces 0) prohibited or 1) allowed
\n");
        scanf("%d",&chflag);
        printf("%d \n",chflag);
        printf("Number of slices in hydration movie \n");
        scanf("%d",&nummovsl);
        printf("%d \n",nummovsl);
        nmovstep=1;
        if(nummovsl>0){
            nmovstep=ncyc/nummovsl;
                if(nmovstep<1){nmovstep=1;}
        }
        printf("Dissolution bias factor for one-pixel particles \n");
        scanf("%f",&onepixelbias);
        printf("%f\n",onepixelbias);
        printf("Enter number of cycles before executing total resaturation \n");
        scanf("%d\n",&resatcyc);
```

```
        printf("%d\n",resatcyc);

        printf("Enter choice for C-S-H geometry 0) random or 1) plates \n");
        scanf("%d\n",&cshgeom);
        printf("%d \n",cshgeom);
        printf("Does pH influence hydration kinetics 0) no or 1) yes \n");
        scanf("%d\n",&pHactive);
        printf("%d\n",pHactive);
        fflush(stdout);

sprintf(heatname,"%s.heat.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaf
lag,sealed);

sprintf(moviename,"%s.mov.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaf
lag,sealed);

sprintf(chshrname,"%s.chs.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaf
lag,sealed);

sprintf(adianame,"%s.adi.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafl
ag,sealed);

sprintf(parname,"%s.par.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
      /* Store filename for pH file and initialize with column headings */

sprintf(pHname,"%s.phv.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaflag
,sealed);
/*      pHfile=fopen(pHname,"w");
        fprintf(pHfile,"Cycle time(h) alpha_mass pH sigma [Na+] [K+] [Ca++] [SO4--]
activityCa activityOH activitySO4 activityK molesSyngenite\n");
        fclose(pHfile); */

sprintf(fileo,"%s.img.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaflag,
sealed);

sprintf(phname,"%s.pha.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiaflag
,sealed);

sprintf(ppsname,"%s.pps.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
        /* Store parameters input in parameter file */
        sprintf(cmdnew,"cp disrealnew.out %s",parname);
        system(cmdnew);
        if(burnfreq<=ncyc){
           ptmpfile=fopen(ppsname,"w");
           fprintf(ptmpfile,"Cycle time(h) alpha_mass conn_por total_por
frac_conn\n");
           fclose(ptmpfile);
       }

sprintf(ptsname,"%s.pts.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
        if(setfreq<=ncyc){
           ptmpfile=fopen(ptsname,"w");
           fprintf(ptmpfile,"Cycle time(h) alpha_mass conn_solid total_solid
frac_conn\n");
```

```
           fclose(ptmpfile);
       }

sprintf(phrname,"%s.phr.%d.%d.%1d%1d%1d",fileroot,ncyc,(int)temp_0,csh2flag,adiafla
g,sealed);
        krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Determine pozzolanic and slag reaction rate constants */
        kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        kslag=exp(-(1000.*E_act_slag/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Update probability of pozzolanic reaction */
        /* based on ratio of pozzolanic reaction rate to hydration rate */
        ppozz=PPOZZ*kpozz/krate;
        /* Assume same holds for dissolution of fly ash phases */
        disprob[ASG]=disbase[ASG]*kpozz/krate;
        disprob[CAS2]=disbase[CAS2]*kpozz/krate;
        /* Update probability of slag dissolution */
        disprob[SLAG]=slagreact*disbase[SLAG]*kslag/krate;
        printf("%s\n",adianame);
        fflush(stdout);
        adiafile=fopen(adianame,"w");
fprintf(adiafile,"Time(h) Temperature  Alpha  Krate   Cp_now  Mass_cem kpozz/khyd
kslag/khyd\n");
        /* Set initial properties of CSH */
        molarvcsh[0]=molarv[CSH];
        watercsh[0]=waterc[CSH];
        /* Determine surface counts */

        measuresurf();
        for(icyc=1;icyc<=ncyc;icyc++){
            if((sealed==1)&&(icyc==(resatcyc+1))&&(resatcyc!=0)){
                  resaturate();
                  sealed=0;
            }
        if(temp_cur<=80.0){
            molarvcsh[icyc]=molarv[CSH]-8.0*((temp_cur-20.)/(80.-20.));
            watercsh[icyc]=waterc[CSH]-1.3*((temp_cur-20.)/(80.-20.));
        }
        else{
            molarvcsh[icyc]=molarv[CSH]-8.0;
            watercsh[icyc]=waterc[CSH]-1.3;
        }
                if(icyc==ncyc){cycflag=1;}
                printf("Calling dissolve \n");
                fflush(stdout);
                dissolve(icyc);
printf("Number dissolved this pass- %ld total diffusing- %ld \n",nmade,ngoing);
                fflush(stdout);
                if(icyc==1){
                     printf("ncsbar is %ld   netbar is %ld \n",ncsbar,netbar);
                }

hydrate(cycflag,ntimes,pnucch,pscalech,pnuchg,pscalehg,pnucfh3,pscalefh3,pnucgyp,ps
calegyp);
                printf("Returned from hydrate \n");
                fflush(stdout);
                temp_0=temp_cur;
                /* Handle adiabatic case first */
```

```
            /* Cement + aggregate +water + filler=1;  that's all there is */
            mass_cement=1.-mass_agg-mass_fill-mass_water-mass_CH;
            mass_cem_now=mass_cement;
            if(adiaflag==1){
                    /* determine heat capacity of current mixture, */
                    /* accounting for imbibed water if necessary */
                    if(sealed==1){
                            Cp_now=mass_agg*Cp_agg;
                            Cp_now+=Cp_pozz*mass_fill;
                            Cp_now+=Cp_cement*mass_cement;
                            Cp_now+=Cp_CH*mass_CH;
  Cp_now+=(Cp_h2o*mass_water-alpha_cur*WN*mass_cement*(Cp_h2o-Cp_bh2o));
                            mass_cem_now=mass_cement;
                      }
            /* Else need to account for extra capillary water drawn in */
    /* Basis is WCHSH(0.06) g H2O per gram cement for chemical shrinkage */
            /* Need to adjust mass basis to account for extra imbibed H2O */
              else{
                            mass_cur=1.+WCHSH*mass_cement*alpha_cur;
                            Cp_now=mass_agg*Cp_agg/mass_cur;
                            Cp_now+=Cp_pozz*mass_fill/mass_cur;
                            Cp_now+=Cp_cement*mass_cement/mass_cur;
                            Cp_now+=Cp_CH*mass_CH/mass_cur;
    Cp_now+=(Cp_h2o*mass_water-alpha_cur*WN*mass_cement*(Cp_h2o-Cp_bh2o));
                            Cp_now+=(WCHSH*Cp_h2o*alpha_cur*mass_cement);
                            mass_cem_now=mass_cement/mass_cur;
                      }
            /* Determine rate constant based on Arrhenius expression */
            /* Recall that temp_cur is in degrees Celsius */
        krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Determine pozzolanic and slag reaction rate constant */
     kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
     kslag=exp(-(1000.*E_act_slag/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
        /* Update probability of pozzolanic and slag reactions */
/* based on ratio of pozzolanic (slag) reaction rate to hydration rate */
        ppozz=PPOZZ*kpozz/krate;
        disprob[ASG]=disbase[ASG]*kpozz/krate;
        disprob[CAS2]=disbase[CAS2]*kpozz/krate;
        disprob[SLAG]=slagreact*disbase[SLAG]*kslag/krate;

            /* Update temperature based on heat generated and current Cp */
        if(mass_cem_now>0.01){
                    temp_cur=temp_0+mass_cem_now*heat_cf*(heat_new-
                          heat_old)/Cp_now;
        }
        else{
                    temp_cur=temp_0+mass_fill_pozz*heat_cf*(heat_new-
                          heat_old)/Cp_now;

        }
        /* Update system temperature due to heat loss/gain to/from */
        /* surroundings (semi-adiabatic case)  */
        temp_cur-=(temp_cur-T_ambient)*time_step*U_coeff/Cp_now;
        }
        else if(adiaflag==2){
              /* Update system temperature based on current time */
              /* and requested temperature history */
```

```
                  while((time_cur>thtimehi)&&(!feof(thfile))){
                        fscanf(thfile,"%f %f %f
%f",&thtimelo,&thtimehi,&thtemplo,&thtemphi);
                        printf("New temperature history values : \n");
                        printf("%f %f %f
%f\n",thtimelo,thtimehi,thtemplo,thtemphi);
                  }
            if((thtimehi-thtimelo)>0.0){
                  temp_cur=thtemplo+(thtemphi-thtemplo)*(time_cur-
thtimelo)/(thtimehi-thtimelo);
            }
            else{
                  temp_cur=thtemplo;
            }
         krate=exp(-(1000.*E_act/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
         kpozz=exp(-(1000.*E_act_pozz/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
         kslag=exp(-(1000.*E_act_slag/8.314)*((1./(temp_cur+273.15))-(1./298.15)));
            ppozz=PPOZZ*kpozz/krate;
            disprob[ASG]=disbase[ASG]*kpozz/krate;
            disprob[CAS2]=disbase[CAS2]*kpozz/krate;
            disprob[SLAG]=slagreact*disbase[SLAG]*kslag/krate;
            }
                /* Update time based on simple numerical integration */
                /* simulating maturity approach */
                /* with parabolic kinetics (Knudsen model) */
                if(cyccnt>1){
                  time_cur+=(2.*(float)(cyccnt-1)-1.0)*beta/krate;
                  time_step=(2.*(float)(cyccnt-1)-1.0)*beta/krate;
            }
                fprintf(adiafile,"%f %f %f %f %f %f %f %f\n",time_cur,temp_cur,
                 alpha_cur,krate,Cp_now,mass_cem_now,kpozz/krate,kslag/krate);
                fflush(adiafile);
                gsratio2=0.0;
                gsratio2+=(float)(count[CH]+count[CSH]+count[C3AH6]+count[ETTR]);

gsratio2+=(float)(count[POZZCSH]+count[SLAGCSH]+count[FH3]+count[AFM]+count[ETTRC4A
F]);

gsratio2+=(float)(count[FREIDEL]+count[STRAT]+count[ABSGYP]+count[AFMC]);

gsratio2=(gsratio2)/(gsratio2+(float)(count[POROSITY]+count[EMPTYP]));
                heatfile=fopen(heatname,"a");
                if(w_to_c!=0.0){
                          fprintf(heatfile,"%d %f %f %f %f %f  %f \n",
                     cyccnt-
1,time_cur,alpha,alpha_cur,heat_new*heat_cf,gsratio2,((0.68*alpha_cur)/(0.32*alpha_
cur+w_to_c)));
                }
                else{
                        fprintf(heatfile,"%d %f %f %f %f %f  %f \n",
               cyccnt-1,time_cur,alpha,alpha_cur,heat_new*heat_cf,gsratio2,0.0);
                }
                fclose(heatfile);
                chsfile=fopen(chshrname,"a");
                fprintf(chsfile,"%d %f %f %f\n",
                           cyccnt-1,time_cur,alpha_cur,chs_new);
                fclose(chsfile);
```

```
            pHpred();
            printf("Returned from call to pH \n");
            fflush(stdout);
      /* Check percolation of pore space */
      /* Note that first variable passed corresponds to phase to check */
      /* Could easily add calls to check for percolation of CH, CSH, etc. */
      if(((icyc%burnfreq)==0)&&((porefl1+porefl2+porefl3)!=0)){
            porefl1=burn3d(0,1,0,0);
            porefl2=burn3d(0,0,1,0);
            porefl3=burn3d(0,0,0,1);
         /* Switch to self-desiccating conditions when porosity */
         /* disconnects */
         if(((porefl1+porefl2+porefl3)==0)&&(sealed==0)){
               water_off=water_left;
               pore_off=countkeep;
               sealed=1;
               printf("Switching to self-desiccating at cycle %d \n",cyccnt);
               fflush(stdout);
         }
      }
      /* Check percolation of solids (set point) */
      if(((icyc%setfreq)==0)&&(setflag==0)){
            sf1=burnset(1,0,0);
            sf2=burnset(0,1,0);
            sf3=burnset(0,0,1);
            setflag=sf1*sf2*sf3;
      }


      /* Check hydration of particles */
      if((icyc%phydfreq)==0){
            parthyd();
      }
      /* Output movie microstructure if desired */
            if((nummovsl>0)&&((icyc%nmovstep)==0)){
                     if(icyc==nmovstep){
                     movfile=fopen(moviename,"w");
                     }
                     else{
                     movfile=fopen(moviename,"a");
                     }

               for(ix=0;ix<SYSIZE;ix++){
               for(iy=0;iy<SYSIZE;iy++){
                     fprintf(movfile,"%d\n",(int)mic[50][ix][iy]);
               }
               }
               fclose(movfile);
        }
        /* Output complete microstructure every outfreq cycles */
            if((icyc>0)&&((icyc%outfreq)==0)){

sprintf(micname,"%s.ima.%d.%d.%1d%1d%1d",fileroot,icyc,(int)temp_0,csh2flag,adiafla
g,sealed);
               micfile=fopen(micname,"w");

               for(ix=0;ix<SYSIZE;ix++){
```

```
            for(iy=0;iy<SYSIZE;iy++){
            for(iz=0;iz<SYSIZE;iz++){
                        pixtmp=(int)mic[ix][iy][iz];
                        if(pixtmp==DIFFCSH){
                           pixtmp=CSH;
                        }
                        else if (pixtmp==DIFFANH){
                           pixtmp=ANHYDRITE;
                        }
                        else if (pixtmp==DIFFHEM){
                           pixtmp=HEMIHYD;
                        }
                        else if (pixtmp==DIFFGYP){
                           pixtmp=GYPSUM;
                        }
                        else if (pixtmp==DIFFCACL2){
                           pixtmp=CACL2;
                        }
                        else if (pixtmp==DIFFCACO3){
                           pixtmp=CACO3;
                        }
                        else if (pixtmp==DIFFCAS2){
                           pixtmp=CAS2;
                        }
                        else if (pixtmp==DIFFAS){
                           pixtmp=ASG;
                        }
                        else if (pixtmp==DIFFETTR){
                           pixtmp=ETTR;
                        }
                        else if (pixtmp==DIFFC3A){
                           pixtmp=C3A;
                        }
                        else if (pixtmp==DIFFC4A){
                           pixtmp=C3A;
                        }
                        else if (pixtmp==DIFFFH3){
                           pixtmp=FH3;
                        }
                        else if (pixtmp==DIFFCH){
                           pixtmp=CH;
                        }
                  fprintf(micfile,"%d\n",pixtmp);
            }
            }
            }
            fclose(micfile);
      }

  }
/* Last call to dissolve to terminate hydration */
dissolve(0);
/* Check percolation of pore space */
/* Note that first variable passed corresponds to phase to check */
/* Could easily add calls to check for percolation of CH, CSH, etc. */
if((burnfreq!=0)&&(burnfreq<=ncyc)&&((porefl1+porefl2+porefl3)!=0)){
         porefl1=burn3d(0,1,0,0);
```

```
            porefl2=burn3d(0,0,1,0);
            porefl3=burn3d(0,0,0,1);
      }
      /* Check percolation of solids (set point) */
      if((setfreq!=0)&&(setfreq<=ncyc)){
            setflag=burnset(1,0,0);
            setflag+=burnset(0,1,0);
            setflag+=burnset(0,0,1);
      }

      /* Output last lines of heat and chemical shrinkage files */
      if(cyccnt>1){
            time_cur+=(2.*(float)cyccnt-1.0)*beta/krate;
            time_step=(2.*(float)cyccnt-1.0)*beta/krate;
      }
      fprintf(adiafile,"%f %f %f %f %f %f %f %f\n",time_cur,temp_cur,
          alpha_cur,krate,Cp_now,mass_cem_now,kpozz/krate,kslag/krate);
      fflush(adiafile);
      fclose(adiafile);
      gsratio2=0.0;
      gsratio2+=(float)(count[CH]+count[CSH]+count[C3AH6]+count[ETTR]);

gsratio2+=(float)(count[POZZCSH]+count[SLAGCSH]+count[FH3]+count[AFM]+count[ETTRC4A
F]);
      gsratio2+=(float)(count[FREIDEL]+count[STRAT]+count[ABSGYP]+count[AFMC]);
      gsratio2=(gsratio2)/(gsratio2+(float)(count[POROSITY]+count[EMPTYP]));
      heatfile=fopen(heatname,"a");
      fprintf(heatfile,"%d %f %f %f %f %f %f\n",

cyccnt,time_cur,alpha,alpha_cur,heat_new*heat_cf,gsratio2,((0.68*alpha_cur)/(0.32*a
lpha_cur+w_to_c)));
      fclose(heatfile);
      chsfile=fopen(chshrname,"a");
      fprintf(chsfile,"%d %f %f %f\n",
      cyccnt,time_cur,alpha_cur,((float)(count[EMPTYP]+count[POROSITY]-
water_left)*heat_cf/1000.));
      fclose(chsfile);
      cyccnt+=1;
      pHpred();
      printf("Final count for ncshplategrow is %ld \n",ncshplategrow);
      printf("Final count for ncshplateinit is %ld \n",ncshplateinit);
      /* Output final microstructure if desired */
      outfile=fopen(fileo,"w");

      for(ix=0;ix<SYSIZE;ix++){
      for(iy=0;iy<SYSIZE;iy++){
      for(iz=0;iz<SYSIZE;iz++){
            fprintf(outfile,"%d\n",(int)mic[ix][iy][iz]);
      }
      }
      }
      fclose(outfile);
}
```

Program hydrealnew.c

```
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

#define AGRATE 0.25        /* Probability of gypsum absorption by CSH */

/* routine to select a new neighboring location to (xloc, yloc, zloc) */
/* for a diffusing species */
/* Returns a prime number flag indicating direction chosen */
/* Calls ran1 */
/* Called by movecsh, extettr, extfh3, movegyp, extafm, moveettr, */
/* extpozz, movefh3, movech, extc3ah6, movec3a */
/* extfreidel, movecacl2, extstrat, moveas */
int moveone(xloc,yloc,zloc,act,sumold)
        int *xloc,*yloc,*zloc,*act,sumold;
{
        int plok,sumnew,xl1,yl1,zl1,act1;

        sumnew=1;
        /* store the input values for location */
        xl1=(*xloc);
        yl1=(*yloc);
        zl1=(*zloc);
        act1=(*act);

        /* Choose one of six directions (at random) for the new */
        /* location */
        plok=6.*ran1(seed);
        if((plok>5)||(plok<0)){plok=5;}

        switch (plok){
                case 0:
                        xl1-=1;
                        act1=1;
                        if(xl1<0){xl1=(SYSIZEM1);}
                        if(sumold%2!=0){sumnew=2;}
                        break;
                case 1:
                        xl1+=1;
                        act1=2;
                        if(xl1>=SYSIZE){xl1=0;}
                        if(sumold%3!=0){sumnew=3;}
```

```
                        break;
                case 2:
                        yl1-=1;
                        act1=3;
                        if(yl1<0){yl1=(SYSIZEM1);}
                        if(sumold%5!=0){sumnew=5;}
                        break;
                case 3:
                        yl1+=1;
                        act1=4;
                        if(yl1>=SYSIZE){yl1=0;}
                        if(sumold%7!=0){sumnew=7;}
                        break;
                case 4:
                        zl1-=1;
                        act1=5;
                        if(zl1<0){zl1=(SYSIZEM1);}
                        if(sumold%11!=0){sumnew=11;}
                        break;
                case 5:
                        zl1+=1;
                        act1=6;
                        if(zl1>=SYSIZE){zl1=0;}
                        if(sumold%13!=0){sumnew=13;}
                        break;
                default:
                        break;
        }

        /* Return the new location */
        *xloc=xl1;
        *yloc=yl1;
        *zloc=zl1;
        *act=act1;
        /* sumnew returns a prime number indicating that a specific direction */
        /* has been chosen */
        return(sumnew);
}

/* routine to return count of number of neighboring pixels for pixel */
/* (xck,yck,zck) which are not phase ph1, ph2, or ph3 which are input as */
/* parameters */
/* Calls no other routines */
/* Called by extettr, extfh3, extch, extafm, extpozz, extc3ah6 */
/* extfreidel, extcsh, and extstrat */
int edgecnt(xck,yck,zck,ph1,ph2,ph3)
        int xck,yck,zck,ph1,ph2,ph3;
{
        int ixe,iye,ize,edgeback,x2,y2,z2,check;

/* counter for number of neighboring pixels which are not ph1, ph2, or ph3 */
        edgeback=0;

/* Examine all pixels in a 3*3*3 box centered at (xck,yck,zck) */
/* except for the central pixel */
            for(ixe=(-1);ixe<=1;ixe++){
                 x2=xck+ixe;
```

```
            for(iye=(-1);iye<=1;iye++){
                y2=yck+iye;
            for(ize=(-1);ize<=1;ize++){

                 if((ixe!=0)||(iye!=0)||(ize!=0)){
                        z2=zck+ize;
                        /* adjust to maintain periodic boundaries */
                        if(x2<0){x2=(SYSIZEM1);}
                        else if(x2>=SYSIZE){x2=0;}
                        if(y2<0){y2=(SYSIZEM1);}
                        else if(y2>=SYSIZE){y2=0;}
                        if(z2<0){z2=(SYSIZEM1);}
                        else if(z2>=SYSIZE){z2=0;}
                        check=mic[x2][y2][z2];
                        if((check!=ph1)&&(check!=ph2)&&(check!=ph3)){
                                edgeback+=1;
                        }
                }
        }
        }
        }
        /* return number of neighboring pixels which are not ph1, ph2, or ph3 */
            return(edgeback);
}

/* routine to add extra CSH when diffusing CSH reacts */
/* Called by movecsh */
/* Calls edgecnt */
void extcsh()
{
        int numnear,sump,xchr,ychr,zchr,fchr,i1,plok,check,msface;
        long int tries;

        fchr=0;
        tries=0;
        /* locate CSH at random location */
        /* in pore space in contact with at least another CSH or C3S or C2S */
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                /* if location is porosity, locate the CSH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,CSH,C3S,C2S);
                        /* be sure that at least one neighboring pixel */
                        /* is C2S, C3S, or diffusing CSH */
                        if((numnear<26)||(tries>5000)){
                           mic[xchr][ychr][zchr]=CSH;
                           count[CSH]+=1;
                           count[POROSITY]-=1;
```

```
                        cshage[xchr][ychr][zchr]=cyccnt;
                        if(cshgeom==1){
                           msface=(int)(3.*ran1(seed)+1.);
                           if(msface>3){msface=1;}
                           faces[xchr][ychr][zchr]=msface;
                           ncshplateinit+=1;
                        }
                        fchr=1;
                     }
                }
        }
}

/* routine to move a diffusing CSH species */
/* Inputs: current location (xcur,ycur,zcur) and flag indicating if final */
/* step in diffusion process */
/* Returns flag indicating action taken (reaction or diffusion/no movement) */
/* Calls moveone,extcsh */
/* Called by hydrate */
int movecsh(xcur,ycur,zcur,finalstep,cycorig)
        int xcur,ycur,zcur,finalstep,cycorig;
{
        int xnew,ynew,znew,plok,action,sumback,sumin,check;
        int msface,mstest,mstest2;
        float prcsh,prcsh1,prcsh2,prtest;

        action=0;
        /* Store current location of species */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        sumin=1;
        sumback=moveone(&xnew,&ynew,&znew,&action,sumin);
        if(cshgeom==1){
        /* Determine eligible faces based on direction of move */
            if(xnew!=xcur){
                  mstest=1;
                  mstest2=2;
            }
            if(ynew!=ycur){
                  mstest=2;
                  mstest2=3;
            }
            if(znew!=zcur){
                  mstest=3;
                  mstest2=1;
            }
      }

        if(action==0){printf("Error in value of action \n");}
        check=mic[xnew][ynew][znew];

       /* if new location is solid CSH and plate growth is favorable, */
       /* then convert diffusing CSH species to solid CSH */
       prcsh=ran1(seed);
```

```
if((check==CSH)&&((cshgeom==0)||(faces[xnew][ynew][znew]==0)||(faces[xnew][ynew][zn
ew]==mstest)||(faces[xnew][ynew][znew]==mstest2))){
            /* decrement count of diffusing CSH species */
            count[DIFFCSH]-=1;
            /* and increment count of solid CSH if needed */
            prtest=molarvcsh[cyccnt]/molarvcsh[cycorig];
            prcsh1=ran1(seed);
            if(prcsh1<=prtest){
               mic[xcur][ycur][zcur]=CSH;
               if(cshgeom==1){
                     faces[xcur][ycur][zcur]=faces[xnew][ynew][znew];
                     ncshplategrow+=1;
               }
               cshage[xcur][ycur][zcur]=cyccnt;
               count[CSH]+=1;
            }
            else{
                  mic[xcur][ycur][zcur]=POROSITY;
                  count[POROSITY]+=1;
            }
      /* May need extra solid CSH if temperature goes down with time */
            if(prtest>1.0){
                  prcsh2=ran1(seed);
                  if(prcsh2<(prtest-1.0)){
                        extcsh();
                  }
            }
            action=0;
      }
      /* Changed prcsh limit from 0.1 to 0.01 for CH test 1/27/05 */
      else if((check==SLAGCSH)||(check==POZZCSH)||(finalstep==1)||
         (((check==C3S)||(check==C2S))&&(prcsh<0.001))||
         (((check==C3A)||(check==C4AF))&&(prcsh<0.2))||
         ((check==CH)&&(prcsh<0.01))||
         (check==CACO3)||(check==INERT)){
           /* decrement count of diffusing CSH species */
           count[DIFFCSH]-=1;
           /* and increment count of solid CSH if needed */
           prtest=molarvcsh[cyccnt]/molarvcsh[cycorig];
           prcsh1=ran1(seed);
           if(prcsh1<=prtest){
               mic[xcur][ycur][zcur]=CSH;
               cshage[xcur][ycur][zcur]=cyccnt;
               if(cshgeom==1){
                     msface=(int)(2.*ran1(seed)+1.);
                     if(msface>2){msface=1;}
                     if(msface==1){
                        faces[xcur][ycur][zcur]=mstest;
                     }
                     else{
                        faces[xcur][ycur][zcur]=mstest2;
                     }
                     ncshplateinit+=1;
               }
               count[CSH]+=1;
            }
```

```
            else{
                  mic[xcur][ycur][zcur]=POROSITY;
                  count[POROSITY]+=1;
            }
      /* May need extra solid CSH if temperature goes down with time */
            if(prtest>1.0){
                  prcsh2=ran1(seed);
                  if(prcsh2<(prtest-1.0)){
                        extcsh();
                  }
            }
            action=0;
      }

      if(action!=0){
      /* if diffusion step is possible, perform it */
                if(check==POROSITY){

                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFCSH;
                }
                else{
                        /* indicate that diffusing CSH species remained */
                        /* at original location */
                        action=7;
                }
        }
        return(action);
}

/* routine to add extra FH3 when gypsum, hemihydrate, anhydrite, CAS2, or */
/* CaCl2 reacts with C4AF at location (xpres,ypres,zpres) */
/* Called by movegyp, moveettr, movecas2, movehem, moveanh, and movecacl2 */
/* Calls moveone and edgecnt */
void extfh3(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
/*    a) successful                            */
/*    b) all 6 sites are tried and full or     */
/*    c) 500 tries are made                    */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);

                if(newact==0){printf("Error in value of newact in extfh3 \n");}
                check=mic[xchr][ychr][zchr];
```

```
                /* if neighbor is porosity   */
                /* then locate the FH3 there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=FH3;
                        count[FH3]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate FH3 at random location */
/* in pore space in contact with at least one FH3 */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the FH3 there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,FH3,FH3,DIFFFH3);
                        /* be sure that at least one neighboring pixel */
                        /* is FH3 or diffusing FH3 */
                        if((numnear<26)||(tries>5000)){
                           mic[xchr][ychr][zchr]=FH3;
                           count[FH3]+=1;
                           count[POROSITY]-=1;
                           fchr=1;
                        }
                }
        }
}

/* routine to add extra ettringite when gypsum, anhydrite, or hemihydrate */
/* reacts with aluminates addition adjacent to location (xpres,ypres,zpres) */
/* in a fashion to preserve needle growth */
/* etype=0 indicates primary ettringite */
/* etype=1 indicates iron-rich stable ettringite */
/* Returns flag indicating action taken */
/* Calls moveone and edgecnt */
/* Called by movegyp, movehem, moveanh, and movec3a */
int extettr(xpres,ypres,zpres,etype)
        int xpres,ypres,zpres,etype;
{
        int check,newact,multf,numnear,sump,xchr,ychr,zchr,fchr,i1,plok;
        int numalum,numsil;
        float pneigh,ptest;
        long int tries;
```

```
/* first try neighboring locations until        */
/*    a) successful                             */
/*    c) 1000 tries are made                    */
       fchr=0;
       sump=1;
      /* Note that 30030 = 2*3*5*7*11*13 */
      /* indicating that all six sites have been tried */
      for(i1=1;((i1<=1000)&&(fchr==0));i1++){

/* determine location of neighbor (using periodic boundaries) */
               xchr=xpres;
               ychr=ypres;
               zchr=zpres;
               newact=0;
               multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
               if(newact==0){printf("Error in value of action \n");}

               check=mic[xchr][ychr][zchr];

               /* if neighbor is porosity, and conditions are favorable */
               /* based on number of neighboring ettringite, C3A, or C4AF */
               /* pixels then locate the ettringite there */
               if(check==POROSITY){
                       /* be sure ettringite doesn't touch C3S */
                          numsil=edgecnt(xchr,ychr,zchr,C3S,C2S,C3S);
                          numsil=26-numsil;
                 if(etype==0){
                             numnear=edgecnt(xchr,ychr,zchr,ETTR,ETTR,ETTR);
                             numalum=edgecnt(xchr,ychr,zchr,C3A,C3A,C3A);
                             numalum=26-numalum;
                 }
                 else{

                 numnear=edgecnt(xchr,ychr,zchr,ETTRC4AF,ETTRC4AF,ETTRC4AF);
                       numalum=edgecnt(xchr,ychr,zchr,C4AF,C4AF,C4AF);
                       numalum=26-numalum;
                 }
                 pneigh=(float)(numnear+1)/26.0;
                 pneigh*=pneigh;
                 if(numalum<=1){pneigh=0.0;}
                 if(numalum>=2){pneigh+=0.5;}
                 if(numalum>=3){pneigh+=0.25;}
                 if(numalum>=5){pneigh+=0.25;}
                       ptest=ran1(seed);
                       if(numsil<1){
                       if(pneigh>=ptest){
                       if(etype==0){
                             mic[xchr][ychr][zchr]=ETTR;
                             count[ETTR]+=1;
                       }
                       else{
                             mic[xchr][ychr][zchr]=ETTRC4AF;
                             count[ETTRC4AF]+=1;
                       }
                       fchr=1;
                       count[POROSITY]-=1;
```

```
                        }
                        }
                }
        }

/* if no neighbor available, locate ettringite at random location */
/* in pore space in contact with at least another ettringite */
/* or aluminate surface  */
        tries=0;
        while(fchr==0){
                tries+=1;
                newact=7;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}

                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the ettringite there */
                if(check==POROSITY){
                          numsil=edgecnt(xchr,ychr,zchr,C3S,C2S,C3S);
                          numsil=26-numsil;
                  if(etype==0){
                          numnear=edgecnt(xchr,ychr,zchr,ETTR,C3A,C4AF);
                  }
                  else{
                          numnear=edgecnt(xchr,ychr,zchr,ETTRC4AF,C3A,C4AF);
                  }
                        /* be sure that at least one neighboring pixel */
                        /* is ettringite, or aluminate clinker */
                        if((tries>5000)||((numnear<26)&&(numsil<1))){
                        if(etype==0){
                                    mic[xchr][ychr][zchr]=ETTR;
                                    count[ETTR]+=1;
                        }
                        else{
                                    mic[xchr][ychr][zchr]=ETTRC4AF;
                                    count[ETTRC4AF]+=1;
                        }
                        count[POROSITY]-=1;
                        fchr=1;
                        }
                }
        }
        return(newact);
}

/* routine to add extra CH when gypsum, hemihydrate, anhydrite, CaCl2, or */
/* diffusing CAS2  reacts with C4AF */
/* Called by movegyp, movehem, moveanh, moveettr, movecas2, and movecacl2 */
/* Calls edgecnt */
void extch()
{
        int numnear,sump,xchr,ychr,zchr,fchr,i1,plok,check;
```

```
        long int tries;

        fchr=0;
        tries=0;
        /* locate CH at random location */
        /* in pore space in contact with at least another CH */
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                /* if location is porosity, locate the CH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,CH,DIFFCH,CH);
                        /* be sure that at least one neighboring pixel */
                        /* is CH or diffusing CH */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=CH;
                                count[CH]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}

/* routine to add extra gypsum when hemihydrate or anhydrite hydrates */
/* Called by movehem and moveanh */
/* Calls moveone and edgecnt */
void extgyps(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
/*    a) successful                            */
/*    b) all 6 sites are tried and full or     */
/*    c) 500 tries are made                    */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extfh3 \n");}
                check=mic[xchr][ychr][zchr];
```

```
                /* if neighbor is porosity   */
                /* then locate the GYPSUMS there */
                if(check==POROSITY){
                  mic[xchr][ychr][zchr]=GYPSUMS;
                  count[GYPSUMS]+=1;
                  count[POROSITY]-=1;
                  fchr=1;
                }
                else{
                  sump*=multf;
                }
        }

/* if no neighbor available, locate GYPSUMS at random location */
/* in pore space in contact with at least one GYPSUMS */
        tries=0;
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the GYPSUMS there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,HEMIHYD,GYPSUMS,ANHYDRITE);
                        /* be sure that at least one neighboring pixel */
                        /* is Gypsum in some form */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=GYPSUMS;
                                count[GYPSUMS]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}

/* routine to move a diffusing ANHYDRITE species */
/* Inputs: current location (xcur,ycur,zcur) and flag indicating if final */
/* step in diffusion process */
/* Returns flag indicating action taken (reaction or diffusion/no movement) */
/* Calls moveone */
/* Called by hydrate */
int moveanh(xcur,ycur,zcur,finalstep,nucprgyp)
        int xcur,ycur,zcur,finalstep;
        float nucprgyp;
{
        int xnew,ynew,znew,plok,action,sumback,sumin,check;
        int nexp,iexp,xexp,yexp,zexp,newact,sumold,sumgarb,ettrtype;
        float pgen,pexp,pext,p2diff;

        action=0;
```

```
        /* first check for nucleation */
        pgen=ran1(seed);
        p2diff=ran1(seed);
        if((nucprgyp>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=GYPSUMS;
                count[DIFFANH]-=1;
                count[GYPSUMS]+=1;
                pexp=ran1(seed);
            if(pexp<0.4){
                  extgyps(xcur,ycur,zcur);
            }
        }
        else{
              /* Store current location of species */
                   xnew=xcur;
                   ynew=ycur;
                   znew=zcur;
                   sumin=1;
                   sumback=moveone(&xnew,&ynew,&znew,&action,sumin);

                   if(action==0){printf("Error in value of action \n");}
                   check=mic[xnew][ynew][znew];

/* if new location is solid GYPSUM(S) or diffusing GYPSUM, then convert */
/* diffusing ANHYDRITE species to solid GYPSUM */
            if((check==GYPSUM)||(check==GYPSUMS)||(check==DIFFGYP)){
                      mic[xcur][ycur][zcur]=GYPSUMS;
                      /* decrement count of diffusing ANHYDRITE species */
                      /* and increment count of solid GYPSUMS */
                      count[DIFFANH]-=1;
                      count[GYPSUMS]+=1;
                      action=0;
                      /* Add extra gypsum as necessary */
                      pexp=ran1(seed);
                      if(pexp<0.4){
                         extgyps(xnew,ynew,znew);
                      }
                   }
        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to ettringite (including necessary volumetric expansion) */
        else
if(((check==C3A)&&(p2diff<SOLIDC3AGYP))||((check==DIFFC3A)&&(p2diff<C3AGYP))||((che
ck==DIFFC4A)&&(p2diff<C3AGYP))){
        /* Convert diffusing gypsum to an ettringite pixel */
            ettrtype=0;
            mic[xcur][ycur][zcur]=ETTR;
            if(check==DIFFC4A){
                  ettrtype=1;
                  mic[xcur][ycur][zcur]=ETTRC4AF;
            }
            action=0;
            count[DIFFANH]-=1;
            count[check]-=1;

                /* determine if C3A should be converted to ettringite */
                /* 1 unit of hemihydrate requires 0.569 units of C3A */
```

```
                /* and should form 4.6935 units of ettringite */
                pexp=ran1(seed);
                nexp=3;
                if(pexp<=0.569){
                if(ettrtype==0){
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                  }
                  else{
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                  }
                  nexp=2;
                }
                else{
                        /* maybe someday, use a new FIXEDC3A here */
                        /* so it won't dissolve later */
                        if(check==C3A){
                                mic[xnew][ynew][znew]=C3A;
                                count[C3A]+=1;
                        }
                        else{
                        if(ettrtype==0){
                                count[DIFFC3A]+=1;
                                mic[xnew][ynew][znew]=DIFFC3A;
                        }
                        else{
                                count[DIFFC4A]+=1;
                                mic[xnew][ynew][znew]=DIFFC4A;
                        }
                        }
                        nexp=3;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
```

```
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last ettringite pixel */
        pexp=ran1(seed);
        if(pexp<=0.6935){
                newact=extettr(xexp,yexp,zexp,ettrtype);
        }
}

/* if new location is C4AF execute conversion */
/* to ettringite (including necessary volumetric expansion) */
if((check==C4AF)&&(p2diff<SOLIDC4AFGYP)){
        mic[xcur][ycur][zcur]=ETTRC4AF;
        count[ETTRC4AF]+=1;
        count[DIFFANH]-=1;

        /* determine if C4AF should be converted to ettringite */
        /* 1 unit of gypsum requires 0.8174 units of C4AF */
        /* and should form 4.6935 units of ettringite */
        pexp=ran1(seed);
        nexp=3;
        if(pexp<=0.8174){
          mic[xnew][ynew][znew]=ETTRC4AF;
          count[ETTRC4AF]+=1;
          count[C4AF]-=1;
          nexp=2;
          pext=ran1(seed);
          /* Addition of extra CH */
          if(pext<0.2584){
                        extch();
          }
          pext=ran1(seed);
          /* Addition of extra FH3 */
          if(pext<0.5453){
                        extfh3(xnew,ynew,znew);
          }

        }
        else{
                /* maybe someday, use a new FIXEDC4AF here */
                /* so it won't dissolve later */
                mic[xnew][ynew][znew]=C4AF;
                nexp=3;
```

```
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6935){
                        newact=extettr(xexp,yexp,zexp,1);
                }
                action=0;
        }
      }

        if(action!=0){
        /* if diffusion step is possible, perform it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFANH;
                }
                else{
```

```
                        /* indicate that diffusing ANHYDRITE species remained */
                        /* at original location */
                        action=7;
                }
        }
        return(action);
}

/* routine to move a diffusing HEMIHYDRATE species */
/* Inputs: current location (xcur,ycur,zcur) and flag indicating if final */
/* step in diffusion process */
/* Returns flag indicating action taken (reaction or diffusion/no movement) */
/* Calls moveone, extettr, extch, and extfh3 */
/* Called by hydrate */
int movehem(xcur,ycur,zcur,finalstep,nucprgyp)
        int xcur,ycur,zcur,finalstep;
        float nucprgyp;
{
        int xnew,ynew,znew,plok,action,sumback,sumin,check;
        int nexp,iexp,xexp,yexp,zexp,newact,sumold,sumgarb,ettrtype;
        float pgen,pexp,pext,p2diff;

        action=0;
        /* first check for nucleation */
        pgen=ran1(seed);
        p2diff=ran1(seed);
        if((nucprgyp>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=GYPSUMS;
                count[DIFFHEM]-=1;
                count[GYPSUMS]+=1;
                /* Add extra gypsum as necessary */
                pexp=ran1(seed);
                if(pexp<0.4){
                  extgyps(xcur,ycur,zcur);
                }
        }
        else{
        /* Store current location of species */
                   xnew=xcur;
                   ynew=ycur;
                   znew=zcur;
                   sumin=1;
                   sumback=moveone(&xnew,&ynew,&znew,&action,sumin);

                   if(action==0){printf("Error in value of action \n");}
                   check=mic[xnew][ynew][znew];

/* if new location is solid GYPSUM(S) or diffusing GYPSUM, then convert */
/* diffusing HEMIHYDRATE species to solid GYPSUM */
            if((check==GYPSUM)||(check==GYPSUMS)||(check==DIFFGYP)){
                      mic[xcur][ycur][zcur]=GYPSUMS;
                      /* decrement count of diffusing HEMIHYDRATE species */
                      /* and increment count of solid GYPSUMS */
                      count[DIFFHEM]-=1;
                      count[GYPSUMS]+=1;
                      action=0;
```

```
                    /* Add extra gypsum as necessary */
                    pexp=ran1(seed);
                    if(pexp<0.4){
                      extgyps(xnew,ynew,znew);
                    }
             }
        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to ettringite (including necessary volumetric expansion) */
        else
if(((check==C3A)&&(p2diff<SOLIDC3AGYP))||((check==DIFFC3A)&&(p2diff<C3AGYP))||((che
ck==DIFFC4A)&&(p2diff<C3AGYP))){
        /* Convert diffusing gypsum to an ettringite pixel */
            ettrtype=0;
            mic[xcur][ycur][zcur]=ETTR;
            if(check==DIFFC4A){
                  ettrtype=1;
                  mic[xcur][ycur][zcur]=ETTRC4AF;
            }
                action=0;
                count[DIFFHEM]-=1;
                count[check]-=1;

                /* determine if C3A should be converted to ettringite */
                /* 1 unit of hemihydrate requires 0.5583 units of C3A */
                /* and should form 4.6053 units of ettringite */
                pexp=ran1(seed);
                nexp=3;
                if(pexp<=0.5583){
                  if(ettrtype==0){
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                  }
                  else{
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                  }
                  nexp=2;
                }
                else{
                        /* maybe someday, use a new FIXEDC3A here */
                        /* so it won't dissolve later */
                        if(check==C3A){
                                mic[xnew][ynew][znew]=C3A;
                                count[C3A]+=1;
                        }
                        else{
                        if(ettrtype==0){
                                 count[DIFFC3A]+=1;
                                 mic[xnew][ynew][znew]=DIFFC3A;
                        }
                        else{
                                 count[DIFFC4A]+=1;
                                 mic[xnew][ynew][znew]=DIFFC4A;
                        }
                        }
                        nexp=3;
                }
```

```
/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6053){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                }
        }

        /* if new location is C4AF execute conversion */
        /* to ettringite (including necessary volumetric expansion) */
        if((check==C4AF)&&(p2diff<SOLIDC4AFGYP)){
                mic[xcur][ycur][zcur]=ETTRC4AF;
                count[ETTRC4AF]+=1;
                count[DIFFHEM]-=1;

                /* determine if C4AF should be converted to ettringite */
                /* 1 unit of gypsum requires 0.802 units of C4AF */
                /* and should form 4.6053 units of ettringite */
```

```
                pexp=ran1(seed);
                nexp=3;
                if(pexp<=0.802){
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        count[C4AF]-=1;
                        nexp=2;
                        pext=ran1(seed);
                        /* Addition of extra CH */
                        if(pext<0.2584){
                              extch();
                        }
                        pext=ran1(seed);
                        /* Addition of extra FH3 */
                        if(pext<0.5453){
                              extfh3(xnew,ynew,znew);
                        }

                }
                else{
                        /* maybe someday, use a new FIXEDC4AF here */
                        /* so it won't dissolve later */
                        mic[xnew][ynew][znew]=C4AF;
                        nexp=3;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
```

```
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.6053){
                        newact=extettr(xexp,yexp,zexp,1);
                }
                action=0;
        }
      }

        if(action!=0){
        /* if diffusion step is possible, perform it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFHEM;
                }
                else{
                        /* indicate that diffusing HEMIHYDRATE species */
                        /* remained at original location */
                        action=7;
                }
        }
        return(action);
}

/* routine to add extra Freidel's salt when CaCl2 reacts with */
/* C3A or C4AF at location (xpres,ypres,zpres) */
/* Called by movecacl2 and movec3a */
/* Calls moveone and edgecnt */
int extfreidel(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
/*    a) successful                            */
/*    b) all 6 sites are tried and full or     */
/*    c) 500 tries are made                    */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extfreidel \n");}
```

```
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity   */
                /* then locate the freidel's salt there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=FREIDEL;
                        count[FREIDEL]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate FREIDEL at random location */
/* in pore space in contact with at least one FREIDEL */
        tries=0;
        while(fchr==0){
                tries+=1;
                newact=7;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the FREIDEL there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,FREIDEL,FREIDEL,DIFFCACL2);
                        /* be sure that at least one neighboring pixel */
                        /* is FREIDEL or diffusing CACL2 */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=FREIDEL;
                                count[FREIDEL]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
      return(newact);
}

/* routine to add extra stratlingite when AS reacts with */
/* CH at location (xpres,ypres,zpres) */
/* or when diffusing CAS2 reacts with aluminates */
/* Called by moveas, movech, and movecas2 */
/* Calls moveone and edgecnt */
int extstrat(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int multf,numnear,sump,xchr,ychr,zchr,check,fchr,i1,plok,newact;
        long int tries;

/* first try 6 neighboring locations until      */
```

```
/*    a) successful                              */
/*    b) all 6 sites are tried and full or       */
/*    c) 500 tries are made                      */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=500)&&(fchr==0)&&(sump!=30030));i1++){

                /* choose a neighbor at random */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                multf=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extstrat \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity   */
                /* then locate the stratlingite there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=STRAT;
                        count[STRAT]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
                else{
                        sump*=multf;
                }
        }

/* if no neighbor available, locate STRAT at random location */
/* in pore space in contact with at least one STRAT */
        tries=0;
        while(fchr==0){
                tries+=1;
                newact=7;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
                /* if location is porosity, locate the STRAT there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,STRAT,DIFFCAS2,DIFFAS);
                        /* be sure that at least one neighboring pixel */
                        /* is STRAT, diffusing CAS2, or diffusing AS */
                        if((numnear<26)||(tries>5000)){
                                mic[xchr][ychr][zchr]=STRAT;
                                count[STRAT]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
      return(newact);
```

```
}

/* routine to move a diffusing gypsum species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extettr, extch, and extfh3 */
int movegyp(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,ettrtype;
        float pexp,pext,p2diff;

        sumold=1;

/* First be sure that a diffusing gypsum species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFGYP){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movegyp \n");}
        check=mic[xnew][ynew][znew];
        p2diff=ran1(seed);
        /* if new location is CSH, check for absorption of gypsum */
        if((check==CSH)&&((float)count[ABSGYP]<(gypabsprob*(float)count[CSH]))){
                pexp=ran1(seed);
                if(pexp<AGRATE){
                        /* update counts for absorbed and diffusing gypsum */
                        count[ABSGYP]+=1;
                        count[DIFFGYP]-=1;
                        mic[xcur][ycur][zcur]=ABSGYP;
                        action=0;
                }
        }

        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to ettringite (including necessary volumetric expansion) */
      /* Use p2diff to try to favor formation of ettringite on */
      /* aluminate surfaces as opposed to in solution */
        else
if(((check==C3A)&&(p2diff<SOLIDC3AGYP))||((check==DIFFC3A)&&(p2diff<C3AGYP))||((che
ck==DIFFC4A)&&(p2diff<C3AGYP))){
        /* Convert diffusing gypsum to an ettringite pixel */
            ettrtype=0;
            mic[xcur][ycur][zcur]=ETTR;
            if(check==DIFFC4A){
                  ettrtype=1;
                  mic[xcur][ycur][zcur]=ETTRC4AF;
```

```
            }
            action=0;
            count[DIFFGYP]-=1;
            count[check]-=1;

                /* determine if C3A should be converted to ettringite */
                /* 1 unit of gypsum requires 0.40 units of C3A */
                /* and should form 3.30 units of ettringite */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.40){
                  if(ettrtype==0){
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                  }
                  else{
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                  }
                        nexp=1;
                }
                else{
                        /* maybe someday, use a new FIXEDC3A here */
                        /* so it won't dissolve later */
                        if(check==C3A){
                                mic[xnew][ynew][znew]=C3A;
                                count[C3A]+=1;
                        }
                        else{
                        if(ettrtype==0){
                                     count[DIFFC3A]+=1;
                                     mic[xnew][ynew][znew]=DIFFC3A;
                        }
                        else{
                                     count[DIFFC4A]+=1;
                                     mic[xnew][ynew][znew]=DIFFC4A;
                        }
                        }
                        nexp=2;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,ettrtype);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
```

```
                                if(xexp>=SYSIZE){xexp=0;}
                                break;
                        case 3:
                                yexp-=1;
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last ettringite pixel */
        pexp=ran1(seed);
        if(pexp<=0.30){
                newact=extettr(xexp,yexp,zexp,ettrtype);
        }
}

/* if new location is C4AF execute conversion */
/* to ettringite (including necessary volumetric expansion) */
if((check==C4AF)&&(p2diff<SOLIDC4AFGYP)){
        mic[xcur][ycur][zcur]=ETTRC4AF;
        count[ETTRC4AF]+=1;
        count[DIFFGYP]-=1;

        /* determine if C4AF should be converted to ettringite */
        /* 1 unit of gypsum requires 0.575 units of C4AF */
        /* and should form 3.30 units of ettringite */
        pexp=ran1(seed);
        nexp=2;
        if(pexp<=0.575){
                mic[xnew][ynew][znew]=ETTRC4AF;
                count[ETTRC4AF]+=1;
                count[C4AF]-=1;
                nexp=1;
                pext=ran1(seed);
                /* Addition of extra CH */
                if(pext<0.2584){
                      extch();
                }
                pext=ran1(seed);
                /* Addition of extra FH3 */
                if(pext<0.5453){
                      extfh3(xnew,ynew,znew);
                }
```

```
                }
                else{
                        /* maybe someday, use a new FIXEDC4AF here */
                        /* so it won't dissolve later */
                        mic[xnew][ynew][znew]=C4AF;
                        nexp=2;
                }

/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added ettringite */
/* species as we attempt to grow a needle like structure */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,1);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                               }
                }

                /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.30){
                        newact=extettr(xexp,yexp,zexp,1);
                }
                action=0;
        }

        /* if last diffusion step and no reaction, convert back to */
```

```
        /* primary solid gypsum */
        if((action!=0)&&(finalstep==1)){
                action=0;
                count[DIFFGYP]-=1;
                count[GYPSUM]+=1;
                mic[xcur][ycur][zcur]=GYPSUM;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFGYP;
                }
                else{
                        /* indicate that diffusing gypsum remained at */
                        /* original location */
                        action=7;
                }
        }
        return(action);

}

/* routine to move a diffusing CaCl2 species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extfreidel, extch, and extfh3 */
int movecacl2(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;

/* First be sure that a diffusing CaCl2 species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFCACL2){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movecacl2 \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to freidel's salt (including necessary volumetric expansion) */
```

```
        if((check==C3A)||(check==DIFFC3A)||(check==DIFFC4A)){
        /* Convert diffusing C3A or C3A to a freidel's salt pixel */
                action=0;
                mic[xnew][ynew][znew]=FREIDEL;
                count[FREIDEL]+=1;
                count[check]-=1;

                /* determine if diffusing CaCl2 should be converted to FREIDEL */
                /* 0.5793 unit of CaCl2 requires 1 unit of C3A */
                /* and should form 3.3295 units of FREIDEL */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.5793){
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        count[DIFFCACL2]-=1;
                        nexp=1;
                }
                else{
                        keep=1;
                        nexp=2;
                }

/* create extra Freidel's salt pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added FREIDEL */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extfreidel(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
```

```
                                        break;
                        }
                }

                /* probabilistic-based expansion for last FREIDEL pixel */
                pexp=ran1(seed);
                if(pexp<=0.3295){
                        newact=extfreidel(xexp,yexp,zexp);
                }
        }

        /* if new location is C4AF execute conversion */
        /* to freidel's salt (including necessary volumetric expansion) */
        else if(check==C4AF){
                mic[xnew][ynew][znew]=FREIDEL;
                count[FREIDEL]+=1;
                count[C4AF]-=1;

                /* determine if CACL2 should be converted to FREIDEL */
                /* 0.4033 unit of CaCl2 requires 1 unit of C4AF */
                /* and should form 2.3176 units of FREIDEL */
                /* Also 0.6412 units of CH and 1.3522 units of FH3 */
                /* per unit of CACL2 */
                pexp=ran1(seed);
                nexp=1;
                if(pexp<=0.4033){
                  mic[xcur][ycur][zcur]=FREIDEL;
                  count[FREIDEL]+=1;
                  count[DIFFCACL2]-=1;
                  nexp=0;
                  pext=ran1(seed);
                  /* Addition of extra CH */
                  if(pext<0.6412){
                              extch();
                  }
                  pext=ran1(seed);
                  /* Addition of extra FH3 */
                  if(pext<0.3522){
                              extfh3(xnew,ynew,znew);
                  }
                  extfh3(xnew,ynew,znew);

                }
                else{
                  nexp=1;
                  keep=1;
                }

/* create extra freidel's salt pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added FREIDEL */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extfreidel(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
```

```
                        case 1:
                                xexp-=1;
                                if(xexp<0){xexp=(SYSIZEM1);}
                                break;
                        case 2:
                                xexp+=1;
                                if(xexp>=SYSIZE){xexp=0;}
                                break;
                        case 3:
                                yexp-=1;
                                if(yexp<0){yexp=(SYSIZEM1);}
                                break;
                        case 4:
                                yexp+=1;
                                if(yexp>=SYSIZE){yexp=0;}
                                break;
                        case 5:
                                zexp-=1;
                                if(zexp<0){zexp=(SYSIZEM1);}
                                break;
                        case 6:
                                zexp+=1;
                                if(zexp>=SYSIZE){zexp=0;}
                                break;
                        default:
                                break;
                }
        }

        /* probabilistic-based expansion for last FREIDEL pixel */
        pexp=ran1(seed);
        if(pexp<=0.3176){
                newact=extfreidel(xexp,yexp,zexp);
        }
        action=0;
}

/* if last diffusion step and no reaction, convert back to */
/* solid CaCl2 */
if((action!=0)&&(finalstep==1)){
        action=0;
        count[DIFFCACL2]-=1;
        count[CACL2]+=1;
        mic[xcur][ycur][zcur]=CACL2;
}

if(action!=0){
        /* if diffusion is possible, execute it */
        if(check==POROSITY){
                mic[xcur][ycur][zcur]=POROSITY;
                mic[xnew][ynew][znew]=DIFFCACL2;
        }
        else{
                /* indicate that diffusing CACL2 remained at */

                /* original location */
                action=7;
```

```
                }
        }
      if(keep==1){action=7;}
      return(action);
}

/* routine to move a diffusing CAS2 species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extstrat, extch, and extfh3 */
int movecas2(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;

/* First be sure that a diffusing CAS2 species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFCAS2){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movecas2 \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to stratlingite (including necessary volumetric expansion) */
        if((check==C3A)||(check==DIFFC3A)||(check==DIFFC4A)){
        /* Convert diffusing CAS2 to a stratlingite pixel */
                action=0;
                mic[xcur][ycur][zcur]=STRAT;
                count[STRAT]+=1;
                count[DIFFCAS2]-=1;

                /* determine if diffusing or solid C3A should be converted to
STRAT*/
                /* 1 unit of CAS2 requires 0.886 units of C3A */
                /* and should form 4.286 units of STRAT */
                pexp=ran1(seed);
                nexp=3;
                if(pexp<=0.886){
                        mic[xnew][ynew][znew]=STRAT;
                        count[STRAT]+=1;
                        count[check]-=1;
                        nexp=2;
```

```
                }

/* create extra stratlingite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added STRAT */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last STRAT pixel */
                pexp=ran1(seed);
                if(pexp<=0.286){
                        newact=extstrat(xexp,yexp,zexp);
                }
        }

        /* if new location is C4AF execute conversion */
        /* to stratlingite (including necessary volumetric expansion) */
        else if(check==C4AF){
                mic[xnew][ynew][znew]=STRAT;
                count[STRAT]+=1;
                count[C4AF]-=1;

                /* determine if CAS2 should be converted to STRAT */
                /* 0.786 units of CAS2 requires 1 unit of C4AF */
                /* and should form 3.37 units of STRAT */
```

```
                /* Also 0.2586 units of CH and 0.5453 units of FH3 */
                /* per unit of C4AF */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.786){
                        mic[xcur][ycur][zcur]=STRAT;
                        count[STRAT]+=1;
                        count[DIFFCAS2]-=1;
                        nexp=1;
                        pext=ran1(seed);
                        /* Addition of extra CH */
                        /* 0.329= 0.2586/0.786 */
                        if(pext<0.329){
                              extch();
                        }
                        pext=ran1(seed);
                        /* Addition of extra FH3 */
                        /* 0.6938= 0.5453/0.786 */
                        if(pext<0.6938){
                               extfh3(xnew,ynew,znew);
                        }

                }
                else{
                        nexp=2;
                        keep=1;
                }

/* create extra stratlingite pixels to maintain volume stoichiometry */
/* xexp, yexp and zexp hold coordinates of most recently added STRAT */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
```

```
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last STRAT pixel */
                pexp=ran1(seed);
                if(pexp<=0.37){
                        newact=extstrat(xexp,yexp,zexp);
                }
                action=0;
        }

        /* if last diffusion step and no reaction, convert back to */
        /* solid CAS2 */
        if((action!=0)&&(finalstep==1)){
                action=0;
                count[DIFFCAS2]-=1;
                count[CAS2]+=1;
                mic[xcur][ycur][zcur]=CAS2;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFCAS2;
                }
                else{
                        /* indicate that diffusing CAS2 remained at */
                        /* original location */
                        action=7;
                }
        }
      if(keep==1){action=7;}
      return(action);
}

/* routine to move a diffusing AS species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
/* Called by hydrate */
/* Calls moveone, extstrat */
int moveas(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;
```

```
/* First be sure that a diffusing AS species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFAS){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in moveas \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is CH or diffusing CH, execute conversion */
        /* to stratlingite (including necessary volumetric expansion) */
        if((check==CH)||(check==DIFFCH)){
        /* Convert diffusing CH or CH to a stratlingite pixel */
                action=0;
                mic[xnew][ynew][znew]=STRAT;
                count[STRAT]+=1;
                count[check]-=1;

                /* determine if diffusing AS should be converted to STRAT */
                /* 0.7538 unit of AS requires 1 unit of CH */
                /* and should form 3.26 units of STRAT */
                pexp=ran1(seed);
                nexp=2;
                if(pexp<=0.7538){
                        mic[xcur][ycur][zcur]=STRAT;
                        count[STRAT]+=1;
                        count[DIFFAS]-=1;
                        nexp=1;
                }
                else{
                        keep=1;
                        nexp=2;
                }

/* create extra stratlingite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added STRAT */
                xexp=xcur;
                yexp=ycur;
                zexp=zcur;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp as needed */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
```

```
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                        }
                }

                /* probabilistic-based expansion for last stratlingite pixel */
                pexp=ran1(seed);
                if(pexp<=0.326){
                        newact=extstrat(xexp,yexp,zexp);
                }
        }

        /* if last diffusion step and no reaction, convert back to */
        /* solid ASG */
        if((action!=0)&&(finalstep==1)){
                action=0;
                count[DIFFAS]-=1;
                count[ASG]+=1;
                mic[xcur][ycur][zcur]=ASG;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFAS;
                }
                else{
                        /* indicate that diffusing AS remained at */
                        /* original location */
                        action=7;
                }
        }
      if(keep==1){action=7;}
      return(action);
}

/* routine to move a diffusing CACO3 species */
/* from current location (xcur,ycur,zcur) */
/* Returns action flag indicating response taken */
```

```
/* Called by hydrate */
/* Calls moveone, extettr */
int movecaco3(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb,keep;
        float pexp,pext;

        sumold=1;
        keep=0;

/* First be sure that a diffusing CACO3 species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFCACO3){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in moveas \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is AFM execute conversion */
        /* to carboaluminate and ettringite (including necessary */
        /* volumetric expansion) */
        if(check==AFM){
        /* Convert AFM to a carboaluminate or ettringite pixel */
                action=0;
                pexp=ran1(seed);
                if(pexp<=0.479192){
                      mic[xnew][ynew][znew]=AFMC;
                      count[AFMC]+=1;
                }
                else{
                      mic[xnew][ynew][znew]=ETTR;
                      count[ETTR]+=1;
                }
                count[check]-=1;

                /* determine if diffusing CACO3 should be converted to AFMC */
                /* 0.078658 unit of AS requires 1 unit of AFM */
                /* and should form 0.55785 units of AFMC */
                pexp=ran1(seed);
                if(pexp<=0.078658){
                        mic[xcur][ycur][zcur]=AFMC;
                        count[AFMC]+=1;
                        count[DIFFCACO3]-=1;
                }
                else{
                        keep=1;
                }
```

```
/* create extra ettringite pixels to maintain volume stoichiometry */
/* xexp, yexp, and zexp hold coordinates of most recently added ETTR */
                xexp=xnew;
                yexp=ynew;
                zexp=znew;

                /* probabilistic-based expansion for new ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.26194){
                        newact=extettr(xexp,yexp,zexp,0);
                }
        }

        /* if last diffusion step and no reaction, convert back to */
        /* solid CACO3 */
        if((action!=0)&&(finalstep==1)){
                action=0;
                count[DIFFCACO3]-=1;
                count[CACO3]+=1;
                mic[xcur][ycur][zcur]=CACO3;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFCACO3;
                }
                else{
                        /* indicate that diffusing CACO3 remained at */
                        /* original location */
                        action=7;
                }
        }
      if(keep==1){action=7;}
      return(action);
}

/* routine to add extra AFm phase when diffusing ettringite reacts */
/* with C3A (diffusing or solid) at location (xpres,ypres,zpres) */
/* Called by moveettr and movec3a */
/* Calls moveone and edgecnt */
void extafm(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,newact,numnear;
        long int tries;

/* first try 6 neighboring locations until      */
/*    a) successful                            */
/*    b) all 6 sites are tried or              */
/*    c) 100 tries are made                    */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){
```

```
                /* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                newact=0;
                sump*=moveone(&xchr,&ychr,&zchr,&newact,sump);
                if(newact==0){printf("Error in value of newact in extafm \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity, locate the AFm phase there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=AFM;
                        count[AFM]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
        }

        /* if no neighbor available, locate AFm phase at random location */
        /* in pore space */
        tries=0;
        while(fchr==0){
               tries+=1;
               /* generate a random location in the 3-D system */
               xchr=(int)((float)SYSIZE*ran1(seed));
               ychr=(int)((float)SYSIZE*ran1(seed));
               zchr=(int)((float)SYSIZE*ran1(seed));
               if(xchr>=SYSIZE){xchr=0;}
               if(ychr>=SYSIZE){ychr=0;}
               if(zchr>=SYSIZE){zchr=0;}
               check=mic[xchr][ychr][zchr];

               /* if location is porosity, locate the extra AFm there */
               if(check==POROSITY){
                       numnear=edgecnt(xchr,ychr,zchr,AFM,C3A,C4AF);
                       /* Be sure that at least one neighboring pixel is */
                       /* Afm phase, C3A, or C4AF */
                       if((tries>5000)||(numnear<26)){
                               mic[xchr][ychr][zchr]=AFM;
                               count[AFM]+=1;
                               count[POROSITY]-=1;
                               fchr=1;
                       }
               }
        }
}

/* routine to move a diffusing ettringite species */
/* currently located at (xcur,ycur,zcur) */
/* Called by hydrate */
/* Calls moveone, extch, extfh3, and extafm */
int moveettr(xcur,ycur,zcur,finalstep)
        int xcur,ycur,zcur,finalstep;
{
        int check,xnew,ynew,znew,plok,action,nexp,iexp;
        int xexp,yexp,zexp,newact,sumold,sumgarb;
        float pexp,pafm,pgrow;
```

```
/* First be sure a diffusing ettringite species is located at xcur,ycur,zcur */
/* if not, return to calling routine */
        if(mic[xcur][ycur][zcur]!=DIFFETTR){
                action=0;
                return(action);
        }

/* Determine new coordinates (periodic boundaries are used) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumold=1;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in moveettr \n");}
        check=mic[xnew][ynew][znew];

        /* if new location is C4AF, execute conversion */
        /* to AFM phase (including necessary volumetric expansion) */
        if(check==C4AF){
                /* Convert diffusing ettringite to AFM phase */
                mic[xcur][ycur][zcur]=AFM;
                count[AFM]+=1;
                count[DIFFETTR]-=1;

                /* determine if C4AF should be converted to Afm */
                /* or FH3- 1 unit of ettringite requires 0.348 units */
                /* of C4AF to form 1.278 units of Afm, */
                /* 0.0901 units of CH and 0.1899 units of FH3 */
                pexp=ran1(seed);

                if(pexp<=0.278){
                        mic[xnew][ynew][znew]=AFM;
                        count[AFM]+=1;
                        count[C4AF]-=1;
                        pafm=ran1(seed);
                        /* 0.3241= 0.0901/0.278 */
                        if(pafm<=0.3241){
                                extch();
                        }
                        pafm=ran1(seed);
                        /* 0.4313= ((.1899-(.348-.278))/.278)   */
                        if(pafm<=0.4313){
                                extfh3(xnew,ynew,znew);
                        }
                }
                else if (pexp<=0.348){
                        mic[xnew][ynew][znew]=FH3;
                        count[FH3]+=1;
                        count[C4AF]-=1;
                }
                action=0;
        }

        /* if new location is C3A or diffusing C3A, execute conversion */
        /* to AFM phase (including necessary volumetric expansion) */
```

```
        else if((check==C3A)||(check==DIFFC3A)){
                /* Convert diffusing ettringite to AFM phase */
                action=0;
                mic[xcur][ycur][zcur]=AFM;
                count[DIFFETTR]-=1;
                count[AFM]+=1;
                count[check]-=1;

                /* determine if C3A should be converted to AFm */
                /* 1 unit of ettringite requires 0.2424 units of C3A */
                /* and should form 1.278 units of AFm phase */

                pexp=ran1(seed);
                if(pexp<=0.2424){
                        mic[xnew][ynew][znew]=AFM;
                        count[AFM]+=1;
                        pafm=(-0.1);
                }
                else{
                        /* maybe someday, use a new FIXEDC3A here */
                        /* so it won't dissolve later */
                        if(check==C3A){
                                mic[xnew][ynew][znew]=C3A;
                                count[C3A]+=1;
                        }
                        else{
                                count[DIFFC3A]+=1;
                                mic[xnew][ynew][znew]=DIFFC3A;
                        }
/*                      pafm=(0.278-0.2424)/(1.0-0.2424);  */
                  pafm=0.04699;
                }

                /* probabilistic-based expansion for new AFm phase pixel */
                pexp=ran1(seed);
                if(pexp<=pafm){
                        extafm(xcur,ycur,zcur);
                }
        }

        /* Check for conversion back to solid ettringite */
        else if(check==ETTR){
                pgrow=ran1(seed);
                if(pgrow<=ETTRGROW){
                        mic[xcur] [ycur] [zcur]=ETTR;
                        count[ETTR]+=1;
                        action=0;
                        count[DIFFETTR]-=1;
                }
        }

        /* if last diffusion step and no reaction, convert back to */
        /* solid ettringite */
        if((action!=0)&&(finalstep==1)){
                action=0;
                count[DIFFETTR]-=1;
                count[ETTR]+=1;
```

```
                mic[xcur][ycur][zcur]=ETTR;
        }

        if(action!=0){
                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFETTR;
                }
                else{
                        /* indicate that diffusing ettringite remained at */
                        /* original location */
                        action=7;
                }
        }
        return(action);
}

/* routine to add extra pozzolanic CSH when CH reacts at */
/* pozzolanic surface (e.g. silica fume) located at (xpres,ypres,zpres) */
/* Called by movech */
/* Calls moveone and edgecnt */
void extpozz(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,action,numnear;
        long int tries;

/* first try 6 neighboring locations until      */
/*    a) successful                            */
/*    b) all 6 sites are tried or              */
/*    c) 100 tries are made                    */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine location of neighbor (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                action=0;
                sump*=moveone(&xchr,&ychr,&zchr,&action,sump);
                if(action==0){printf("Error in value of action in extpozz \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is porosity, locate the pozzolanic CSH there */
                if(check==POROSITY){
                        mic[xchr][ychr][zchr]=POZZCSH;
                        count[POZZCSH]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
        }

        /* if no neighbor available, locate pozzolanic CSH at random location */
        /* in pore space */
        tries=0;
```

```
        while(fchr==0){
                tries+=1;
                /* generate a random location in the 3-D system */
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];
           /* if location is porosity, locate the extra pozzolanic CSH there */
                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,POZZ,CSH,POZZCSH);
                        /* Be sure that one neighboring species is CSH or */
                        /* pozzolanic material */
                        if((tries>5000)||(numnear<26)){
                                mic[xchr][ychr][zchr]=POZZCSH;
                                count[POZZCSH]+=1;
                                count[POROSITY]-=1;
                                fchr=1;
                        }
                }
        }
}

/* routine to move a diffusing FH3 species */
/* from location (xcur,ycur,zcur) with nucleation probability nucprob */
/* Called by hydrate */
/* Calls moveone */
int movefh3(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumold,sumgarb;
        float pgen;

        /* first check for nucleation */
        pgen=ran1(seed);

        if((nucprob>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=FH3;
                count[FH3]+=1;
                count[DIFFFH3]-=1;
        }
        else{

                /* determine new location (using periodic boundaries) */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                action=0;
                sumold=1;
                sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
                if(action==0){printf("Error in value of action in movefh3 \n");}
                check=mic[xnew][ynew][znew];
```

```
                /* check for growth of FH3 crystal */
                if(check==FH3){
                        mic[xcur][ycur][zcur]=FH3;
                        count[FH3]+=1;
                        count[DIFFFH3]-=1;
                        action=0;
                }

                if(action!=0){
                        /* if diffusion is possible, execute it */
                        if(check==POROSITY){
                                mic[xcur][ycur][zcur]=POROSITY;
                                mic[xnew][ynew][znew]=DIFFFH3;
                        }
                        else{
                                /* indicate that diffusing FH3 species */
                                /* remained at original location */
                                action=7;
                        }
                }
        }
        return(action);
}

/* routine to move a diffusing CH species */
/* from location (xcur,ycur,zcur) with nucleation probability nucprob */
/* Called by hydrate */
/* Calls moveone and extpozz */
int movech(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumgarb,sumold;
        float pexp,pgen,pfix;

        /* first check for nucleation */
        pgen=ran1(seed);
        if((nucprob>=pgen)||(finalstep==1)){
                action=0;
                mic[xcur][ycur][zcur]=CH;
                count[DIFFCH]-=1;
                count[CH]+=1;
        }
        else{

                /* determine new location (using periodic boundaries) */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                action=0;
                sumold=1;
                sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
                if(action==0){printf("Error in value of action in movech \n");}
                check=mic[xnew][ynew][znew];

                /* check for growth of CH crystal */
                if((check==CH)&&(pgen<=CHGROW)){
```

```
                        mic[xcur][ycur][zcur]=CH;
                        count[DIFFCH]-=1;
                        count[CH]+=1;
                        action=0;
                }
              /* check for growth of CH crystal on aggregate or CaCO3 surface */
                /* re suggestion of Sidney Diamond */
                else
if(((check==INERTAGG)||(check==CACO3)||(check==INERT))&&(pgen<=CHGROWAGG)&&(chflag==1)){
                        mic[xcur][ycur][zcur]=CH;
                        count[DIFFCH]-=1;
                        count[CH]+=1;
                        action=0;
                }

                /* check for pozzolanic reaction */
            /* 36.41 units CH can react with 27 units of S */
                else
if((pgen<=ppozz)&&(check==POZZ)&&(npr<=(int)((float)nfill*1.35))){
                        action=0;
                        mic[xcur][ycur][zcur]=POZZCSH;
                        count[POZZCSH]+=1;
                        /* update counter of number of diffusing CH */
                        /* which have reacted pozzolanically */
                        npr+=1;
                        count[DIFFCH]-=1;
                        /* Convert pozzolan to pozzolanic CSH as needed */
                        pfix=ran1(seed);
                        if(pfix<=(1./1.35)){
                           mic[xnew][ynew][znew]=POZZCSH;
                           count[POZZ]-=1;
                           count[POZZCSH]+=1;
                        }
                        /* allow for extra pozzolanic CSH as needed */
                        pexp=ran1(seed);
                        /* should form 101.81 units of pozzolanic CSH for */
                        /* each 36.41 units of CH and 27 units of S */
                        /* 1.05466=(101.81-36.41-27)/36.41 */
                        extpozz(xcur,ycur,zcur);
                        if(pexp<=0.05466){
                                extpozz(xcur,ycur,zcur);
                        }
                }
            else if(check==DIFFAS){
                  action=0;
                  mic[xcur][ycur][zcur]=STRAT;
                  count[STRAT]+=1;
                  /* update counter of number of diffusing CH */
                  /* which have reacted to form stratlingite */
                  nasr+=1;
                  count[DIFFCH]-=1;
                  /* Convert DIFFAS to STRAT as needed */
                  pfix=ran1(seed);
                  if(pfix<=0.7538){
                        mic[xnew][ynew][znew]=STRAT;
                        count[STRAT]+=1;
```

```
                        count[DIFFAS]-=1;
                }
                /* allow for extra stratlingite as needed */
                /* 1.5035=(215.63-66.2-49.9)/66.2 */
                extstrat(xcur,ycur,zcur);

                pexp=ran1(seed);
                if(pexp<=0.5035){
                        extstrat(xcur,ycur,zcur);
                }
        }

        if(action!=0){
                        /* if diffusion is possible, execute it */
                        if(check==POROSITY){
                                mic[xcur][ycur][zcur]=POROSITY;
                                mic[xnew][ynew][znew]=DIFFCH;
                        }
                        else{
                                /* indicate that diffusing CH species */
                                /* remained at original location */
                                action=7;
                        }
                }
        }
        return(action);
}

/* routine to add extra C3AH6 when diffusing C3A nucleates or reacts at */
/* C3AH6 surface at location (xpres,ypres,zpres) */
/* Called by movec3a */
/* Calls moveone and edgecnt */
void extc3ah6(xpres,ypres,zpres)
        int xpres,ypres,zpres;
{
        int check,sump,xchr,ychr,zchr,fchr,i1,plok,action,numnear;
        long int tries;

/* First try 6 neighboring locations until      */
/*    a) successful                            */
/*    b) all 6 sites are tried or              */
/*    c) 100 random attempts are made          */
        fchr=0;
        sump=1;
        for(i1=1;((i1<=100)&&(fchr==0)&&(sump!=30030));i1++){

                /* determine new coordinates (using periodic boundaries) */
                xchr=xpres;
                ychr=ypres;
                zchr=zpres;
                action=0;
                sump*=moveone(&xchr,&ychr,&zchr,&action,sump);
                if(action==0){printf("Error in action value in extc3ah6 \n");}
                check=mic[xchr][ychr][zchr];

                /* if neighbor is pore space, convert it to C3AH6 */
                if(check==POROSITY){
```

```
                        mic[xchr][ychr][zchr]=C3AH6;
                        count[C3AH6]+=1;
                        count[POROSITY]-=1;
                        fchr=1;
                }
        }

        /* if unsuccessful, add C3AH6 at random location in pore space */
        tries=0;
        while(fchr==0){
                tries+=1;
                xchr=(int)((float)SYSIZE*ran1(seed));
                ychr=(int)((float)SYSIZE*ran1(seed));
                zchr=(int)((float)SYSIZE*ran1(seed));
                if(xchr>=SYSIZE){xchr=0;}
                if(ychr>=SYSIZE){ychr=0;}
                if(zchr>=SYSIZE){zchr=0;}
                check=mic[xchr][ychr][zchr];

                if(check==POROSITY){
                        numnear=edgecnt(xchr,ychr,zchr,C3AH6,C3A,C3AH6);
                        /* Be sure that new C3AH6 is in contact with */
                        /* at least one C3AH6 or C3A */
                        if((tries>5000)||(numnear<26)){
                              mic[xchr][ychr][zchr]=C3AH6;
                              count[C3AH6]+=1;
                              count[POROSITY]-=1;
                              fchr=1;
                        }
                }
        }
}

/* routine to move a diffusing C3A species */
/* from location (xcur,ycur,zcur) with nucleation probability of nucprob */
/* Called by hydrate */
/* Calls extc3ah6, moveone, extettr, and extafm */
int movec3a(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumgarb,sumold;
        int xexp,yexp,zexp,nexp,iexp,newact;
        float pgen,pexp,pafm,pgrow,p2diff;

        /* First be sure that a diffusing C3A species is at (xcur,ycur,zcur) */
        if(mic[xcur][ycur][zcur]!=DIFFC3A){
                action=0;

                return(action);
        }

        /* Check for nucleation into solid C3AH6 */
        pgen=ran1(seed);
        p2diff=ran1(seed);

        if((nucprob>=pgen)||(finalstep==1)){
```

```
                action=0;
                mic[xcur][ycur][zcur]=C3AH6;
                count[C3AH6]+=1;
                /* decrement count of diffusing C3A species */
                count[DIFFC3A]-=1;
                /* allow for probabilistic-based expansion of C3AH6 */
                /* crystal to account for volume stoichiometry */
                pexp=ran1(seed);
                if(pexp<=0.69){
                        extc3ah6(xcur,ycur,zcur);
                }
        }
        else{
                /* determine new coordinates (using periodic boundaries) */
                xnew=xcur;
                ynew=ycur;
                znew=zcur;
                action=0;
                sumold=1;
                sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
                if(action==0){printf("Error in value of action in movec3a \n");}
                check=mic[xnew][ynew][znew];

                /* check for growth of C3AH6 crystal */
                if(check==C3AH6){
                        pgrow=ran1(seed);
                        /* Try to slow down growth of C3AH6 crystals to */
                        /* promote ettringite and Afm formation */
                        if(pgrow<=C3AH6GROW){
                                mic[xcur][ycur][zcur]=C3AH6;
                                count[C3AH6]+=1;
                                count[DIFFC3A]-=1;
                                action=0;
                         /* allow for probabilistic-based expansion of C3AH6 */
                         /* crystal to account for volume stoichiometry */
                                pexp=ran1(seed);
                                if(pexp<=0.69){
                                        extc3ah6(xcur,ycur,zcur);
                                }
                        }
                }

                /* examine reaction with diffusing gypsum to form ettringite */
                /* Only allow reaction with diffusing gypsum */
                else if((check==DIFFGYP)&&(p2diff<C3AGYP)){
                        /* convert diffusing gypsum to ettringite */
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                        /* decrement counts of diffusing gypsum */
                        count[DIFFGYP]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.40){
                                mic[xcur][ycur][zcur]=ETTR;
```

```
                        count[ETTR]+=1;
                        count[DIFFC3A]-=1;
                        nexp=1;
                }
                else{
      /* indicate that diffusing species remains in current location */
                        action=7;
                        nexp=2;
                }

/* Perform expansion that occurs when ettringite is formed */
/* xexp, yexp and zexp are the coordinates of the last ettringite */
/* pixel to be added */
                xexp=xnew;
                yexp=ynew;
                zexp=znew;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extettr(xexp,yexp,zexp,0);
                        /* update xexp, yexp and zexp */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;

                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                         }
                }

          /* probabilistic-based expansion for last ettringite pixel */
                pexp=ran1(seed);
                if(pexp<=0.30){
                        newact=extettr(xexp,yexp,zexp,0);
                }
           }
    /* examine reaction with diffusing hemihydrate to form ettringite */
```

```
                /* Only allow reaction with diffusing hemihydrate */
                else if((check==DIFFHEM)&&(p2diff<C3AGYP)){
                        /* convert diffusing hemihydrate to ettringite */
                        mic[xnew][ynew][znew]=ETTR;
                  count[ETTR]+=1;
                        /* decrement counts of diffusing hemihydrate */
                        count[DIFFHEM]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.5583){
                                mic[xcur][ycur][zcur]=ETTR;
                                count[ETTR]+=1;
                                count[DIFFC3A]-=1;
                                nexp=2;
                        }
                        else{
              /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,0);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
                                                break;
                                        case 6:
                                                zexp+=1;
                                                if(zexp>=SYSIZE){zexp=0;}
```

```
                                        break;
                                default:
                                        break;
                                 }
                        }

                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.6053){
                                newact=extettr(xexp,yexp,zexp,0);
                        }
                   }
                /* examine reaction with diffusing anhydrite to form ettringite */
                   /* Only allow reaction with diffusing anhydrite */
                   else if((check==DIFFANH)&&(p2diff<C3AGYP)){
                        /* convert diffusing anhydrite to ettringite */
                        mic[xnew][ynew][znew]=ETTR;
                        count[ETTR]+=1;
                        /* decrement counts of diffusing anhydrite */
                        count[DIFFANH]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.569){
                                mic[xcur][ycur][zcur]=ETTR;
                                count[ETTR]+=1;
                                count[DIFFC3A]-=1;
                                nexp=2;
                        }
                        else{
                 /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,0);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
```

```
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                }
                        }


                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.6935){
                                newact=extettr(xexp,yexp,zexp,0);
                        }
                }
                /* examine reaction with diffusing CaCl2 to form FREIDEL */
                /* Only allow reaction with diffusing CaCl2 */
                else if(check==DIFFCACL2){
                        /* convert diffusing C3A to Freidel's salt */
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        /* decrement counts of diffusing C3A and CaCl2 */
                        count[DIFFC3A]-=1;
                        action=0;

/* convert diffusing CACL2 to solid FREIDEL or else leave as a diffusing CACL2 */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.5793){
                                mic[xnew][ynew][znew]=FREIDEL;
                                count[FREIDEL]+=1;
                                count[DIFFCACL2]-=1;
                                nexp=1;
                        }
                        else{
                                nexp=2;
                        }

        /* Perform expansion that occurs when Freidel's salt is formed */
        /* xexp, yexp and zexp are the coordinates of the last FREIDEL */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extfreidel(xexp,yexp,zexp);
```

```
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
                                                break;
                                        case 6:
                                                zexp+=1;
                                                if(zexp>=SYSIZE){zexp=0;}
                                                break;
                                        default:
                                                break;
                                 }
                        }

                  /* probabilistic-based expansion for last FREIDEL pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.3295){
                                newact=extfreidel(xexp,yexp,zexp);
                        }
                }
                /* examine reaction with diffusing CAS2 to form STRAT */
                /* Only allow reaction with diffusing (not solid) CAS2 */
                else if(check==DIFFCAS2){
                        /* convert diffusing CAS2 to stratlingite */
                        mic[xnew][ynew][znew]=STRAT;
                        count[STRAT]+=1;
                        /* decrement counts of diffusing C3A and CAS2 */
                        count[DIFFCAS2]-=1;
                        action=0;

/* convert diffusing C3A to solid STRAT or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.886){
                                mic[xcur][ycur][zcur]=STRAT;
                                count[STRAT]+=1;
                                count[DIFFC3A]-=1;
                                nexp=2;
                        }
                        else{
```

```
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when stratlingite is formed */
        /* xexp, yexp and zexp are the coordinates of the last STRAT */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extstrat(xexp,yexp,zexp);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
                                                break;
                                        case 6:
                                                zexp+=1;
                                                if(zexp>=SYSIZE){zexp=0;}
                                                break;
                                        default:
                                                break;
                                 }
                        }

                  /* probabilistic-based expansion for last STRAT pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.286){
                                newact=extstrat(xexp,yexp,zexp);
                        }
                }
/* check for reaction with diffusing or solid ettringite to form AFm */
/* reaction at solid ettringite only possible if ettringite is soluble */
/* and even then on a limited bases to avoid a great formation of AFm */
/* when ettringite first becomes soluble */
                pgrow=ran1(seed);
   if((check==DIFFETTR)||((check==ETTR)&&(soluble[ETTR]==1)&&(pgrow<=C3AETTR))){
                /* convert diffusing or solid ettringite to AFm */
                mic[xnew][ynew][znew]=AFM;
```

```
                count[AFM]+=1;
                /* decrement count of ettringite */
                count[check]-=1;
                action=0;

                /* convert diffusing C3A to AFm or leave as diffusing C3A */
                pexp=ran1(seed);
                if(pexp<=0.2424){
                        mic[xcur][ycur][zcur]=AFM;
                        count[AFM]+=1;
                        count[DIFFC3A]-=1;
                        pafm=(-0.1);
                }
                else{
                        action=7;
                        pafm=0.04699;
                }

                /* probabilistic-based expansion for new AFm pixel */
                pexp=ran1(seed);
                if(pexp<=pafm){
                        extafm(xnew,ynew,znew);
                }
        }
        if((action!=0)&&(action!=7)){

                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFC3A;
                }
                else{
                        /* indicate that diffusing C3A remained */
                        /* at original location */
                        action=7;
                }
        }
        }
        return(action);
}

/* routine to move a diffusing C4A species */
/* from location (xcur,ycur,zcur) with nucleation probability of nucprob */
/* Called by hydrate */
/* Calls extc3ah6, moveone, extettr, and extafm */
int movec4a(xcur,ycur,zcur,finalstep,nucprob)
        int xcur,ycur,zcur,finalstep;
        float nucprob;
{
        int check,xnew,ynew,znew,plok,action,sumgarb,sumold;
        int xexp,yexp,zexp,nexp,iexp,newact;
        float pgen,pexp,pafm,pgrow,p2diff;

        /* First be sure that a diffusing C4A species is at (xcur,ycur,zcur) */
        if(mic[xcur][ycur][zcur]!=DIFFC4A){
                action=0;
                return(action);
```

```
}

/* Check for nucleation into solid C3AH6 */
pgen=ran1(seed);
p2diff=ran1(seed);

if((nucprob>=pgen)||(finalstep==1)){
        action=0;
        mic[xcur][ycur][zcur]=C3AH6;
        count[C3AH6]+=1;
        /* decrement count of diffusing C3A species */
        count[DIFFC4A]-=1;
        /* allow for probabilistic-based expansion of C3AH6 */
        /* crystal to account for volume stoichiometry */
        pexp=ran1(seed);
        if(pexp<=0.69){
                extc3ah6(xcur,ycur,zcur);
        }
}
else{
        /* determine new coordinates (using periodic boundaries) */
        xnew=xcur;
        ynew=ycur;
        znew=zcur;
        action=0;
        sumold=1;
        sumgarb=moveone(&xnew,&ynew,&znew,&action,sumold);
        if(action==0){printf("Error in value of action in movec4a \n");}
        check=mic[xnew][ynew][znew];

        /* check for growth of C3AH6 crystal */
        if(check==C3AH6){
                pgrow=ran1(seed);
                /* Try to slow down growth of C3AH6 crystals to */
                /* promote ettringite and Afm formation */
                if(pgrow<=C3AH6GROW){
                        mic[xcur][ycur][zcur]=C3AH6;
                        count[C3AH6]+=1;
                        count[DIFFC4A]-=1;
                        action=0;
                 /* allow for probabilistic-based expansion of C3AH6 */
                 /* crystal to account for volume stoichiometry */
                        pexp=ran1(seed);
                        if(pexp<=0.69){
                                extc3ah6(xcur,ycur,zcur);
                        }
                }
        }

        /* examine reaction with diffusing gypsum to form ettringite */
        /* Only allow reaction with diffusing gypsum */
        else if((check==DIFFGYP)&&(p2diff<C3AGYP)){
                /* convert diffusing gypsum to ettringite */
                mic[xnew][ynew][znew]=ETTRC4AF;
                count[ETTRC4AF]+=1;
                /* decrement counts of diffusing gypsum */
                count[DIFFGYP]-=1;
```

```
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.40){
                                mic[xcur][ycur][zcur]=ETTRC4AF;
                                count[ETTRC4AF]+=1;
                                count[DIFFC4A]-=1;
                                nexp=1;
                        }
                        else{
                /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=2;
                        }


        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;

                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,1);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
                                                break;
                                        case 6:
                                                zexp+=1;
                                                if(zexp>=SYSIZE){zexp=0;}
                                                break;
                                        default:
                                                break;
                                 }
                        }
```

```
                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.30){
                                newact=extettr(xexp,yexp,zexp,1);
                        }
                }
                /* examine reaction with diffusing hemi to form ettringite */
                /* Only allow reaction with diffusing hemihydrate */
                else if((check==DIFFHEM)&&(p2diff<C3AGYP)){
                        /* convert diffusing hemihydrate to ettringite */
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        /* decrement counts of diffusing hemihydrate */
                        count[DIFFHEM]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.5583){
                                mic[xcur][ycur][zcur]=ETTRC4AF;
                                count[ETTRC4AF]+=1;
                                count[DIFFC4A]-=1;
                                nexp=2;
                        }
                        else{
              /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,1);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
```

```
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                }
                        }

                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.6053){
                                newact=extettr(xexp,yexp,zexp,1);
                        }
                }
              /* examine reaction with diffusing anhydrite to form ettringite */
                 /* Only allow reaction with diffusing anhydrite */
                 else if((check==DIFFANH)&&(p2diff<C3AGYP)){
                        /* convert diffusing anhydrite to ettringite */
                        mic[xnew][ynew][znew]=ETTRC4AF;
                        count[ETTRC4AF]+=1;
                        /* decrement counts of diffusing anhydrite */
                        count[DIFFANH]-=1;
                        action=0;

/* convert diffusing C3A to solid ettringite or else leave as a diffusing C3A */
                        pexp=ran1(seed);
                        nexp=3;
                        if(pexp<=0.569){
                                mic[xcur][ycur][zcur]=ETTRC4AF;
                                count[ETTRC4AF]+=1;
                                count[DIFFC4A]-=1;
                                nexp=2;
                        }
                        else{
                 /* indicate that diffusing species remains in current location */
                                action=7;
                                nexp=3;
                        }

        /* Perform expansion that occurs when ettringite is formed */
        /* xexp, yexp and zexp are the coordinates of the last ettringite */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extettr(xexp,yexp,zexp,1);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
```

```
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;
                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                                }
                        }


                  /* probabilistic-based expansion for last ettringite pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.6935){
                                newact=extettr(xexp,yexp,zexp,1);
                        }
                }
                /* examine reaction with diffusing CaCl2 to form FREIDEL */
                /* Only allow reaction with diffusing CaCl2 */
                else if(check==DIFFCACL2){
                        /* convert diffusing C3A to Freidel's salt */
                        mic[xcur][ycur][zcur]=FREIDEL;
                        count[FREIDEL]+=1;
                        /* decrement counts of diffusing C3A and CaCl2 */
                        count[DIFFC4A]-=1;
                        action=0;

/* convert diffusing CACL2 to solid FREIDEL or else leave as a diffusing CACL2 */
                        pexp=ran1(seed);
                        nexp=2;
                        if(pexp<=0.5793){
                                mic[xnew][ynew][znew]=FREIDEL;
                                count[FREIDEL]+=1;
                                count[DIFFCACL2]-=1;
                                nexp=1;
                        }
                        else{
                                nexp=2;
                        }
```

```
        /* Perform expansion that occurs when Freidel's salt is formed */
        /* xexp, yexp and zexp are the coordinates of the last FREIDEL */
        /* pixel to be added */
                        xexp=xnew;
                        yexp=ynew;
                        zexp=znew;
                        for(iexp=1;iexp<=nexp;iexp++){
                                newact=extfreidel(xexp,yexp,zexp);
                                /* update xexp, yexp and zexp */
                                switch (newact){
                                        case 1:
                                                xexp-=1;
                                                if(xexp<0){xexp=(SYSIZEM1);}
                                                break;
                                        case 2:
                                                xexp+=1;
                                                if(xexp>=SYSIZE){xexp=0;}
                                                break;
                                        case 3:
                                                yexp-=1;
                                                if(yexp<0){yexp=(SYSIZEM1);}
                                                break;
                                        case 4:
                                                yexp+=1;
                                                if(yexp>=SYSIZE){yexp=0;}
                                                break;
                                        case 5:
                                                zexp-=1;
                                                if(zexp<0){zexp=(SYSIZEM1);}
                                                break;
                                        case 6:
                                                zexp+=1;
                                                if(zexp>=SYSIZE){zexp=0;}
                                                break;
                                        default:
                                                break;
                                 }
                        }

                  /* probabilistic-based expansion for last FREIDEL pixel */
                        pexp=ran1(seed);
                        if(pexp<=0.3295){
                                newact=extfreidel(xexp,yexp,zexp);
                        }
                }
                /* examine reaction with diffusing CAS2 to form STRAT */
                /* Only allow reaction with diffusing (not solid) CAS2 */
                else if(check==DIFFCAS2){
                        /* convert diffusing CAS2 to stratlingite */
                        mic[xnew][ynew][znew]=STRAT;
                        count[STRAT]+=1;
                        /* decrement counts of diffusing CAS2 */
                        count[DIFFCAS2]-=1;
                        action=0;

/* convert diffusing C3A to solid STRAT or else leave as a diffusing C3A */
                        pexp=ran1(seed);
```

```
                nexp=3;
                if(pexp<=0.886){
                        mic[xcur][ycur][zcur]=STRAT;
                        count[STRAT]+=1;
                        count[DIFFC4A]-=1;
                        nexp=2;
                }
                else{
                        action=7;
                        nexp=3;
                }

/* Perform expansion that occurs when stratlingite is formed */
/* xexp, yexp and zexp are the coordinates of the last STRAT */
/* pixel to be added */
                xexp=xnew;
                yexp=ynew;
                zexp=znew;
                for(iexp=1;iexp<=nexp;iexp++){
                        newact=extstrat(xexp,yexp,zexp);
                        /* update xexp, yexp and zexp */
                        switch (newact){
                                case 1:
                                        xexp-=1;
                                        if(xexp<0){xexp=(SYSIZEM1);}
                                        break;
                                case 2:
                                        xexp+=1;
                                        if(xexp>=SYSIZE){xexp=0;}
                                        break;
                                case 3:
                                        yexp-=1;
                                        if(yexp<0){yexp=(SYSIZEM1);}
                                        break;

                                case 4:
                                        yexp+=1;
                                        if(yexp>=SYSIZE){yexp=0;}
                                        break;
                                case 5:
                                        zexp-=1;
                                        if(zexp<0){zexp=(SYSIZEM1);}
                                        break;
                                case 6:
                                        zexp+=1;
                                        if(zexp>=SYSIZE){zexp=0;}
                                        break;
                                default:
                                        break;
                         }
                }

          /* probabilistic-based expansion for last STRAT pixel */
                pexp=ran1(seed);
                if(pexp<=0.286){
                        newact=extstrat(xexp,yexp,zexp);
                }
```

```
                }
/* check for reaction with diffusing or solid ettringite to form AFm */
/* reaction at solid ettringite only possible if ettringite is soluble */
/* and even then on a limited bases to avoid a great formation of AFm */
/* when ettringite first becomes soluble */
                pgrow=ran1(seed);
   if((check==DIFFETTR)||((check==ETTR)&&(soluble[ETTR]==1)&&(pgrow<=C3AETTR))){
                /* convert diffusing or solid ettringite to AFm */
                mic[xnew][ynew][znew]=AFM;
                count[AFM]+=1;
                /* decrement count of ettringite */
                count[check]-=1;
                action=0;

                /* convert diffusing C4A to AFm or leave as diffusing C4A */
                pexp=ran1(seed);
                if(pexp<=0.2424){
                        mic[xcur][ycur][zcur]=AFM;
                        count[AFM]+=1;
                        count[DIFFC4A]-=1;
                        pafm=(-0.1);
                }
                else{
                        action=7;
                        pafm=0.04699;
                }

                /* probabilistic-based expansion for new AFm pixel */
                pexp=ran1(seed);
                if(pexp<=pafm){
                        extafm(xnew,ynew,znew);
                }
        }
        if((action!=0)&&(action!=7)){

                /* if diffusion is possible, execute it */
                if(check==POROSITY){
                        mic[xcur][ycur][zcur]=POROSITY;
                        mic[xnew][ynew][znew]=DIFFC4A;
                }
                else{
                        /* indicate that diffusing C4A remained */
                        /* at original location */
                        action=7;
                }
        }
        }
        return(action);
}

/* routine to oversee hydration by updating position of all */
/* remaining diffusing species */
/* Calls movech, movec3a, movefh3, moveettr, movecsh, and movegyp */
void
hydrate(fincyc,stepmax,chpar1,chpar2,hgpar1,hgpar2,fhpar1,fhpar2,gypar1,gypar2)
        int fincyc,stepmax;
        float chpar1,chpar2,hgpar1,hgpar2,fhpar1,fhpar2,gypar1,gypar2;
```

```
{
        int xpl,ypl,zpl,phpl,agepl,xpnew,ypnew,zpnew;
        float chprob,c3ah6prob,fh3prob,gypprob;
        long int icnt,nleft,ntodo,ndale;
        int istep,termflag,reactf;
        float beterm;
        struct ants *curant,*antgone;

        ntodo=nmade;
        nleft=nmade;
        termflag=0;

/* Perform diffusion until all reacted or max. # of diffusion steps reached */
        for(istep=1;((istep<=stepmax)&&(nleft>0));istep++){
                if((fincyc==1)&&(istep==stepmax)){termflag=1;}

                nleft=0;
                ndale=0;


                /* determine probabilities for CH and C3AH6 nucleation */
                beterm=exp(-(double)(count[DIFFCH])/chpar2);
                chprob=chpar1*(1.-beterm);
                beterm=exp(-(double)(count[DIFFC3A])/hgpar2);
                c3ah6prob=hgpar1*(1.-beterm);
                beterm=exp(-(double)(count[DIFFFH3])/fhpar2);
                fh3prob=fhpar1*(1.-beterm);
                beterm=exp(-(double)(count[DIFFANH]+count[DIFFHEM])/gypar2);
                gypprob=gypar1*(1.-beterm);

                /* Process each diffusing species in turn */
                curant=headant->nextant;
                while(curant!=NULL){
                        ndale+=1;
                        xpl=curant->x;
                        ypl=curant->y;
                        zpl=curant->z;
                        phpl=curant->id;
                        agepl=curant->cycbirth;

       /* based on ID, call appropriate routine to process diffusing species */
                        switch (phpl) {
                                case DIFFCSH:
                                        /* printf("Calling movecsh \n");
                                        fflush(stdout); */
                                        reactf=movecsh(xpl,ypl,zpl,termflag,agepl);
                                        break;
                                case DIFFANH:
                                        /* printf("Calling moveanh \n");
                                        fflush(stdout); */
                                      reactf=moveanh(xpl,ypl,zpl,termflag,gypprob);
                                        break;
                                case DIFFHEM:
                                        /* printf("Calling movehem \n");
                                        fflush(stdout); */
                                      reactf=movehem(xpl,ypl,zpl,termflag,gypprob);
                                        break;
```

```
        case DIFFCH:
                /* printf("Calling movech \n");
                fflush(stdout); */
          reactf=movech(xpl,ypl,zpl,termflag,chprob);
                break;
        case DIFFFH3:
                /* printf("Calling movefh3 \n");
                fflush(stdout); */
          reactf=movefh3(xpl,ypl,zpl,termflag,fh3prob);
                break;
        case DIFFGYP:
                /* printf("Calling movegyp \n");
                fflush(stdout); */
                reactf=movegyp(xpl,ypl,zpl,termflag);
                break;
        case DIFFC3A:
                /* printf("Calling movec3a \n");
                fflush(stdout); */
         reactf=movec3a(xpl,ypl,zpl,termflag,c3ah6prob);
                break;
        case DIFFC4A:
                /* printf("Calling movec4a \n");
                fflush(stdout); */
         reactf=movec4a(xpl,ypl,zpl,termflag,c3ah6prob);
                break;
        case DIFFETTR:
                /* printf("Calling moveettr \n");
                fflush(stdout); */
                reactf=moveettr(xpl,ypl,zpl,termflag);
                break;
        case DIFFCACL2:
                /* printf("Calling movecacl2 \n");
                fflush(stdout); */
                reactf=movecacl2(xpl,ypl,zpl,termflag);
                break;
        case DIFFCAS2:
                /* printf("Calling movecas2 \n");
                fflush(stdout); */
                reactf=movecas2(xpl,ypl,zpl,termflag);
                break;
        case DIFFAS:
                /* printf("Calling moveas \n");
                fflush(stdout); */
                reactf=moveas(xpl,ypl,zpl,termflag);
                break;
        case DIFFCACO3:
                /* printf("Calling movecaco3 \n");
                fflush(stdout); */
                reactf=movecaco3(xpl,ypl,zpl,termflag);
                break;
        default:
                printf("Error in ID of phase \n");
                break;
}

/* if no reaction */
if(reactf!=0){
```

```
                nleft+=1;
                xpnew=xpl;
                ypnew=ypl;
                zpnew=zpl;

                /* update location of diffusing species */
                switch (reactf) {
                        case 1:
                                xpnew-=1;
                                if(xpnew<0){xpnew=(SYSIZEM1);}
                                break;
                        case 2:
                                xpnew+=1;
                                if(xpnew>=SYSIZE){xpnew=0;}
                                break;
                        case 3:
                                ypnew-=1;
                                if(ypnew<0){ypnew=(SYSIZEM1);}
                                break;
                        case 4:
                                ypnew+=1;
                                if(ypnew>=SYSIZE){ypnew=0;}
                                break;
                        case 5:
                                zpnew-=1;
                                if(zpnew<0){zpnew=(SYSIZEM1);}
                                break;
                        case 6:
                                zpnew+=1;
                                if(zpnew>=SYSIZE){zpnew=0;}
                                break;
                        default:
                                break;
                }

                /* store new location of diffusing species */
                curant->x=xpnew;
                curant->y=ypnew;
                curant->z=zpnew;
                curant->id=phpl;
                curant=curant->nextant;
        } /* end of reactf!=0 loop */
        /* else remove ant from list */
        else{
                if(ndale==1){
                        headant->nextant=curant->nextant;

                }
                else{
                     (curant->prevant)->nextant=curant->nextant;
                }
                if(curant->nextant!=NULL){
                     (curant->nextant)->prevant=curant->prevant;
                }
                else{
                        tailant=curant->prevant;
                }
```

```
                                antgone=curant;
                                curant=curant->nextant;
                                free(antgone);
                                ngoing-=1;
                        }
                } /* end of curant loop */
                ntodo=nleft;
        } /* end of istep loop */
}
```

Program pHpred.c

```
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

/* Note that everything is being done on a one gram cement basis */
/* and we are assuming that 1 pixel is equivalent to 1 micrometer */
#define VOLFACTOR 0.00001 /* dm per pixel Note- dm*dm*dm = Liters */
#define MASSFACTOR 0.0001  /* cm per pixel - specific gravities in g/cm^3 */
/* Molar masses of ions and oxides from sodium and potassium */
/* Na = sodium, K = potassium */
#define MMNa 22.9898
#define MMK 39.102
#define MMNa2O 61.979
#define MMK2O 94.203
/* Basis for B factors must be adapted from 100 g to 1 g */
/* Reference: Taylor, H.F.W., "A Method for Predicting Alkali Ion */
/* Concentrations in Cement Pore Solutions," Advances in Cement Research */
/* Vol. 1, No. 1, 5-16, 1987.  */
#define BNa 0.00031     /* From Taylor paper in liters (31 mL/1000/ 100 g) */
#define BK 0.00020     /* From Taylor paper in liters (20 mL/1000/ 100 g) */
#define BprimeNa 0.0030 /* From Taylor paper in liters (3 mL/1000/ 1 g POZZ) */
#define BprimeK 0.0033 /* From Taylor paper in liters (3.3 mL/1000/ 1 g POZZ) */
/* Ksp values for CH and gypsum from Reardon*/
#define KspCH25C 0.00000646
#define KspGypsum 0.0000263
/* Approximate Ksp value for syngenite from Gartner, Tang, and Weiss */
/* JACerS, Vol. 68 (12), 667-673, 1985. */
#define KspSyngenite 0.00000010
#define SpecgravSyngenite 2.607   /* Source Taylor, H.F.W., Cement Chemistry */
#define KperSyn 2.0   /* moles of K+ per mole of syngenite */
/* Some activity stuff */
#define activeA0 0.0366     /* A at 295 K (from Ken Snyder) */
#define activeB0 0.01035    /* B at 295 K (from Ken Snyder) */
/* z are the absolute charges (valences) per ion */
#define zCa 2.
#define zSO4 2.
#define zOH 1.
#define zNa 1.
#define zK 1.
/* a is similar to an ionic radius (in Angstroms) */
#define aK 1.33
#define aCa 1.
```

```
#define aOH 3.
#define aNa 3.
#define aSO4 4.5     /* Estimate as S ionic radii + O ionic diameter */
/* Ionic conductivities (From Snyder, Feng, Keen, and Mason) */
/* and from CRC Hanbook of Chemistry and Physics (1983) pp. D-175 */
/* pore solution conductivity = sum (zi * [i]*lambdai) */
/* lambdai = (lambdai_0/(1.+Gi*(Istrength^0.5))) */
/* where Istrength is in units of M (mol/L) */
#define lambdaOH_0 198.0      /* Units: S cm-cm eq.^(-1) */
#define lambdaNa_0 50.1
#define lambdaK_0 73.5
#define lambdaSO4_0 39.5
#define lambdaCa_0 29.5       /* Note that CRC has 60./2 for this */
#define GOH 0.353             /* Units: (eq.^2 mol/L)^(-0.5) */
#define GK 0.548
#define GNa 0.733
#define GCa 0.771
#define GSO4 0.877
#define cm2perL2m 0.1         /* Conversion from cm2/Liter to 1/m */

/* From Numerical Recipes in C by Press et al. */

#include "nrutil.c"
#include "complex.c"

#define EPSS 6.e-8
#define MAXIT 100

/* From Numerical Recipes in C by Press et al. */

void laguer(a,m,x,eps,polish)
fcomplex a[],*x;
int m,polish;
float eps;
{
      int j,iter;
      float err,dxold,cdx,abx;
      fcomplex sq,h,gp,gm,g2,g,b,d,dx,f,x1;
      void nrerror();

      dxold=Cabs(*x);
      for (iter=1;iter<=MAXIT;iter++) {
            b=a[m];
            err=Cabs(b);
            d=f=Complex(0.0,0.0);
            abx=Cabs(*x);
            for (j=m-1;j>=0;j--) {
                  f=Cadd(Cmul(*x,f),d);
                  d=Cadd(Cmul(*x,d),b);
                  b=Cadd(Cmul(*x,b),a[j]);
                  err=Cabs(b)+abx*err;
            }
            err *= EPSS;
            if (Cabs(b) <= err) return;
            g=Cdiv(d,b);
            g2=Cmul(g,g);
            h=Csub(g2,RCmul(2.0,Cdiv(f,b)));
```

```
            sq=Csqrt(RCmul((float) (m-1),Csub(RCmul((float) m,h),g2)));
            gp=Cadd(g,sq);
            gm=Csub(g,sq);
            if (Cabs(gp) < Cabs(gm))gp=gm;
            dx=Cdiv(Complex((float) m,0.0),gp);
            x1=Csub(*x,dx);
            if (x->r == x1.r && x->i == x1.i) return;
            *x=x1;
            cdx=Cabs(dx);
            if (iter > 6 && cdx >= dxold) return;
            dxold=cdx;
            if (!polish)
                  if (cdx <= eps*Cabs(*x)) return;
      }
      nrerror("Too many iterations in routine LAGUER");
}


#undef EPSS
#undef MAXIT

#define EPS 2.0e-6
#define MAXM 100

/* From Numerical Recipes in C by Press et al. */

void zroots(a,m,roots,polish)
fcomplex a[],roots[];
int m,polish;
{
      int jj,j,i;
      fcomplex x,b,c,ad[MAXM];
      void laguer();

      for (j=0;j<=m;j++) ad[j]=a[j];
      for (j=m;j>=1;j--) {
            x=Complex(0.0,0.0);
            laguer(ad,j,&x,EPS,0);
            if (fabs(x.i) <= (2.0*EPS*fabs(x.r))) x.i=0.0;
            roots[j]=x;
            b=ad[j];
            for (jj=j-1;jj>=0;jj--) {
                  c=ad[jj];
                  ad[jj]=b;
                  b=Cadd(Cmul(x,b),c);
            }
      }
      if (polish)
            for (j=1;j<=m;j++)

                  laguer(a,m,&roots[j],EPS,1);
      for (j=2;j<=m;j++) {
            x=roots[j];
            for (i=j-1;i>=1;i--) {
                  if (roots[i].r <= x.r) break;
                  roots[i+1]=roots[i];
            }
```

```
            roots[i+1]=x;
      }
}

#undef EPS
#undef MAXM

void pHpred(){
        int j,syngen_change=0,syn_old=0;
        double concnaplus,conckplus;
        double concohminus,A,B,C,conctest,concsulfate1;
        double volpore,grams_cement;
        double releasedna,releasedk,activitySO4,activityK,test_precip;
        double activityCa,activityOH,Istrength,Anow,Bnow,Inew;
        double lambdasum=0.0,conductivity=0.0;
        fcomplex coef[5],roots[5];
        float sumbest,sumtest,pozzreact,KspCH;

      /* Update CH activity product based on current system temperature */
      /* Factors derived from fitting CH solubility vs. temperature */
      /* data in Taylor book (p. 117) */
      KspCH=KspCH25C*(1.534385-0.02057*temp_cur);

      if(conccaplus>1.0){conccaplus=0.0;}
      /* Calculate volume of pore solution in the concrete in Liters */

volpore=(double)count[POROSITY]+0.38*(double)count[CSH]+0.20*(double)count[POZZCSH]
+0.20*count[SLAGCSH];
      /* Convert from pixels (cubic micrometers) to liters (cubic decimeters) */
      volpore*=VOLFACTOR;
      volpore*=VOLFACTOR;
      volpore*=VOLFACTOR;
      /* Compute pore volume per gram of cement */
      grams_cement=cemmasswgyp*MASSFACTOR*MASSFACTOR*MASSFACTOR;
      /* Compute pore volume per gram of cement */
      volpore/=grams_cement;
      /* Compute grams of pozzolan which have reacted */
      pozzreact=((float)npr/1.35)*MASSFACTOR*MASSFACTOR*MASSFACTOR*specgrav[POZZ];
      /* Compute moles of released potassium and sodium per gram of cement*/
      if(time_cur>1.0){
                releasedk=(2.*(rspotassium+(totpotassium-rspotassium)*alpha_cur));
                releasedk/=MMK2O;
                releasedna=(2.*(rssodium+(totsodium-rssodium)*alpha_cur));
                releasedna/=MMNa2O;
      }
      else{
            /* Proportion the early time release over the first hour */
                /* 90% immediately and the remaining 10% over the first hour */
                /* based on limited data from Davide Zampini (MBT) */
                releasedk=(2.*((0.9+0.1*time_cur)*(rspotassium)+(totpotassium-
rspotassium)*alpha_cur));
                releasedk/=MMK2O;
                releasedna=(2.*((0.9+0.1*time_cur)*(rssodium)+(totsodium-
rssodium)*alpha_cur));
                releasedna/=MMNa2O;
      }
      /* Compute concentrations of K+ and Na+ in pore solution currently */
```

```
      /* Remember to decrease K+ by KperSyn*moles of syngenite precipitated */
      /* Units must be in moles/gram for both */
      conckplus=((releasedk-
moles_syn_precip*KperSyn)/(volpore+BK*alpha_cur+BprimeK*pozzreact));
      concnaplus=(releasedna)/(volpore+BNa*alpha_cur+BprimeNa*pozzreact);

      do{  /* while Syngenite precipitating loop */
      /* Now compute the activities (estimated) of Ca++ and OH- */
      activityCa=activityOH=activitySO4=activityK=1.0;
      Inew=0.0;
      if(((concnaplus+conckplus)>0.0)&&(soluble[ETTR]==0)){
        /* Factor of 1000 to convert from M to mmol/L */
        Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
        if(Istrength<1.0){Istrength=1.0;}
        while((abs(Istrength-Inew)/Istrength)>0.10){

      Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
              if(Istrength<1.0){Istrength=1.0;}

Anow=activeA0*295.*sqrt(295.)/((temp_cur+273.15)*sqrt(temp_cur+273.15));
              Bnow=activeB0*sqrt(295.)/(sqrt(temp_cur+273.15));
                /* Equations from papers of Marchand et al. */
              activityCa=(-
Anow*zCa*zCa*sqrt(Istrength))/(1.+aCa*Bnow*sqrt(Istrength));
              activityCa+=(0.2-
0.0000417*Istrength)*Anow*zCa*zCa*Istrength/sqrt(1000.);
              activityCa=exp(activityCa);
              activityOH=(-Anow*zOH*zOH*sqrt(Istrength))/
                        (1.+aOH*Bnow*sqrt(Istrength));
              activityOH+=(0.2-0.0000417*Istrength)
                        *Anow*zOH*zOH*Istrength/sqrt(1000.);
              activityOH=exp(activityOH);
              activityK=(-Anow*zK*zK*sqrt(Istrength))/(1.+aK*Bnow*sqrt(Istrength));
              activityK+=(0.2-0.0000417*Istrength)*Anow*
                        zK*zK*Istrength/sqrt(1000.);
              activityK=exp(activityK);
              activitySO4=(-Anow*zSO4*zSO4*sqrt(Istrength))/
                        (1.+aSO4*Bnow*sqrt(Istrength));
              activitySO4+=(0.2-0.0000417*Istrength)*Anow*zSO4*
                        zSO4*Istrength/sqrt(1000.);
              activitySO4=exp(activitySO4);
            /* Now try to find roots of fourth degree polynomial */
            /* to determine sulfate, OH-, and calcium ion concentrations */
            /*    A=(-KspCH); */
            /* Now with activities */
            A=(-KspCH/(activityCa*activityOH*activityOH));
            B=conckplus+concnaplus;
            C=(-2.*KspGypsum/(activityCa*activitySO4));
                  concohminus=conckplus+concnaplus;
            coef[0]=Complex(C,0.0);
            coef[1]=Complex((A+2.*B*C)/C,0.0);
            coef[2]=Complex(B*B/C+4.,0.0);
            coef[3]=Complex(4.*B/C,0.0);
            coef[4]=Complex(4./C,0.0);
      /*            printf("coef 0 is (%f,%f)\n",coef[0].r,coef[0].i);
                  printf("coef 1 is (%f,%f)\n",coef[1].r,coef[1].i);
                  printf("coef 2 is (%f,%f)\n",coef[2].r,coef[2].i);
```

```
            printf("coef 3 is (%f,%f)\n",coef[3].r,coef[3].i);
            printf("coef 4 is (%f,%f)\n",coef[4].r,coef[4].i); */
      roots[1]=Complex(0.0,0.0);
      roots[2]=Complex(0.0,0.0);
      roots[3]=Complex(0.0,0.0);
      roots[4]=Complex(0.0,0.0);
      zroots(coef,4,roots,1);
      sumbest=100;
      /* Find the best real root for electoneutrality */
      for(j=1;j<=4;j++){
            printf("pH root %d is (%f,%f)\n",j,roots[j].r,roots[j].i);
            fflush(stdout);
            if(((roots[j].i)==0.0)&&((roots[j].r)>0.0)){
       conctest=sqrt(KspCH/(roots[j].r*activityCa*activityOH*activityOH));
               concsulfate1=KspGypsum/(roots[j].r*activityCa*activitySO4);
       sumtest=concnaplus+conckplus+2.*roots[j].r-conctest-2.*concsulfate1;
            if(fabs(sumtest)<sumbest){
                        sumbest=fabs(sumtest);
                        concohminus=conctest;
                        conccaplus=roots[j].r;
                        concsulfate=concsulfate1;
                  }
            }
      }
      /* Update ionic strength */
        Inew=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
      }    /* end of while loop for Istrength-Inew */
}
else{
  /* Factor of 1000 to convert from M to mmol/L */
  Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
  if(Istrength<1.0){Istrength=1.0;}
  while((abs(Istrength-Inew)/Istrength)>0.10){

Istrength=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
 Anow=activeA0*295.*sqrt(295.)/((temp_cur+273.15)*sqrt(temp_cur+273.15));
        Bnow=activeB0*sqrt(295.)/(sqrt(temp_cur+273.15));
        /* Equations from papers of Marchand et al. */
        activityCa=(-Anow*zCa*zCa*sqrt(Istrength))/
                  (1.+aCa*Bnow*sqrt(Istrength));
        activityCa+=(0.2-0.0000417*Istrength)*Anow*zCa*
                  zCa*Istrength/sqrt(1000.);
        activityCa=exp(activityCa);
        activityOH=(-Anow*zOH*zOH*sqrt(Istrength))/
                  (1.+aOH*Bnow*sqrt(Istrength));
        activityOH+=(0.2-0.0000417*Istrength)*Anow*zOH*
                  zOH*Istrength/sqrt(1000.);
        activityOH=exp(activityOH);
        activityK=(-Anow*zK*zK*sqrt(Istrength))/(1.+aK*Bnow*sqrt(Istrength));
        activityK+=(0.2-0.0000417*Istrength)*Anow*zK*
                  zK*Istrength/sqrt(1000.);
        activityK=exp(activityK);
/* Calculate pH assuming simply that OH- balances sum of Na+ and K+ */
        concohminus=conckplus+concnaplus;
        if((conccaplus)>(0.1*(concohminus))){
              concohminus+=(2.*conccaplus);
        }
```

```
   conccaplus=(KspCH/(activityCa*activityOH*activityOH*concohminus*concohminus));
            concsulfate=0.0;
                   /* Update ionic strength */
             Inew=1000.*(zK*zK*conckplus+zNa*zNa*concnaplus+zCa*zCa*conccaplus);
             }    /* end of while loop for Istrength-Inew */
         }
           /* Check for syngenite precipitation */
           syngen_change=0;
           if(syn_old!=2){
              test_precip=conckplus*conckplus*activityK*activityK;
              test_precip*=conccaplus*activityCa;
              test_precip*=concsulfate*concsulfate*activitySO4*activitySO4;
              if(test_precip>KspSyngenite){
                 printf("Syngenite precipitating at cycle %d\n",icyc);
                 syngen_change=syn_old=1;
                 /* Units of moles_syn_precip are moles per gram of cement */
                 if(conckplus>0.002){
                    conckplus-=0.001;
                    moles_syn_precip+=0.001*volpore/KperSyn;
                 }
                 else if(conckplus>0.0002){
                    conckplus-=0.0001;
                    moles_syn_precip+=0.0001*volpore/KperSyn;
                 }
                 else{
                    moles_syn_precip+=conckplus*volpore/KperSyn;
                        conckplus=0.0;
                 }
              }
              /* Check for syngenite dissolution */
              /* How to control dissolution rates??? */
              /* Have 0.001*KperSyn increase in conckplus each cycle */
              /* Only one dissolution per cycle --- purpose of syn_old */
              /* and no dissolution if some precipitation in that cycle */
              if((syn_old==0)&&(moles_syn_precip>0.0)){
                syngen_change=syn_old=2;
/*                conckplus+=(moles_syn_precip/10.0)/volpore; */
                if((moles_syn_precip/volpore)>0.001){
                      conckplus+=0.001*KperSyn;
                          moles_syn_precip-=(0.001*volpore);
                }
                else{
                        conckplus+=(moles_syn_precip*KperSyn/volpore);
                        moles_syn_precip=0.0;
                }
              }
           }
         } while(syngen_change!=0);

      if(concohminus<(0.0000001)){
            concohminus=0.0000001;
   conccaplus=(KspCH/(activityCa*activityOH*activityOH*concohminus*concohminus));
      }
        pH_cur=14.0+log10(concohminus*activityOH);
        /* Calculation of solution conductivity (Snyder and Feng basis) */
        /* First convert ionic strength back to M units */
        Istrength/=1000.;
```

```
        conductivity+=zCa*conccaplus*(lambdaCa_0/(1.+GCa*sqrt(Istrength)));
        conductivity+=zOH*concohminus*(lambdaOH_0/(1.+GOH*sqrt(Istrength)));
        conductivity+=zNa*concnaplus*(lambdaNa_0/(1.+GNa*sqrt(Istrength)));
        conductivity+=zK*conckplus*(lambdaK_0/(1.+GK*sqrt(Istrength)));
        conductivity+=zSO4*concsulfate*(lambdaSO4_0/(1.+GSO4*sqrt(Istrength)));
        conductivity*=cm2perL2m;

      /* Output results to logging file */
      pHfile=fopen(pHname,"a");

      if((cyccnt-1)==0){
      fprintf(pHfile,"Cycle time(h) alpha_mass pH sigma(S/m) [Na+] [K+] [Ca++]
[SO4--] activityCa activityOH activitySO4 activityK molesSyngenite\n");
      }
      fprintf(pHfile,"%d %.4f %f %.4f %f %f %f %f %f %.4f %.4f %.4f %.4f
%f\n",cyccnt-1,time_cur,alpha_cur,pH_cur,conductivity,concnaplus,
conckplus,conccaplus,concsulfate,activityCa,activityOH,activitySO4,
activityK,moles_syn_precip);
        fclose(pHfile);

}
```

Program burn3d.c

```
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

#define BURNT 70      /* label for a burnt pixel */
#define SIZE2D 49000 /* size of matrices for holding burning locations */
/* functions defining coordinates for burning in any of three directions */
#define cx(x,y,z,a,b,c) (1-b-c)*x+(1-a-c)*y+(1-a-b)*z
#define cy(x,y,z,a,b,c) (1-a-b)*x+(1-b-c)*y+(1-a-c)*z
#define cz(x,y,z,a,b,c) (1-a-c)*x+(1-a-b)*y+(1-b-c)*z

/* routine to assess the connectivity (percolation) of a single phase */
/* Two matrices are used here: one to store the recently burnt locations */
/*                the other to store the newly found burnt locations */
int burn3d(npix,d1,d2,d3)
        int npix;    /* ID of phase to perform burning on */
        int d1,d2,d3;   /* directional flags */
{
        long int ntop,nthrough,ncur,nnew,ntot,nphc;
        int i,inew,j,k,nmatx[SIZE2D],nmaty[SIZE2D],nmatz[SIZE2D];
        int xl,xh,j1,k1,px,py,pz,qx,qy,qz,xcn,ycn,zcn;
        int x1,y1,z1,igood,nnewx[SIZE2D],nnewy[SIZE2D],nnewz[SIZE2D];
        int jnew,icur;
        int bflag;
        float mass_burn=0.0,alpha_burn=0.0,con_frac;
        FILE *fileperc;

/* counters for number of pixels of phase accessible from surface #1 */
/* and number which are part of a percolated pathway to surface #2 */
        ntop=0;
        bflag=0;
        nthrough=0;

        nphc=0;

        /* percolation is assessed from top to bottom only */
        /* and burning algorithm is periodic in other two directions */
        /* use of directional flags allow transformation of coordinates */
        /* to burn in direction of choosing (x, y, or z) */
        i=0;

        for(k=0;k<SYSIZE;k++){
```

```
        for(j=0;j<SYSIZE;j++){

                igood=0;
                ncur=0;
                ntot=0;
                /* Transform coordinates */
                px=cx(i,j,k,d1,d2,d3);
                py=cy(i,j,k,d1,d2,d3);
                pz=cz(i,j,k,d1,d2,d3);
                if(mic [px] [py] [pz]==npix){
                        /* Start a burn front */
                        mic [px] [py] [pz]=BURNT;
                        ntot+=1;
                        ncur+=1;
                        /* burn front is stored in matrices nmat* */
                        /* and nnew* */
                        nmatx[ncur]=i;
                        nmaty[ncur]=j;
                        nmatz[ncur]=k;
                        /* Burn as long as new (fuel) pixels are found */
                        do{
                                nnew=0;
                                    for(inew=1;inew<=ncur;inew++){
                                        xcn=nmatx[inew];
                                          ycn=nmaty[inew];
                                          zcn=nmatz[inew];

                                        /* Check all six neighbors */
                                        for(jnew=1;jnew<=6;jnew++){
                                                x1=xcn;
                                                y1=ycn;
                                                z1=zcn;
                                                if(jnew==1){x1-=1;}
                                                if(jnew==2){x1+=1;}
                                                if(jnew==3){y1-=1;}
                                                if(jnew==4){y1+=1;}
                                                if(jnew==5){z1-=1;}
                                                if(jnew==6){z1+=1;}
                                    /* Periodic in y and */
                                                if(y1>=SYSIZE){y1-=SYSIZE;}
                                                else if(y1<0){y1+=SYSIZE;}
                                    /* Periodic in z direction */
                                                if(z1>=SYSIZE){z1-=SYSIZE;}
                                                else if(z1<0){z1+=SYSIZE;}

/* Nonperiodic so be sure to remain in the 3-D box */
                                                if((x1>=0)&&(x1<SYSIZE)){
                                                /* Transform coordinates */
                                                px=cx(x1,y1,z1,d1,d2,d3);
                                                py=cy(x1,y1,z1,d1,d2,d3);
                                                pz=cz(x1,y1,z1,d1,d2,d3);
                                                if(mic [px] [py] [pz]==npix){
                                                   ntot+=1;
                                                   mic [px] [py] [pz]=BURNT;
                                                   nnew+=1;
                                                   if(nnew>=SIZE2D){
                                              printf("error in size of nnew \n");
```

```
                                        }
                                        nnewx[nnew]=x1;
                                        nnewy[nnew]=y1;
                                        nnewz[nnew]=z1;
                                    }
                                }
                                }
                        }
                        if(nnew>0){
                                ncur=nnew;
                                /* update the burn front matrices */
                                  for(icur=1;icur<=ncur;icur++){
                                        nmatx[icur]=nnewx[icur];
                                        nmaty[icur]=nnewy[icur];
                                        nmatz[icur]=nnewz[icur];
                                }
                        }
                    }while (nnew>0);

                    ntop+=ntot;
                    xl=0;
                    xh=SYSIZE-1;
                 /* See if current path extends through the microstructure */
                    for(j1=0;j1<SYSIZE;j1++){
                    for(k1=0;k1<SYSIZE;k1++){
                            px=cx(xl,j1,k1,d1,d2,d3);
                            py=cy(xl,j1,k1,d1,d2,d3);
                            pz=cz(xl,j1,k1,d1,d2,d3);
                            qx=cx(xh,j1,k1,d1,d2,d3);
                            qy=cy(xh,j1,k1,d1,d2,d3);
                            qz=cz(xh,j1,k1,d1,d2,d3);
                  if((mic [px] [py] [pz]==BURNT)&&(mic [qx] [qy] [qz]==BURNT)){
                                igood=2;
                            }
                            if(mic [px] [py] [pz]==BURNT){
                                mic [px] [py] [pz]=BURNT+1;
                            }
                            if(mic [qx] [qy] [qz]==BURNT){
                                mic [qx] [qy] [qz]=BURNT+1;
                            }
                    }
                    }

                    if(igood==2){
                            nthrough+=ntot;
                    }
                }
        }
        }
/* return the burnt sites to their original phase values */
        for(i=0;i<SYSIZE;i++){
        for(j=0;j<SYSIZE;j++){
        for(k=0;k<SYSIZE;k++){
            if(mic [i] [j] [k]>=BURNT){
                        nphc+=1;
                        mic [i] [j] [k]=npix;
            }
```

```
            else if(mic[i][j][k]==npix){
                        nphc+=1;
            }
        }
        }
        }

        printf("Phase ID= %d \n",npix);
        printf("Number accessible from first surface = %ld \n",ntop);
        printf("Number contained in through pathways= %ld \n",nthrough);
        fileperc=fopen(ppsname,"a");
        mass_burn+=specgrav[C3S]*count[C3S];
        mass_burn+=specgrav[C2S]*count[C2S];
        mass_burn+=specgrav[C3A]*count[C3A];
        mass_burn+=specgrav[C4AF]*count[C4AF];
        alpha_burn=1.-(mass_burn/cemmass);
        con_frac=0.0;
        if(nphc>0){
              con_frac=(float)nthrough/(float)nphc;
        }
        fprintf(fileperc,"%ld %f %f %ld %ld
%f\n",cyccnt,time_cur+(2.*(float)(cyccnt)-
1.0)*beta/krate,alpha_burn,nthrough,nphc,con_frac);
        fclose(fileperc);
        if(nthrough>0){
            bflag=1;
        }
        return(bflag);
}
```

Program burnset.c

```
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

#define BURNT 70    /* label for burnt pixels */
#define SIZESET 100000
/* Transformation functions for changing direction of burn propagation */
#define cx(x,y,z,a,b,c) (1-b-c)*x+(1-a-c)*y+(1-a-b)*z
#define cy(x,y,z,a,b,c) (1-a-b)*x+(1-b-c)*y+(1-a-c)*z
#define cz(x,y,z,a,b,c) (1-a-c)*x+(1-a-b)*y+(1-b-c)*z

/* routine to assess connectivity (percolation) of solids for set estimation */
/* Definition of set is a through pathway of cement and fly ash (slag)  */
/* particles connected together by CSH, C3AH6, or ettringite */
/* Two matrices are used here: one to store the recently burnt locations */
/*                the other to store the newly found burnt locations */
int burnset(d1,d2,d3)
        int d1,d2,d3;
{
        long int ntop,nthrough,icur,inew,ncur,nnew,ntot,count_solid;
        int i,j,k,setyet;
        static int nmatx[SIZESET],nmaty[SIZESET],nmatz[SIZESET];
        int xl,xh,j1,k1,px,py,pz,qx,qy,qz;
        int xcn,ycn,zcn,x1,y1,z1,igood;
        static int nnewx[SIZESET],nnewy[SIZESET],nnewz[SIZESET];
        int jnew;
        float mass_burn=0.0,alpha_burn=0.0,con_frac;
        FILE *percfile;
        static char newmat [SYSIZE] [SYSIZE] [SYSIZE];

/* counters for number of pixels of phase accessible from surface #1 */
/* and number which are part of a percolated pathway to surface #2 */
        ntop=0;
        nthrough=0;
        setyet=0;
        for(k=0;k<SYSIZE;k++){
        for(j=0;j<SYSIZE;j++){
        for(i=0;i<SYSIZE;i++){
                newmat[i][j][k]=mic[i][j][k];

        }
        }
```

```
        }

        /* percolation is assessed from top to bottom only */
        /* in transformed coordinates */
        /* and burning algorithm is periodic in other two directions */
        i=0;

        for(k=0;k<SYSIZE;k++){
        for(j=0;j<SYSIZE;j++){

                igood=0;
                ncur=0;
                ntot=0;
                /* Transform coordinates */
                px=cx(i,j,k,d1,d2,d3);
                py=cy(i,j,k,d1,d2,d3);
                pz=cz(i,j,k,d1,d2,d3);
/* start from a cement clinker, slag, fly ash ettringite, C3AH6, or CSH pixel */
         if((mic [px] [py] [pz]==C3S) ||
                (mic [px] [py] [pz]==C2S) ||
                (mic [px] [py] [pz]==SLAG) ||
                (mic [px] [py] [pz]==ASG) ||
                (mic [px] [py] [pz]==CAS2) ||
                (mic [px] [py] [pz]==POZZ) ||
                (mic [px] [py] [pz]==CSH) ||
                (mic [px] [py] [pz]==C3AH6) ||
                (mic [px] [py] [pz]==ETTR) ||
                (mic [px] [py] [pz]==ETTRC4AF) ||
                (mic [px] [py] [pz]==C3A) ||
                (mic [px] [py] [pz]==C4AF)){
                        /* Start a burn front */
                        mic [px] [py] [pz]=BURNT;
                        ntot+=1;
                        ncur+=1;
                        /* burn front is stored in matrices nmat* */
                        /* and nnew* */
                        nmatx[ncur]=i;
                        nmaty[ncur]=j;
                        nmatz[ncur]=k;
                        /* Burn as long as new (fuel) pixels are found */
                        do{
                                nnew=0;
                                    for(inew=1;inew<=ncur;inew++){
                                        xcn=nmatx[inew];
                                        ycn=nmaty[inew];
                                        zcn=nmatz[inew];
                                        /* Convert to directional coordinates */
                                        qx=cx(xcn,ycn,zcn,d1,d2,d3);
                                        qy=cy(xcn,ycn,zcn,d1,d2,d3);
                                        qz=cz(xcn,ycn,zcn,d1,d2,d3);

                                        /* Check all six neighbors */
                                        for(jnew=1;jnew<=6;jnew++){
                                                x1=xcn;
                                                y1=ycn;
                                                z1=zcn;
                                                if(jnew==1){x1-=1;}
```

```
                                                    if(jnew==2){x1+=1;}
                                                    if(jnew==3){y1-=1;}
                                                    if(jnew==4){y1+=1;}
                                                    if(jnew==5){z1-=1;}
                                                    if(jnew==6){z1+=1;}
                                                    /* Periodic in y and */
                                                    if(y1>=SYSIZE){y1-=SYSIZE;}
                                                    else if(y1<0){y1+=SYSIZE;}
                                                    /* Periodic in z direction */
                                                    if(z1>=SYSIZE){z1-=SYSIZE;}
                                                    else if(z1<0){z1+=SYSIZE;}

/* Nonperiodic so be sure to remain in the 3-D box */
                                                    if((x1>=0)&&(x1<SYSIZE)){
                                                    px=cx(x1,y1,z1,d1,d2,d3);
                                                    py=cy(x1,y1,z1,d1,d2,d3);
                                                    pz=cz(x1,y1,z1,d1,d2,d3);
                /* Conditions for propagation of burning */
                /* 1) new pixel is CSH or ETTR or C3AH6 */

if((mic[px][py][pz]==CSH)||(mic[px][py][pz]==ETTRC4AF)||(mic[px][py][pz]==C3AH6)||(
mic[px][py][pz]==ETTR)){
                        ntot+=1;
                        mic [px] [py] [pz]=BURNT;
                        nnew+=1;
                        if(nnew>=SIZESET){
                              printf("error in size of nnew %d\n", nnew);
                        }
                        nnewx[nnew]=x1;
                        nnewy[nnew]=y1;
                        nnewz[nnew]=z1;
                }
/* 2) old pixel is CSH or ETTR or C3AH6 and new pixel is one of cement clinker,
slag, of fly ash phases */
                else
if(((newmat[qx][qy][qz]==CSH)||(newmat[qx][qy][qz]==ETTRC4AF)||(newmat[qx][qy][qz]=
=C3AH6)||(newmat[qx][qy][qz]==ETTR))
                &&((mic [px] [py] [pz]==C3S) ||
                (mic [px] [py] [pz]==C2S) ||
                (mic [px] [py] [pz]==CAS2) ||
                (mic [px] [py] [pz]==SLAG) ||
                (mic [px] [py] [pz]==POZZ) ||
                (mic [px] [py] [pz]==ASG) ||
                (mic [px] [py] [pz]==C3A) ||
                (mic [px] [py] [pz]==C4AF))){
                                ntot+=1;
                                mic [px] [py] [pz]=BURNT;
                                nnew+=1;
                                if(nnew>=SIZESET){
                                   printf("error in size of nnew %d\n", nnew);
                                }
                                nnewx[nnew]=x1;
                                nnewy[nnew]=y1;
                                nnewz[nnew]=z1;
                }
        /* 3) old and new pixels belong to one of cement clinker, slag, or fly ash
phases and */
```

```
/* are contained in the same initial cement particle */
/* and it is not a one-pixel particle */
        else if((micpart[qx][qy][qz]==micpart[px][py][pz])
        &&(micpart[qx][qy][qz]!=0)
        &&((mic [px] [py] [pz]==C3S) ||
        (mic [px] [py] [pz]==C2S) ||
        (mic [px] [py] [pz]==POZZ) ||
        (mic [px] [py] [pz]==SLAG) ||
        (mic [px] [py] [pz]==ASG) ||
        (mic [px] [py] [pz]==CAS2) ||
        (mic [px] [py] [pz]==C3A) ||
        (mic [px] [py] [pz]==C4AF))&&((newmat[qx][qy][qz]==C3S)||
        (newmat [qx] [qy] [qz]==C2S) ||
        (newmat [qx] [qy] [qz]==SLAG) ||
        (newmat [qx] [qy] [qz]==ASG) ||
        (newmat [qx] [qy] [qz]==POZZ) ||
        (newmat [qx] [qy] [qz]==CAS2) ||
        (newmat [qx] [qy] [qz]==C3A) ||
        (newmat[qx][qy][qz]==C4AF))){
                                ntot+=1;
                                mic [px] [py] [pz]=BURNT;
                                nnew+=1;
                                if(nnew>=SIZESET){
                             printf("error in size of nnew %d\n", nnew);
                                }
                                nnewx[nnew]=x1;
                                nnewy[nnew]=y1;
                                nnewz[nnew]=z1;
        }
                               } /* nonperiodic if delimiter */
                               } /* neighbors loop */
                        } /* propagators loop */
                            if(nnew>0){
                               ncur=nnew;
                               /* update the burn front matrices */
                               for(icur=1;icur<=ncur;icur++){
                                       nmatx[icur]=nnewx[icur];
                                       nmaty[icur]=nnewy[icur];
                                       nmatz[icur]=nnewz[icur];
                               }
                            }
                }while (nnew>0);

                ntop+=ntot;
                xl=0;
                xh=SYSIZE-1;
                /* Check for percolated path through system */
                for(j1=0;j1<SYSIZE;j1++){
                for(k1=0;k1<SYSIZE;k1++){
                        px=cx(xl,j1,k1,d1,d2,d3);
                        py=cy(xl,j1,k1,d1,d2,d3);
                        pz=cz(xl,j1,k1,d1,d2,d3);
                        qx=cx(xh,j1,k1,d1,d2,d3);
                        qy=cy(xh,j1,k1,d1,d2,d3);
                        qz=cz(xh,j1,k1,d1,d2,d3);
          if((mic [px] [py] [pz]==BURNT)&&(mic [qx] [qy] [qz]==BURNT)){
                               igood=2;
```

```
                                }
                                    if(mic [px] [py] [pz]==BURNT){
                                        mic [px] [py] [pz]=BURNT+1;
                                }
                                if(mic [qx] [qy] [qz]==BURNT){
                                          mic [qx] [qy] [qz]=BURNT+1;
                                }
                        }
                        }

                        if(igood==2){
                               nthrough+=ntot;
                        }
                }
        }
            }

            printf("Phase ID= Solid Phases \n");
            printf("Number accessible from first surface = %ld \n",ntop);
            printf("Number contained in through pathways= %ld \n",nthrough);
            percfile=fopen(ptsname,"a");
            mass_burn+=specgrav[C3S]*count[C3S];
            mass_burn+=specgrav[C2S]*count[C2S];
            mass_burn+=specgrav[C3A]*count[C3A];
            mass_burn+=specgrav[C4AF]*count[C4AF];
            alpha_burn=1.-(mass_burn/cemmass);
            con_frac=0.0;
      count_solid=count[C3S]+count[C2S]+count[C3A]+count[C4AF]+count[ETTR]+count[CS
H]+count[C3AH6]+count[ETTRC4AF]+count[POZZ]+count[ASG]+count[SLAG]+count[CAS2];
            if(count_solid>0){
            con_frac=(float)nthrough/(float)count_solid;
      }
      fprintf(percfile,"%ld  %f %f  %ld %ld
%f\n",cyccnt,time_cur+(2.*(float)(cyccnt)-
1.0)*beta/krate,alpha_burn,nthrough,count[C3S]+count[C2S]+count[C3A]+count[C4AF]+co
unt[CAS2]+count[SLAG]+count[ASG]+count[POZZ]+count[ETTR]+count[C3AH6]+count[ETTRC4A
F]+count[CSH],con_frac);
      fclose(percfile);
      if(con_frac>0.985){setyet=1;}

/* return the burnt sites to their original phase values */
        for(i=0;i<SYSIZE;i++){
        for(j=0;j<SYSIZE;j++){
        for(k=0;k<SYSIZE;k++){
                if(mic [i] [j] [k]>=BURNT){
                        mic [i] [j] [k]= newmat [i] [j] [k];
                  }
        }
        }
        }
        /* Return flag indicating if set has indeed occurred */
        return(setyet);
}
```

Program parthyd.c

```
/* This software was developed at the National Institute of */
/* Standards and Technology by employees of the Federal Government */
/* in the course of their official duties. Pursuant to title 17 */
/* Section 105 of the United States Code this software is not */
/* subject to copyright protection and is in the public domain. */
/* CEMHYD3D is an experimental system. NIST assumes no */
/* responsibility  whatsoever for its use by other parties, and */
/* makes no guarantees, expressed or implied, about its quality, */
/* reliability, or any other characteristic. We would appreciate */
/* acknowledgement if the software is used.  This software can be */
/* redistributed and/or modified freely provided that any */
/* derivative works bear some notice that they are derived from it, */
/* and any modified versions bear some notice that they have been */
/* modified.  */

/* Routine to assess relative particle hydration */
void parthyd(){
      int norig[50000],nleft[50000];
      int ix,iy,iz;
      char valmic,valmicorig;
      int valpart,partmax;
      float alpart;
      FILE *phydfile;

      /* Initialize the particle count arrays */
      for(ix=0;ix<50000;ix++){
            nleft[ix]=norig[ix]=0;
      }
      phydfile=fopen(phrname,"a");
      fprintf(phydfile,"%d %f\n",cyccnt,alpha_cur);

      partmax=0;
      /* Scan the microstructure pixel by pixel and update counts */
      for(ix=0;ix<SYSIZE;ix++){
      for(iy=0;iy<SYSIZE;iy++){
      for(iz=0;iz<SYSIZE;iz++){

            if(micpart[ix][iy][iz]!=0){
                  valpart=micpart[ix][iy][iz];
                  if(valpart>partmax){partmax=valpart;}
                  valmic=mic[ix][iy][iz];
                  if((valmic==C3S)||(valmic==C2S)||(valmic==C3A)||(valmic==C4AF)){
                        nleft[valpart]+=1;
                  }
                  valmicorig=micorig[ix][iy][iz];

      if((valmicorig==C3S)||(valmicorig==C2S)||(valmicorig==C3A)||(valmicorig==C4AF
)){
                        norig[valpart]+=1;
                  }
            }
      }
      }
```

```
	}

	  /* Output results to end of particle hydration file */
	for(ix=100;ix<=partmax;ix++){
		alpart=0.0;
		if(norig[ix]!=0){
			alpart=1.-(float)nleft[ix]/(float)norig[ix];
		}
		fprintf(phydfile,"%d %d %d %.3f\n",ix,norig[ix],nleft[ix],alpart);
	}
	fclose(phydfile);
}
```