VRen: a C++ class for VR volume rendering
Final Project Report for EECS590

Randy Hudson
University of Illinois at Chicago

April 30, 1996

PROBLEM STATEMENT

A great deal of data that we want to visualize in a virtual environment is volumetric: both scalar and vector data. Examples of scalar data are 1) measured physical systems such as medical CT scan data, 2) simulations of physical systems such as computations of star formation and 3) 3D mathematical sets. Examples of vector data are the output from fluid dynamics computations. An important technique for visualizing volumetric data is "volume rendering".

Only recently has it become possible to volume render volumetric data rapidly enough in a virtual environment to satisfy the VR requirement of real-time (at least 10-frame-per-second) scene updates. Also, each CAVE developer has had to write unique code to implement this technique.

1 PREVIOUS WORK

1.1 Volumetric data and visualization

A volumetric dataset is a 3D grid of either scalar or vector data elements. Most volumetric dataset grids are rectangular, and, so far, the work described here deals only with those. There is one data element per grid point. The computer graphical methods that have been developed to visualize this kind of data are referred to collectively as "volume visualization".

Volume visualization techniques are usually separated into two groups: iso-surface modeling and (direct) volume rendering[5].

1.2 Volume rendering and texture mapping

Volume rendering usually displays volumetric data as a translucent substance of varying opacity, making it possible to view the entire volume at once (figure 1).

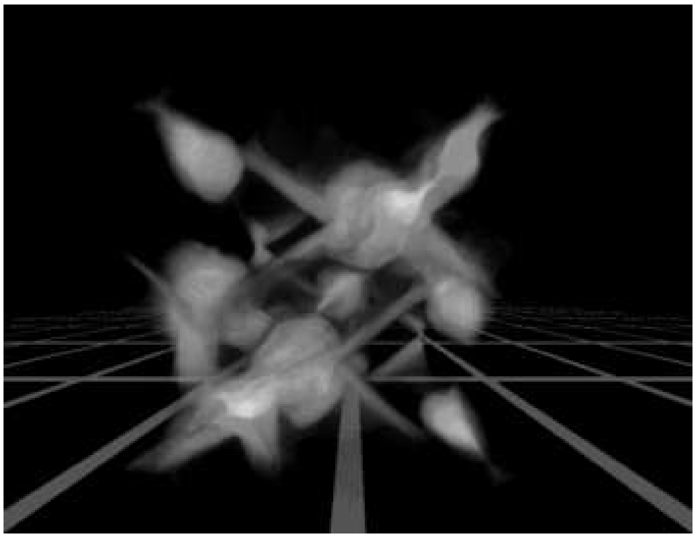

Figure 1: A volume rendering from the CAVE application, *astro*, which depicts interstellar gas collapsing to form proto-stars.

This is one reason it is worth doing in a virtual environment, where the display medium is, in a sense, also volumetric. One way volume rendering can be done is to cast rays from the center of projection, through each pixel of the image being computed, and into the data volume. The opacity, e.g., of the volume elements along each ray can be added up and the result written to the corresponding pixel. In general, volume rendering is a time-consuming technique.

Little volume rendering has been done in virtual environments until recently, because the time it has taken to render a single time step has been too great. However, on the Silicon Graphics, Inc. (SGI), ONYX and Indigo 2 Impact computers, 3D texture mapping[3, 1], done in hardware, can be used to do volume rendering in real time. In 3D texture mapping, the texture is a 3D image—a volumetric dataset—such as those described above. Here, if a polygon intersects the volume, its pixels are texture mapped with the values of the volume that lie in the plane of intersection.

Volume rendering is accomplished with 3D texture mapping as follows. The data volume defines a discrete scalar function over a rectangular interval of 3D space. Treating this function as a 3D texture, the texturing hardware trilinearly interpolates between the discrete values to reconstruct it as a continuous function. This new function is then sampled with a dense stack of regularly spaced slicing polygons perpendicular to the user's line of sight. Parts of the polygons, in general, extend beyond the faces of the data volume into empty space and are

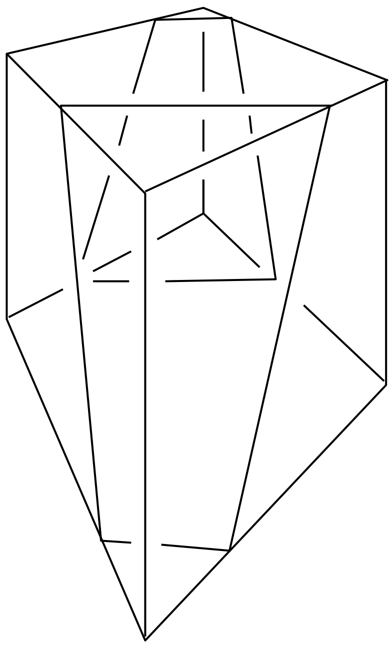

Figure 2: A data volume and two clipped slicing polygons at an arbitrary orientation with respect to the volume.

clipped away (figure 2). For all polygons in the stack, the texturing process is performed on the parts that remain: a given polygon's pixels take on the values of the data volume that lie in the slice of the volume that is coincident with the polygon. The translucence required for volume rendering is attained by the additional mapping of the data to alpha channel values. Via use of the alpha blending hardware, the slicing polygons are composited into the frame buffer during rendering [6]. The blending allows the entire data volume to contribute to the final picture. The effect is of a translucent block of substance of varying opacity. Volume rendering via 3D texture mapping is further described in [2, 7] and implemented in SGI's public domain *volren*‡ application.

2 *VREN* VOLUME RENDERING CLASS

VRen is an OpenGL-based C++ class that uses the above technique, and that

‡"Interactive Volume Rendering Using Advanced Graphics Architectures," *Online Technical Center white paper—URL: http://www.sgi.com/Technology/volume/VolumeRendering.html,* Silicon Graphics, Inc.

can be used from any C++ CAVE application to do volume rendering of 3D scalar volumes defined on a rectangular grid.

2.1 Limitations

VRen assumes that the calling application is using a shared memory class which has overloaded *new* and *delete* to handle shared memory. It requires an Impact running Irix 6.2, an Onyx with Irix 5.3 on which patch 918 has been installed, or an ONYX Infinite Reality. Only these configurations support 3D texture mapping extensions to the base OpenGL language.

The size of the dataset that can be visualized is limited by the amount of texture memory on the graphics machine being used and how fast subsets of the data can be swapped in and out of that memory (§2.3). The data volume resolution must also be 2^n in each dimension.

Another limitation is a pixel-fill bottleneck that is encountered if the user is immersed in the data. When this happens, if there are enough polygons in the slicing stack for good resolution, there are too many polygons the size of the display screen to be pixel-filled at a good frame rate. The frame rate can drop to about 0.5 frames per second or worse.

2.2 Features

The input data currently must be a single 3D raw (no header, unformatted) image passed to *VRen* as a 1D array (see §5). Any x, y and z resolutions that satisfy the above constraints (§2.1) are acceptable. The data may have any spatial extent, and may be of the types *float*, *unsigned short* or *unsigned character*. By way of a transformation matrix that the calling application must pass to *VRen*, the application developer has control over translation, rotation and scaling of the displayed volume as well as navigation transformations.

A feature called "bricking" actually allows one of the above constraints to be circumvented to some extent by allowing more texture data to be visualized than there is texture memory. Subsets of the entire dataset can be defined at the start of the application as separate textures, each of which can, in turn, be swapped from RAM into texture memory and rendered and composited into the frame buffer in one animation frame during the drawing process.

Parameters specifying the data resolution, spatial extent, element type, subsampling factor and an identifying string are passed to *VRen* in a parameter structure. The subsampling factor is not yet used; the identifying string may be used by the developer as a name for the data.

2.3 Implementation

A stack of from 100 to 250 slicing polygons is used. At least 100 are needed for good quality volume rendering. More produce even better visual results. Too many will result in a slower frame rate (see also §2.1). Currently the number of polygons is not adjustable (see §5).

Results are best if the slicing polygons are kept perpendicular to the user's line of sight. In a virtual environment, where the user can move in world space, the line of sight can have any orientation, and the polygons must be rotated to remain perpendicular to it. In fact, since the use of transparency requires that the polygons be drawn in the correct order, the normals of the polygons in the stack must always point toward the user. So, since the polygons can slice the texture volume at any orientation, the vertices of the clipped polygons must in general be recomputed for each animation frame.

VRen uses the viewing frustum clipping hardware to compute these vertices as follows. In a separate step that precedes the actual drawing of the scene each frame, drawing is turned off (the graphics hardware is put into *feedback buffer* mode), the rectangular solid of the texture volume is treated as an orthographic viewing frustum, and the slicing polygon stack is drawn. The graphics hardware clips the polygons against this frustum and stores the results in the feedback buffer. The vertices of the clipped polygons are then read from the feedback buffer, the graphics hardware is put into *draw* mode, and the vertices are used to draw the scene. This process is done for each "brick" of data (§2.2).

Input data values are mapped directly onto the range $[0, 255]$. (Currently, though several internal texture element formats are available, only unsigned character is used in *VRen*. See §5) Via a texture lookup table, high data values are mapped to high display intensities and opacities, and low data values are mapped to low display intensities and opacities. This mapping of data to color and transparency is currently the only one available (see §5).

3 RESULTS

VRen was tested in an Argonne application—*boiler*—that simulates a boiler that incinerates tires [4]. Volume rendering will soon be used in *boiler* to display rectangular volumes of temperature, of arbitrary shape, size and orientation, inside the incinerator. The temperature data will come from the IBM SP at Argonne. Later, vector fields will likely be volume rendered as well.

For the test, we simply added a few lines of code to *boiler* to create a *VRen* object, and to read in and display data from *astro*. We call the resulting hybrid application *astroboiler*. Thus, *VRen* can be used from another C++ CAVE application (figure 3). *Boiler* uses Terry Franguiadakis' shared memory class with overloaded *new* and *delete*, so *VRen* simply makes use of these. *Astro*'s data consists of 64^3 1-byte elements. One hundred seventy-five slicing polygons were used.

We have been able to run *astroboiler* on a patch-918-equipped ONYX with Irix 5.3, an Impact running Irix 6.2, and an Infinite Reality machine also running Irix 6.2. (Because of texture mapping problems with the Impact itself, no

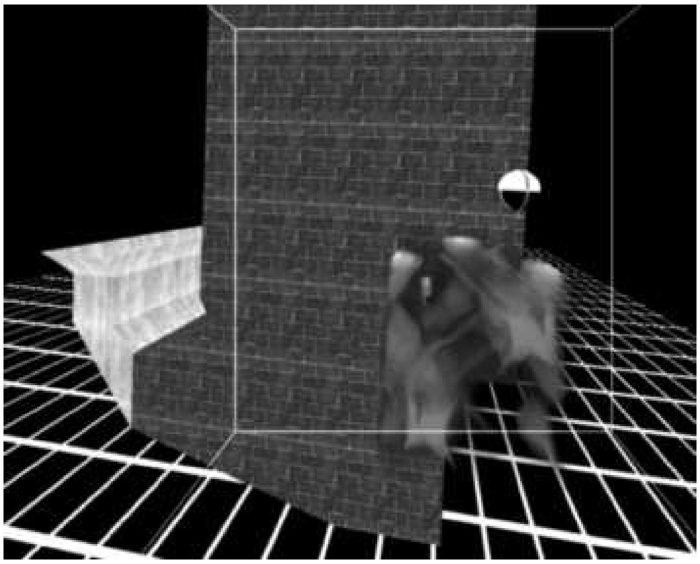

Figure 3: Data from *astro* volume rendered in *boiler* using *VRen*.

textures appeared on the polygons in the scene run on the Impact.)

4 PSEUDOCODE

```
#include "VRen.h"
#include "sharedm.h"

SharedMemArena *aSharedMemArena;
VRen *aVolume;
void *data;
float mtx[16];
ParmStruct parms;

main(int argc, char **argv) {

   InstantiateSharedMem();
   SharedMemOn();

   CAVEConfigure(&argc,argv,NULL);
   InitSharedData();
```

```
   CAVEInit();
   CAVEInitApplication(InitApplication, 0);
   CAVEFrameFunction(FrameApplication, 0);

   while(!CAVEgetbutton(CAVE_ESCKEY)) {
      while (!CAVEgetbutton(CAVE_ESCKEY)) {
         CalcFrame();
         CAVEDisplay(Draw, 0);
      }
   }
   CAVEExit();

   return 0;
}

void InitSharedData() {
   int i;

   SetupParameters();
   AllocDataSpace();
   ReadInputFile();
   InitMtx();
   aVolume = new VRen(data, &parms);
}

void Draw() {
   ClearBuffers();
   //   Developer transformations
   aVolume->Draw();
}

//   Perform updates in main process
void CalcFrame() {
   aVolume->CalcFrame();
}

//   Initialize graphics
void InitApplication() {
   aVolume->InitApplication();
}

//   Pass updated matrix to drawing processes
void FrameApplication() {
   aVolume->FrameApplication(mtx);
```

```
}
```

5 FURTHER WORK

VRen currently provides minimal volume rendering. In the near future, because of demand for more versatility and control by other application developers and users, support for such as the following will be added to *VRen*.

- 3D vector fields
- Multiple arbitrary clipping planes
- Developer control of color and transparency lookup tables, and the mapping of the data to these tables
- Support for time series
- Input data that can be either a set of 2d image files or a single 3D image file.
- Parameters specifying the number of slicing polygons
- If necessary, more data element types
- Subsampling
- Direct support of more internal texture formats

There are no plans to support parameters or data that come in the form of input files.

6 ACKNOWLEDGEMENTS

Thanks to Daniel Heath for his help.

References

[1] Blinn, J., and Newell, M. "Texture and Reflection in Computer Generated Images". *Communications of the Association for Computing Machinery 19*, 10 (October 1976), 542–547.

[2] Cabral, B., Cam, N., and Foran, J. "Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware". In *Proceedings of the IEEE 1994 Symposium on Volume Visualization* (Washington, DC, October 1994), pp. 91–94.

[3] Catmull, E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. PhD thesis, University of Utah, 1974.

[4] Diachin, D., Freitag, L., Heath, D., Herzog, J., Michels, W., and Plassman, P. "Interactive Simulation and Analysis of Emission Reduction Systems for Commercial Boilers". In *Proceedings of the 1996 Simulation Multiconference* (New Orleans, LA, April 1996), pp. 352–357.

[5] Drebin, R., Carpenter, L., and Hanrahan, P. "Volume Rendering". In *Proceedings of SIGGRAPH '88* (Atlanta, GA, August 1988), pp. 65–74.

[6] Porter, T., and Duff, T. "Compositing Digital Images". In *Proceedings of SIGGRAPH '84* (August 1984), pp. 253–259.

[7] Teschner, M. "Texture Mapping". *IRIS Universe 29* (Fall 1994), 8–11.