Daylighting Analysis in the BigHorn Home Improvement Center

Sara Hastings

DOE ERULF Program

Pomona College

National Renewable Energy Laboratory

Golden, CO

August 3, 2000

Prepared in partial fulfillment of the requirements of the DOE ERULF Program under the direction of Paul Torcellini in the Center for Buildings and Thermal Systems at the National Renewable Energy Laboratory.

Participant: _____

Research Advisor: _____

Table of Contents

Abstract

Daylighting is the most important element in creating a low-energy building in many climates. Reducing lighting loads decreases the electrical energy required for operating lighting systems and reduces building cooling loads. However, many commercial building owners are skeptical about daylighting strategies because of the perceived risk associated with what is considered a "new" concept. Also, architects and engineers find it difficult to optimize the daylighting system so as to minimize the lighting energy costs. This reluctance is especially true in the retail-building sector where "bright" lighting is equated to meeting product sales goals.

This paper describes the daylighting and lighting control design in the hardware store portion of the BigHorn Home Improvement Center in Silverthorne, Colorado. Energy consultants optimized the daylighting design through detailed modeling using an hourly building energy simulation tool.

The post construction study analyzed the daylighting effectiveness in the building space. Electric circuit lighting was optimized using the downhill simplex method and results were used to program the energy management system to control the lights. Lighting load energy savings were calculated and compared to the as-built simulation predictions. Recommendations for changes in the BigHorn building and future daylighting designs were made.

Research Category
ERULF

School Author Attends: Pomona College
DOE National Laboratory Attended: National Renewable Energy Laboratory
Mentor's Name: Paul A. Torcellini
Phone: 303.384.7528
E-mail Address: **Paul_Torcellini@nrel.gov**

Presenter's Name: Sara Hastings
Mailing Address: 2322 Westernesse Rd
City/State/Zip: Davis, CA 95616
Phone: 530.756.7474
E-mail Address: **Shastings@pomona.edu**

Is this being submitted for publication: YES NO
DOE Program: ERULF

Introduction

One of the most important elements of low-energy building designs is daylighting. Designing buildings to increase natural lighting decreases electrical lighting loads, which leads to a reduction in building cooling loads. However, in order to realize this reduction, electric lights in the buildings must be turned off when there is sufficient daylight. These reductions are greatest when the electric lights are controlled by an energy management system (EMS) that responds to changing light levels and controls the electric lights dynamically.

The Bighorn Home Improvement Center was designed as a participant in the National Renewable Energy Laboratory's (NREL) High-performance Building's Research Project. Funded by the U.S. Department of Energy (DOE), the purpose of High-performance Building's Research Project is to design, construct, and evaluate (through monitoring) buildings that exhibit exemplary energy performance. The owner was committed to integrating aggressive daylighting and other efficiency features into the building design. In addition, the owner believes the building design will reduce the impact on the environment and also improve sales.

The design followed a nine-step energy design process. The process developed by the High-Performance Research Project uses building energy simulations and economic analysis from pre-design through construction (1). This process resulted in a building that uses daylighting, automated daylighting controls, transpired solar collector for heated ventilation air, radiant floor heating and a building-integrated photovoltaic (PV) system. Full building energy modeling, using the hourly building energy simulation

tool DOE-2 (2), was performed as a part of the design process. It was estimated that the hardware store and warehouse would use 38% of the energy consumed in a comparable energy code-compliant retail building (3).

Building

The Bighorn Center is located in Silverthorne, Colorado, at an elevation of 2700 m (8858 ft) (Figure 1). The hardware store and warehouse building was completed in April 2000. The building houses a 1115-m^2 (12,000-ft^2) hardware store, 465-m^2 (5000-ft^2) office area, and a 2044-m^2 (22,000-ft^2) buildings materials warehouse. The architect for the project was Michael Schultz at Marketplace Architects, and NREL provided analysis assistance for the overall energy concept development.

Daylighting Strategies

Several different techniques were used to increase daylighting in the hardware store. The goal was to maximize daylighting potential while balancing the heating and cooling requirements. It is estimated that daylighting will reduce the lighting load by 79% compared to the base-case model (3). North-facing clerestory windows (Figure 2) as well as dormer windows along the north edge of the store bring diffuse daylight into the retail area (Figure 3). South-facing clerestory windows with engineered overhangs provide the majority of the lighting for the space (Figure 4). East- and west- facing windows provide a limited amount of direct-beam sunlight to the front and back of the building. Limited west-facing windows provide daylighting to the contractor service are and a view of the warehouse yard. A borrow window (interior window) brings daylight from the warehouse to the adjacent retail aisles (Figure 5).

Distribution of daylighting is improved by painting the ceilings and walls white. The floor tiles in the retail space are also white. Bright surfaces reflect the daylighting within the space, making the area brighter.

Electric Lighting

The electric lighting fixtures in the retail space each contains eight 26-W, compact fluorescent lamps within a domed glass shade (Figure 6). Pairs of lamps are controlled together (four pairs per fixture), to allow four levels of lighting from each fixture. The multi-stepped lighting control provides similar flexibly as continuous dimming control at a lower capital cost. The installed lighting density is 13.0 W/m^2 (1.2 W/ft^2) (3).

The light fixture placement complements the daylighting design – fixtures are located in rows parallel to the clerestory. The lamps are controlled by the energy management system (EMS), which also controls other building mechanical and electrical systems. Required lighting levels are based on store operation schedules. An override is available for off-hours occupancy. Motion sensors control the lighting in the interior offices, employee break room and restrooms.

Post Construction Daylighting Study

A post occupancy study of the hardware store and warehouse building was initiated in the summer of 2000 to evaluate the effectiveness of the daylighting strategies and design the lighting control program. Measurements of interior and exterior horizontal illumination were taken for a week around the summer solstice. Electric lighting use was also monitored during that period.

Monitoring Protocol

The retail lighting space was monitored from June 22 to June 28, 2000 for the performance analysis. Illumination measurements were collected by an automated data system. Thirteen LiCor Photometer cells, model LI-210SA, were connected to a Campbell Scientific, Inc. CR10X datalogger for the illumination readings. All of the photometers were calibrated at NREL prior to use. The datalogger read the interior and exterior photometers at 10-second intervals and recorded 15-minute average data.

Twelve photometer cells were placed on the top of the shelves in the retail area (Figure 7). An additional photometer was placed on the west end of the roof. The employees were informed of the study and were asked not to obstruct or move the photometers during the measurement period. Measurements with a handheld photometer verified the cells were wired correctly and not obstructed.

During this period the electric lights were occupant controlled, this study information was needed before the EMS could be programmed to control the lights. Measurements of the electrical lighting capacity were conducted the evening of June 25. At 9:00 p.m., after the sun had gone down and there was no natural light in the store, each lighting circuit was turned on and illumination levels at each of the 12 sensors was recorded. This was repeated for each lighting circuit, to establish the electric lighting contributions.

Summer Solstice Performance

Figures 8 and 9 plot the total illumination levels for a typical summer day at the sensor positions shown in Figure 7. The illumination levels track closely with the exterior

illumination level throughout the day. The breaks in the data for the outside sensor occur when outside illumination levels exceed the sensor range.

The daylighting component to the indoor lighting is calculated by subtracting the contributions of the electrical lights from the total illumination levels. These results, the daylighting at each of the 12 sensors for this day, are plotted in Figures 10 and 11. The illumination graphs during this period show a horizontal daylight illumination within the building of 100-300 lux.

Daylighting Design Assessment

In the morning hours, the most significant daylight contribution is from the east windows next to and above the glass entry doors (visible in Figure 1). Direct sunlight washed across the light-colored floor and reflected to the underside of the clerestory. This condition produced a high direct glare problem for customers walking toward the entry doors along the clerestory space. A louver or lightshelf system on the high east glass window would more efficiently bounce the light to the ceiling without disturbing the occupants. The deep overhang of the covered walk (visible in Figure 1) effectively protects the ground level windows from direct glare problems during most of the year. Low winter sun angles may enter briefly in the early morning hours (Figure 12).

Lighting from the central clerestory is evenly diffused in the space. The dark green standing seam roofing together with a cedar colored stained wood soffit results in a low level of daylight bounced onto the clerestory ceiling. All of the windows along the north clerestory wall have bug screens and reduced glazing area due to the electric window operators. The south windows have clear double-pane low-E glazing without operators or screens. White soffits, fewer screen and larger windows units would increase

the daylighting effectiveness by providing larger glazed area with fewer obstructions and more supportive reflecting surfaces.

The dark underside of the soffits were visible from the retail floor. Figure 4 shows that the south clerestory wall surface is nicely illuminated from the north windows. The deep steel beam parallel to the clerestory, however, blocks some of this light from washing down the sloped ceiling. Extending the sloped ceiling to the bottom of the beam would allow the light to spread down the ceiling

The borrow light between the retail and warehouse spaces brings welcomed daylight into an otherwise dark southwest corner of the retail floor (Figure 5). The success of this feature can be credited to the high daylight illumination levels under the translucent skylight in the warehouse.

Three dormers were built into the ceiling along the north wall. Two of them are in the retail space (Figure 3) while the other is in the shop. The retail dormers are effective in raising the illumination levels in the adjacent aisles. While the aisle in the northwest corner of the store only receive between 20 and 35 lux, near the dormers the illumination increases to 130 to 200 lux. Because of the orientation of the aisles and the narrow width of the dormers the benefits are limited to one or two aisles. They do help balance the illumination at the lower north wall.

Lighting Control Plan

From the monitoring data, a correlation between outside light and inside light is developed. The relationship, defined as outside light divided by inside light, is a daylighting coefficient. Because the outside sensor was placed in the sun, the coefficient calculated is much smaller than a traditional daylighting coefficient. The modified

daylighting coefficient for each sensor zone is given in Table 1. The coefficient changes throughout the day as the position of the sun affects the amount of light that reaches each inside zone.

The lighting design in the BigHorn Center hardware store and warehouse is approached with a focus on maximizing energy savings from daylighting. The floor plan is divided into lighting zones and fixture placement is designed to maintain set light levels in each zone (Figure 7). The fixtures are controlled in lighting circuits, with eight groups of fixtures over the retail space. A mathematical method was developed to determine the most efficient lighting plan. Using the modified daylighting coefficient and a single sensor placed on the roof of the building, the EMS determines the daylighting levels at each zone in real-time. Electric lighting requirements are determined by taking the difference between the scheduled lighting requirement and the daylighting available. The electric light required at each zone, y, given by Equation 1,

$$y = \lambda - \beta * \sigma, \qquad \text{Equation 1}$$

where λ is the light desired at the zone, β is the outside light level and σ is the modified daylighting coefficient.

To optimize the lighting the combination of lights that provides only the amount of electrical light required in each zone should be used. Each of the seven lighting groups provides different amounts of illumination to each of the twelve lighting zones. The optimization problem can be translated to a system of linear equations. The electrical lighting required, the *m* x 1 matrix **x**, where *m* is the number of lighting circuits in the building, is determined by Equation 2,

$$\mathbf{Ax} = \mathbf{y}, \qquad \text{Equation 2}$$

where **A** is a *n* x *m* matrix that represents the contribution of each light to each lighting zone, and **y** is a *n* x 1 matrix, the total electrical light required in each zone.

Because the building has more lighting zones than lighting circuits ($n > m$), Equation 2 is an over-determined system of linear equations. The equations are inconsistent and thus no solution exists. In this case an approximate solution is desired. An *n* x 1 error vector **e** is introduced,

$$\mathbf{e} = \mathbf{y} - \mathbf{Ax}. \qquad \text{Equation 3}$$

Here **e** is the excess illumination in each sensor zone. To achieve the maximum possible energy savings the excess illumination, **e,** must be minimized. The least-squares approximate solution is used to minimize the norm of **e**,

$$\|\mathbf{e}\|^2 = (\mathbf{y} - \mathbf{Ax})^{\mathbf{T}} (\mathbf{y} - \mathbf{Ax}). \qquad \text{Equation 4}$$

The minimization method used is the Downhill Simplex method. Computer code to solve this equation was written in C and is a modification of the "amoeba" method from Numerical Recipes (4) (full code text contained in Appendix A). The code determines the most efficient lighting plan and calculates the excess illumination based on the desired electrical lighting levels. Because the electric lights are multi-stepped, the lighting control is an approximation based on the results of the Downhill Simplex minimization.

Energy Savings Assessment

Using the downhill simplex method computer code, a lighting plan for the BigHorn Home Improvement Center hardware store and warehouse was developed (Table 2). The lighting electrical use was calculated for the building using this lighting plan and compared to the electrical use at the level of the installed capacity. The

calculated savings in lighting electrical use per day in the EMS-controlled building is 56% (Figure 13)

These savings are less than the predicted 79% reduction. The lack of precision in control results in the need to over-illuminate the building. Rather than a switching scheme that is based solely upon the rows of luminaries parallel to the clerestory, this may be modified to respond to the daylight provided in the dormers, east windows and perhaps even the borrow light. For example, the luminaries closest to the window wall in the dormers could be off or minimally switched during daylight hours. Likewise the number of lamps on in the row of luminaries near the east window wall could also be reduced. If the luminaries in the zones of highest illumination were wired together, they could be switched off while neighboring lamps remained on to provide necessary illumination in areas with less daylighting. Wiring could have been designed pre-construction based on predicted daylighting levels.

With all of the electric lights turned off in the retail space, certain display areas need supplemental illumination. Both the northwest corner and the southwest bank of display shelves are inadequately illuminated. It may be necessary to add some strip fluorescent or compact fluorescent in these areas (shown in Figure 14) when the rest of the store is relying on daylight.

A similar monitoring study must be conducted in the winter, spring, and fall seasons to develop a complete yearlong lighting plan for the building. The modified daylighting coefficient is expected to change in the winter, spring and fall seasons as a result of the change of position of the sun. The new coefficients can be used in the existing computer code to determine an annual lighting plan.

Summary

The next step in improving the lighting savings is to install the recommended fluorescent lighting in the dark areas throughout the building. Alternatively, the circuits could be rewired so that the areas with dark spots are on the same circuit and can be illuminated together. Once this is completed, the lighting plan can be recalculated using the Downhill Simplex method. This improved lighting plan can then be used to program the EMS at Bighorn.

The BigHorn Center is one of the first examples in the United States of truly integrated daylighting in a retail space. The design team used a whole-building design approach to integrate improved daylighting opportunities, and automated light controls to maximize energy sayings. The overall design is successful in meeting the owner's goal of minimizing the environmental impacts of the new building. The daylighting design results in lighting energy savings of 56%. The overall energy savings are 38%.

Acknowledgments

I would like to thank the Department of Energy, and the National Renewable Energy Laboratory for giving me the opportunity to participate in the Energy Research Undergraduate Laboratory Fellowship program.

Special thanks to my mentor, Dr. Paul Torcellini, visiting professor, Dr. Thomas Wood, and NREL Senior Engineer Shelia Hayter for all their help and guidance this summer. Also, thanks to Linda Lung, Science Education Specialist and ERULF manager.

Support for the energy design of the BigHorn Center, Phase III was provided by the NREL's High-Performance Buildings Research Project and Photovoltaics for

Buildings Task. The U.S. Department of Energy (DOE) provides project funding for these two activities from the Office of Building Technology, State and Community Programs, and the Office of Power Technologies, respectively.

Special thanks to Don and Betsy Sather the owner of the BigHorn Center, for following through with their vision for a sustainable building project. Also, thanks to the City of Silverthorne, Colorado, for working closely with the building owner to make this project a reality.

Marketplace Architects in Dillon, Colorado completed the architectural design. Tolin Mechanical of Silverthorne, Colorado, was the controls contractor. The building-integrated PV design was completed by Burdick Technologies Unlimited of Lakewood, Colorado, and the SolarWall system was designed by Foltz Engineering in Estes Park, Colorado.

Literature Cited

(1) Torcellini, P.A., S.J. Hayter, R. Judkoff, "Low-Energy Building Design -- the Process and a Case Study," ASHRAE Transactions, V 105, Part 2, pp. 802 - 810, 1999

(2) Lawrence Berkeley National Laboratory and Hirsch and Associates, *DOE-2 version 2.1e*, 1994

(3) Hayter, S.J., P.A. Torcellini, M. Eastment, R. Judkoff, "Using the Whole-Building Design Approach to Incorporate Daylighting into a Retail Space," Proceedings for the 2000 ACEEE Summer Study on Energy Efficiency in Buildings, to be published August 2000.

(4) Press, William H.; Teukolsky, Saul A.; Vetterling, Willian T.; Flannery, Brian P, Numerical Recipes in C. Cambridge University Press; 1992.

Figure 1. BigHorn Home Improvement Center Front Façade (East Elevation)

Figure 2. North Clerestory Windows

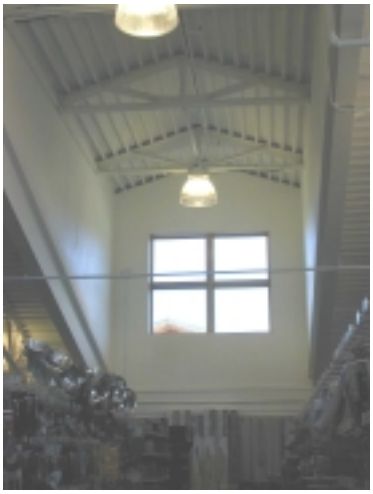

Figure 3. Dormer Window

Figure 4. South Clerestory Windows

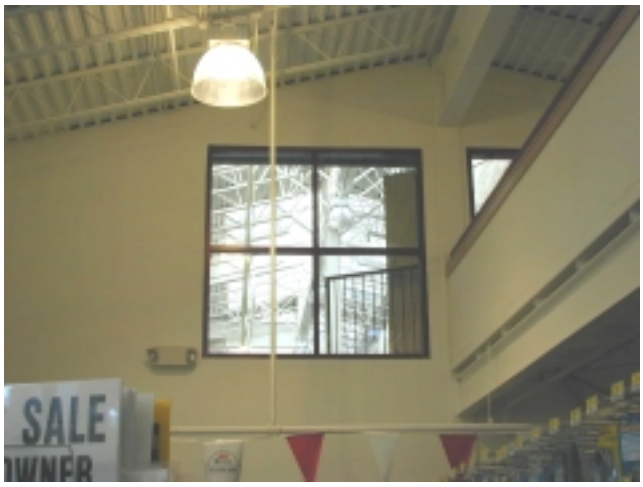

Figure 5. Borrow Light

Figure 6. Sport lamp domed shade fixture

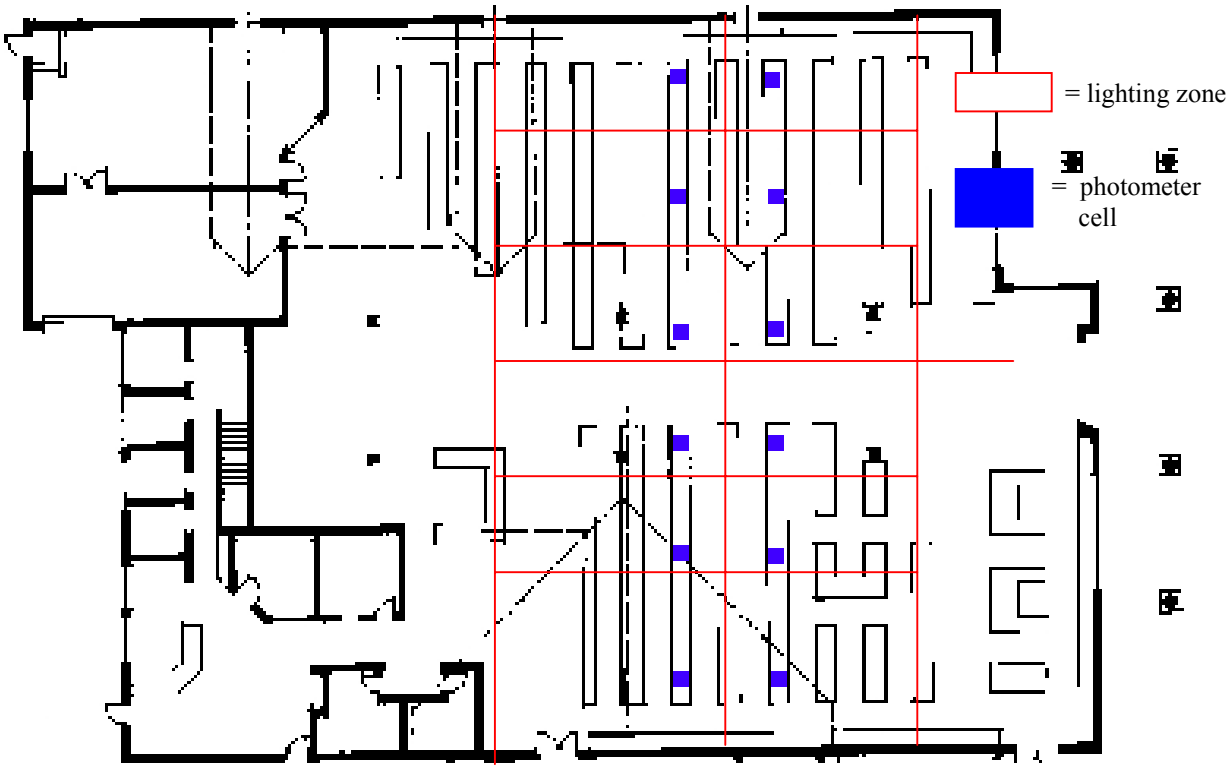

Figure 7. Sensor locations

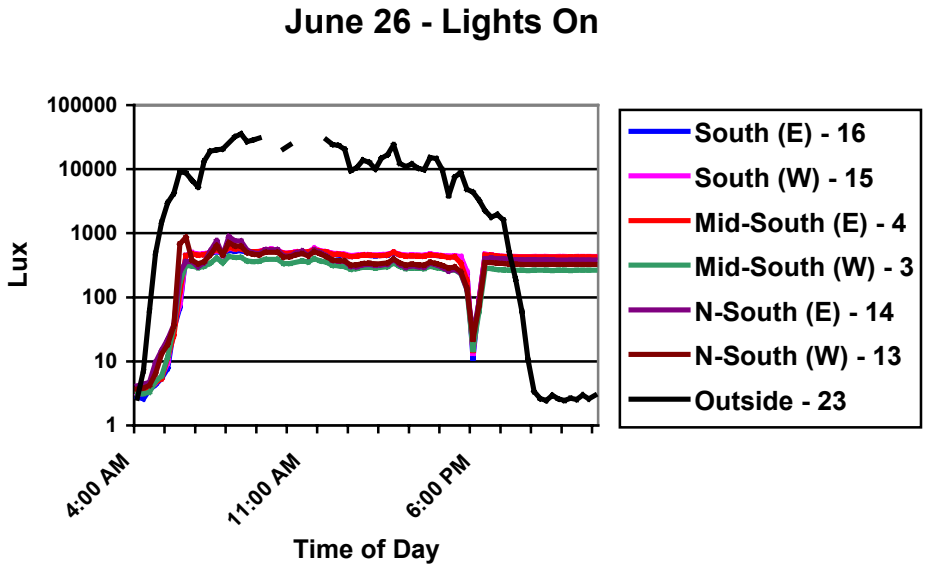

Figure 8. Lights on illumination at South sensors

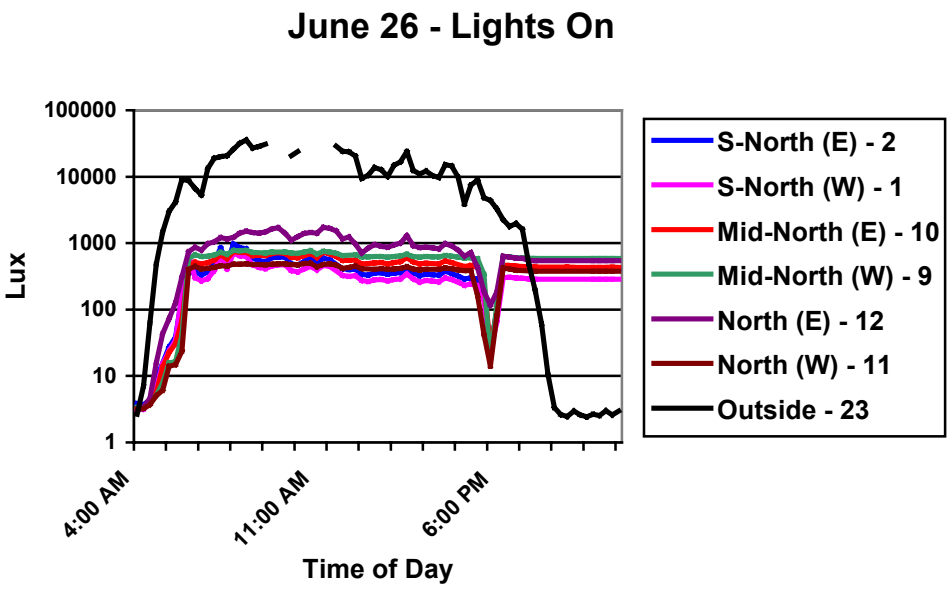

Figure 9. Lights on illumination at North sensors

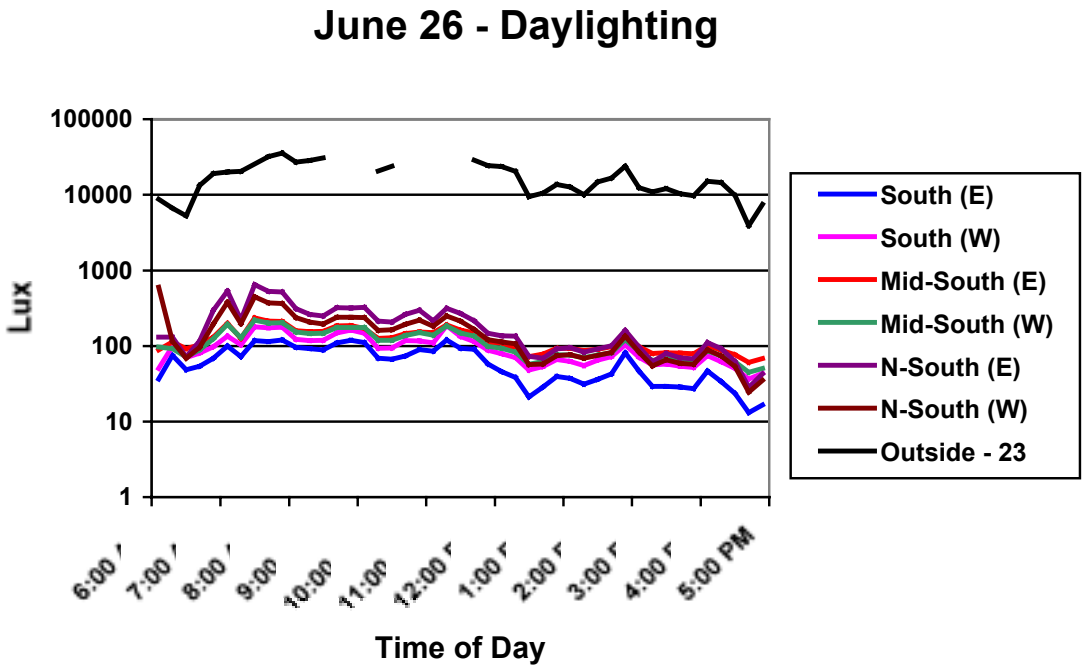

Figure 10. Daylighting only illumination at South sensors

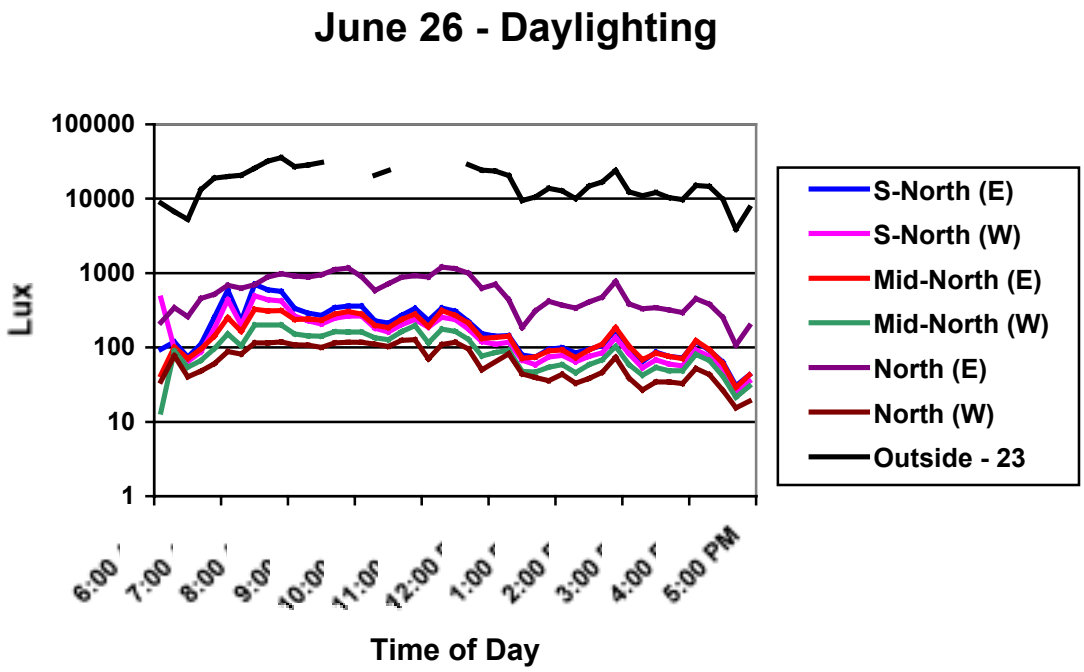

Figure 11. Daylighting only illumination at North sensors

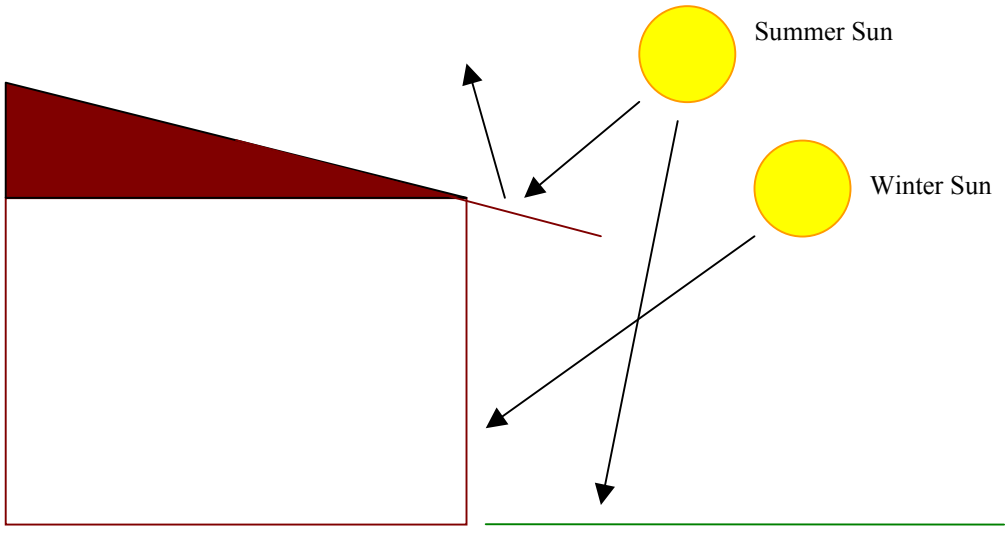

Figure 12. Low overhangs prevent summer sun from entering building while allowing lower, winter sun in

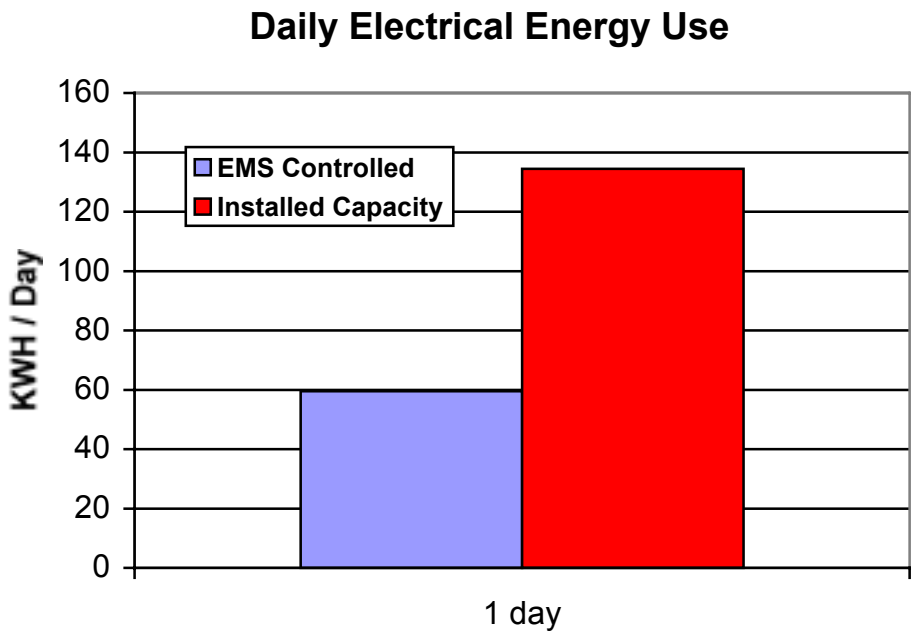

Figure 13. Lighting energy use for the EMS controlled building versus the installed capacity.

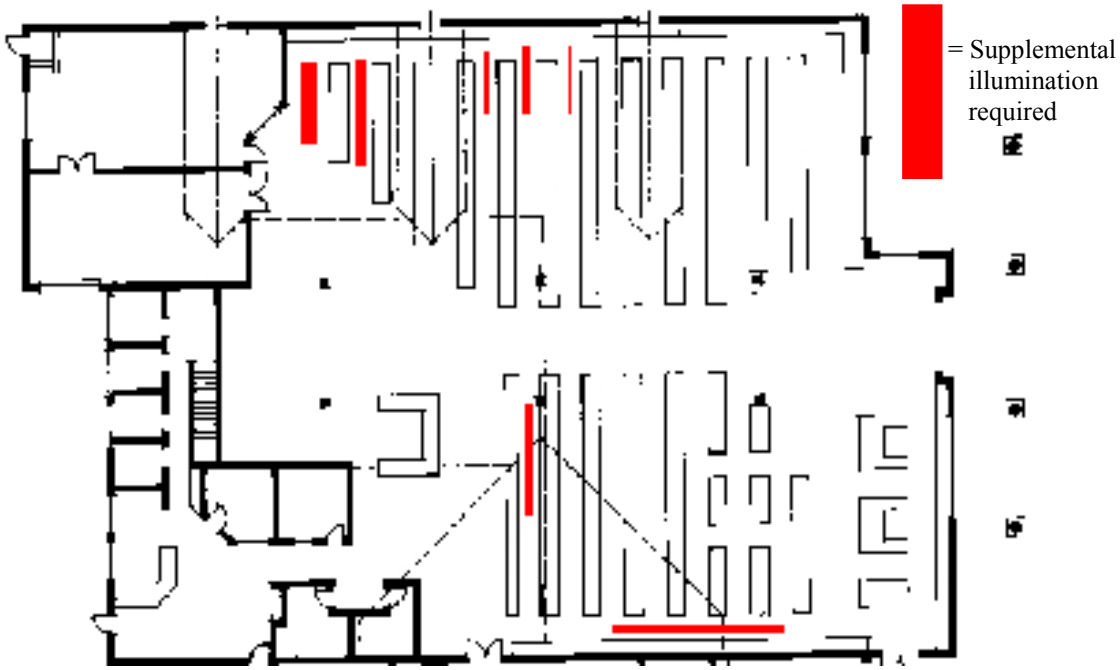

Figure 14. Display areas requiring supplemental illumination

Index of Figures

All photographs and diagrams: Thomas Wood 6/00

Table 1: Modified daylighting coefficients

	Zone											
Time of Day		16	15	4	3	14	13	2	1	10	9	12	11
	6am-8am	.0035	.005	.006	.0055	.008	.007	.01	.008	.007	.005	.027	.003
	9am - 12pm	.002	.0035	.004	.003	.005	.005	.006	.0045	.0055	.0035	.022	.0022
	1pm - 3pm	.002	.003	.004	.004	.0055	.0045	.006	.0045	.0055	.0035	.025	.002
	4pm - 6pm	.002	.0032	.004	.004	.005	.0045	.005	.0045	.0055	.003	.024	.002

Table 2: Lighting Plan

Outside illumination	A	B	C	D	E	F	G
0	5	6	3	8	7	2	4
1000	5	6	3	8	7	2	4
2000	5	6	3	8	7	2	3
4000	5	6	2	8	7	2	3
8000	5	6	2	8	6	1	3
10000	5	6	2	6	6	1	2
14000	5	5	2	6	6	1	2
18000	5	5	2	5	5	1	2
20000	5	5	2	5	5	2	1
24000	5	5	2	4	4	2	1
28000	4	4	2	4	4	2	1
30000	4	4	2	3	3	2	1
32000	4	4	2	3	3	2	1
35000	4	4	2	3	3	2	1

Appendix A

Complied in Microsoft Visual C++ v 6.0

```
#include "nr.h"
#include "nrutil.h"

#define MP 8
#define NP 7
#define NC 12
#define FTOL .000001

float func(float x[])
{

	/* units are lux */

	return (float)(pow(236-39.5*x[1]-10.55*x[2]-5.25*x[3]-3.0*x[4]-1.75*x[5]-1.325*x[6]-
1.325*x[7],2.0) + \
		pow(204-40.5*x[1]-6.95*x[2]-5.95*x[3]-3.0*x[4]-1.85*x[5]-.675*x[6]-1.375*x[7],2.0)
+ \
		pow(172-3.05*x[1]-41.55*x[2]-21.95*x[3]-3.0*x[4]-2.03*x[5]-1.2*x[6]-
1.225*x[7],2.0) + \
		pow(204-3.825*x[1]-8.9*x[2]-17.85*x[3]-3.0*x[4]-2.13*x[5]-1.275*x[6]-
1.275*x[7],2.0) + \
		pow(140-1.85*x[1]-4.9*x[2]-24.65*x[3]-3.0*x[4]-6.58*x[5]-2.45*x[6]-2.475*x[7],2.0)
+ \
		pow(156-1.8125*x[1]-4.8*x[2]-31.35*x[3]-3.0*x[4]-4.42*x[5]-2.4*x[6]-1.2*x[7],2.0)
+ \
		pow(140-.65*x[1]-2.6*x[2]-7.8*x[3]-3.0*x[4]-29.8*x[5]-3.9*x[6]-1.95*x[7],2.0) + \
		pow(156-.66*x[1]-2.65*x[2]-7.95*x[3]-3.0*x[4]-23*x[5]-3.825*x[6]-1.325*x[7],2.0) +
\
		pow(124-.9625*x[1]-2.55*x[2]-3.85*x[3]-3.0*x[4]-31.17*x[5]-32.175*x[6]-
7.725*x[7],2.0) + \
		pow(204-.6625*x[1]-2.7*x[2]-2.65*x[3]-3.0*x[4]-32.67*x[5]-53.5*x[6]-5.35*x[7],2.0)
+ \
		pow(0-1.2125*x[1]-2.45*x[2]-4.85*x[3]-3.0*x[4]-4.0*x[5]-8.475*x[6]-
64.825*x[7],2.0) + \
		pow(236-.6875*x[1]-2.75*x[2]-2.75*x[3]-3.0*x[4]-2.3*x[5]-4.85*x[6]-
51.925*x[7],2.0));
}

int main(void)
{
	int i,nfunc,j,ndim=NP;
	float *x,*y,**p;

	x=vector(1,NP);
	y=vector(1,MP);
	p=matrix(1,MP,1,NP);


   /* Enter best guess for circuts */

	x[1] = p[1][1] = 2;
	x[2] = p[1][2] = 2;
```

```
        x[3] = p[1][3] = 2;
        x[4] = p[1][4] = 2;
        x[5] = p[1][5] = 2;
        x[6] = p[1][6] = 2;
x[7] = p[1][7] = 2;
        y[1] = func(x);

        x[1] = p[2][1] = 3;
        x[2] = p[2][2] = 3;
        x[3] = p[2][3] = 7;
x[4] = p[2][4] = 3;
        x[5] = p[2][5] = 3;
        x[6] = p[2][6] = 7;
        x[7] = p[2][7] = 7;
        y[2] = func(x);

        x[1] = p[3][1] = 1;
        x[2] = p[3][2] = 1;
        x[3] = p[3][3] = 1;
        x[4] = p[3][4] = 1;
        x[5] = p[3][5] = 1;
        x[6] = p[3][6] = 1;
        x[7] = p[3][7] = 1;
        y[3] = func(x);

        x[1] = p[4][1] = 2;
        x[2] = p[4][2] = 3;
        x[3] = p[4][3] = 5;
        x[4] = p[4][4] = 8;
        x[5] = p[4][5] = 6;
        x[6] = p[4][6] = 4;
        x[7] = p[4][7] = 7;
        y[4] = func(x);

        x[1] = p[5][1] = 8;
        x[2] = p[5][2] = 8;
        x[3] = p[5][3] = 8;
        x[4] = p[5][4] = 8;
        x[5] = p[5][5] = 8;
        x[6] = p[5][6] = 8;
        x[7] = p[5][7] = 8;
        y[5] = func(x);

        x[1] = p[6][1] = 4;
        x[2] = p[6][2] = 4;
        x[3] = p[6][3] = 4;
        x[4] = p[6][4] = 4;
        x[5] = p[6][5] = 4;
        x[6] = p[6][6] = 4;
        x[7] = p[6][7] = 4;
        y[6] = func(x);

        x[1] = p[7][1] = 6;
        x[2] = p[7][2] = 6;
        x[3] = p[7][3] = 6;
        x[4] = p[7][4] = 6;
```

```
	x[5] = p[7][5] = 6;
	x[6] = p[7][6] = 6;
	x[7] = p[7][7] = 6;
	y[7] = func(x);

	x[1] = p[8][1] = 0;
	x[2] = p[8][2] = 0;
	x[3] = p[8][3] = 0;
	x[4] = p[8][4] = 0;
	x[5] = p[8][5] = 0;
	x[6] = p[8][6] = 0;
	x[7] = p[8][7] = 0;
	y[8] = func(x);

	amoeba(p,y,ndim,FTOL,func,&nfunc);
	printf("\nNumber of function evaluations: %3d\n",nfunc);
	printf("Vertices of final 3-d simplex and\n");
	printf("function values at the vertices:\n\n");
	printf("%2s %10s %12s %12s %12s %12s %12s %12s %12s\n\n",
		"i","A[i]","B[i]","C[i]","D[i]","E[i]","F[i]","G[i]","function");
	for (i=1;i<=MP;i++) {
		printf("%2d ",i);
		for (j=1;j<=NP;j++) printf("%12.6f ",p[i][j]);
		printf("%12.6f\n",y[i]);
	}
	free_matrix(p,1,MP,1,NP);
	free_vector(y,1,MP);
	free_vector(x,1,NP);
	return 0;
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software .j,. */
/*	Updated 2000 Sara Hastings	*/

#include <math.h>
#define NRANSI
#include "nrutil.h"
#define NMAX 50000000
#define GET_PSUM \
					for (j=1;j<=ndim;j++) {\
					for (sum=0.0,i=1;i<=mpts;i++) sum += p[i][j];\
					psum[j]=sum;}
#define SWAP(a,b) {swap=(a);(a)=(b);(b)=swap;}

void amoeba(float **p, float y[], int ndim, float ftol,
	float (*funk)(float []), int *nfunk)
{
	float amotry(float **p, float y[], float psum[], int ndim,
		float (*funk)(float []), int ihi, float fac);
	int i,ihi,ilo,inhi,j,mpts=ndim+1;
	float rtol,sum,swap,ysave,ytry,*psum;

	psum=vector(1,ndim);
	*nfunk=0;
	GET_PSUM
	for (;;) {
```

```
                ilo=1;
                ihi = y[1]>y[2] ? (inhi=2,1) : (inhi=1,2);
                for (i=1;i<=mpts;i++) {
                        if (y[i] <= y[ilo]) ilo=i;
                        if (y[i] > y[ihi]) {
                                inhi=ihi;
                                ihi=i;
                        } else if (y[i] > y[inhi] && i != ihi) inhi=i;
                }
                        if (fabs((y[ihi])-(y[ilo])) == 0) {
                        rtol = .000001;
                        }
                        else {
                                rtol=(2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo])));
                        }
                printf("%d\n",ilo);
                printf("%f %f\n",y[ihi],y[ilo]);
                printf("%f\n",rtol);
                printf("%f\n",(fabs(y[ihi])+fabs(y[ilo])));
                printf("%f\n",fabs(y[ihi]-y[ilo]));
                printf("%f\n",(2.0*fabs(y[ihi]-y[ilo])/(fabs(y[ihi])+fabs(y[ilo]))));
                if (rtol < ftol) {
                        SWAP(y[1],y[ilo])
                        for (i=1;i<=ndim;i++) SWAP(p[1][i],p[ilo][i])
                        break;
                }
                if (*nfunk >= NMAX) nrerror("NMAX exceeded");
                *nfunk += 2;
                ytry=amotry(p,y,psum,ndim,funk,ihi,-1.0);
                if (ytry <= y[ilo]) {
                        ytry=amotry(p,y,psum,ndim,funk,ihi,2.0);
                        puts("working");
                }
                else if (ytry >= y[inhi]) {
                        ysave=y[ihi];
                        ytry=amotry(p,y,psum,ndim,funk,ihi,0.5);
                        if (ytry >= ysave) {
                                for (i=1;i<=mpts;i++) {
                                        printf("in mpts\n");
                                        if (i != ilo) {
                                                for (j=1;j<=ndim;j++)
                                                        p[i][j]=psum[j]=0.5*(p[i][j]+p[ilo][j]);
                                                y[i]=(*funk)(psum);
                                        }
                                }
                                *nfunk += ndim;
                                GET_PSUM
                        }
                } else --(*nfunk);
        }
        free_vector(psum,1,ndim);
}
#undef SWAP
#undef GET_PSUM
#undef NMAX
#undef NRANSI
```

```
/* (C) Copr. 1986-92 Numerical Recipes Software .j,. */

#define NRANSI
#include "nrutil.h"

float amotry(float **p, float y[], float psum[], int ndim,
	float (*funk)(float []), int ihi, float fac)
{
	int j;
	float fac1,fac2,ytry,*ptry;

	ptry=vector(1,ndim);
	fac1=(1.0-fac)/ndim;
	fac2=fac1-fac;
	for (j=1;j<=ndim;j++) ptry[j]=psum[j]*fac1-p[ihi][j]*fac2;
	ytry=(*funk)(ptry);
	if (ytry < y[ihi]) {
		y[ihi]=ytry;
		for (j=1;j<=ndim;j++) {
			psum[j] += ptry[j]-p[ihi][j];
			p[ihi][j]=ptry[j];
		}
	}
	free_vector(ptry,1,ndim);
	return ytry;
}
#undef NRANSI
/* (C) Copr. 1986-92 Numerical Recipes Software .j,. */

#if defined(__STDC__) || defined(ANSI) || defined(NRANSI) /* ANSI */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(char error_text[])
/* Numerical Recipes standard error handler */
{
	fprintf(stderr,"Numerical Recipes run-time error...\n");
	fprintf(stderr,"%s\n",error_text);
	fprintf(stderr,"...now exiting to system...\n");
	exit(1);
}

float *vector(long nl, long nh)
/* allocate a float vector with subscript range v[nl..nh] */
{
	float *v;

	v=(float *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(float)));
	if (!v) nrerror("allocation failure in vector()");
	return v-nl+NR_END;
}
```

```
int *ivector(long nl, long nh)
/* allocate an int vector with subscript range v[nl..nh] */
{
	int *v;

	v=(int *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(int)));
	if (!v) nrerror("allocation failure in ivector()");
	return v-nl+NR_END;
}

unsigned char *cvector(long nl, long nh)
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
	unsigned char *v;

	v=(unsigned char *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(unsigned char)));
	if (!v) nrerror("allocation failure in cvector()");
	return v-nl+NR_END;
}

unsigned long *lvector(long nl, long nh)
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
	unsigned long *v;

	v=(unsigned long *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(long)));
	if (!v) nrerror("allocation failure in lvector()");
	return v-nl+NR_END;
}

double *dvector(long nl, long nh)
/* allocate a double vector with subscript range v[nl..nh] */
{
	double *v;

	v=(double *)malloc((size_t) ((nh-nl+1+NR_END)*sizeof(double)));
	if (!v) nrerror("allocation failure in dvector()");
	return v-nl+NR_END;
}

float **matrix(long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix with subscript range m[nrl..nrh][ncl..nch] */
{
	long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
	float **m;

	/* allocate pointers to rows */
	m=(float **) malloc((size_t)((nrow+NR_END)*sizeof(float*)));
	if (!m) nrerror("allocation failure 1 in matrix()");
	m += NR_END;
	m -= nrl;

	/* allocate rows and set pointers to them */
	m[nrl]=(float *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float)));
	if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
	m[nrl] += NR_END;
```

```
	m[nrl] -= ncl;

	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}

double **dmatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
	long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
	double **m;

	/* allocate pointers to rows */
	m=(double **) malloc((size_t)((nrow+NR_END)*sizeof(double*)));
	if (!m) nrerror("allocation failure 1 in matrix()");
	m += NR_END;
	m -= nrl;

	/* allocate rows and set pointers to them */
	m[nrl]=(double *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(double)));
	if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
	m[nrl] += NR_END;
	m[nrl] -= ncl;

	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}

int **imatrix(long nrl, long nrh, long ncl, long nch)
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
	long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
	int **m;

	/* allocate pointers to rows */
	m=(int **) malloc((size_t)((nrow+NR_END)*sizeof(int*)));
	if (!m) nrerror("allocation failure 1 in matrix()");
	m += NR_END;
	m -= nrl;


	/* allocate rows and set pointers to them */
	m[nrl]=(int *) malloc((size_t)((nrow*ncol+NR_END)*sizeof(int)));
	if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
	m[nrl] += NR_END;
	m[nrl] -= ncl;

	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}
```

```
float **submatrix(float **a, long oldrl, long oldrh, long oldcl, long oldch,
	long newrl, long newcl)
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
	long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
	float **m;

	/* allocate array of pointers to rows */
	m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
	if (!m) nrerror("allocation failure in submatrix()");
	m += NR_END;
	m -= newrl;

	/* set pointers to rows */
	for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}

float **convert_matrix(float *a, long nrl, long nrh, long ncl, long nch)
/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
	long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
	float **m;

	/* allocate pointers to rows */
	m=(float **) malloc((size_t) ((nrow+NR_END)*sizeof(float*)));
	if (!m) nrerror("allocation failure in convert_matrix()");
	m += NR_END;
	m -= nrl;

	/* set pointers to rows */
	m[nrl]=a-ncl;
	for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
	/* return pointer to array of pointers to rows */
	return m;
}

float ***f3tensor(long nrl, long nrh, long ncl, long nch, long ndl, long ndh)
/* allocate a float 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */
{
	long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
	float ***t;

	/* allocate pointers to pointers to rows */
	t=(float ***) malloc((size_t)((nrow+NR_END)*sizeof(float**)));
	if (!t) nrerror("allocation failure 1 in f3tensor()");
	t += NR_END;
	t -= nrl;

	/* allocate pointers to rows and set pointers to them */
```

```
	t[nrl]=(float **) malloc((size_t)((nrow*ncol+NR_END)*sizeof(float*)));
	if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");
	t[nrl] += NR_END;
	t[nrl] -= ncl;

	/* allocate rows and set pointers to them */
	t[nrl][ncl]=(float *) malloc((size_t)((nrow*ncol*ndep+NR_END)*sizeof(float)));
	if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
	t[nrl][ncl] += NR_END;
	t[nrl][ncl] -= ndl;

	for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
	for(i=nrl+1;i<=nrh;i++) {
		t[i]=t[i-1]+ncol;
		t[i][ncl]=t[i-1][ncl]+ncol*ndep;
		for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
	}

	/* return pointer to array of pointers to rows */
	return t;
}

void free_vector(float *v, long nl, long nh)
/* free a float vector allocated with vector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_ivector(int *v, long nl, long nh)
/* free an int vector allocated with ivector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_cvector(unsigned char *v, long nl, long nh)
/* free an unsigned char vector allocated with cvector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_lvector(unsigned long *v, long nl, long nh)
/* free an unsigned long vector allocated with lvector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_dvector(double *v, long nl, long nh)
/* free a double vector allocated with dvector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_matrix(float **m, long nrl, long nrh, long ncl, long nch)
/* free a float matrix allocated by matrix() */
{
	free((FREE_ARG) (m[nrl]+ncl-NR_END));
```

```
	free((FREE_ARG) (m+nrl-NR_END));
}

void free_dmatrix(double **m, long nrl, long nrh, long ncl, long nch)
/* free a double matrix allocated by dmatrix() */
{
	free((FREE_ARG) (m[nrl]+ncl-NR_END));
	free((FREE_ARG) (m+nrl-NR_END));
}

void free_imatrix(int **m, long nrl, long nrh, long ncl, long nch)
/* free an int matrix allocated by imatrix() */
{
	free((FREE_ARG) (m[nrl]+ncl-NR_END));
	free((FREE_ARG) (m+nrl-NR_END));
}

void free_submatrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a submatrix allocated by submatrix() */
{
	free((FREE_ARG) (b+nrl-NR_END));
}

void free_convert_matrix(float **b, long nrl, long nrh, long ncl, long nch)
/* free a matrix allocated by convert_matrix() */
{
	free((FREE_ARG) (b+nrl-NR_END));
}

void free_f3tensor(float ***t, long nrl, long nrh, long ncl, long nch,
	long ndl, long ndh)
/* free a float f3tensor allocated by f3tensor() */
{
	free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
	free((FREE_ARG) (t[nrl]+ncl-NR_END));
	free((FREE_ARG) (t+nrl-NR_END));
}

#else /* ANSI */
/* traditional - K&R */

#include <stdio.h>
#define NR_END 1
#define FREE_ARG char*

void nrerror(error_text)
char error_text[];
/* Numerical Recipes standard error handler */
{
	void exit();

	fprintf(stderr,"Numerical Recipes run-time error...\n");
	fprintf(stderr,"%s\n",error_text);
	fprintf(stderr,"...now exiting to system...\n");
	exit(1);
}
```

```
float *vector(nl,nh)
long nh,nl;
/* allocate a float vector with subscript range v[nl..nh] */
{
	float *v;

	v=(float *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(float)));
	if (!v) nrerror("allocation failure in vector()");
	return v-nl+NR_END;
}

int *ivector(nl,nh)
long nh,nl;
/* allocate an int vector with subscript range v[nl..nh] */
{
	int *v;

	v=(int *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(int)));
	if (!v) nrerror("allocation failure in ivector()");
	return v-nl+NR_END;
}

unsigned char *cvector(nl,nh)
long nh,nl;
/* allocate an unsigned char vector with subscript range v[nl..nh] */
{
	unsigned char *v;

	v=(unsigned char *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(unsigned char)));
	if (!v) nrerror("allocation failure in cvector()");
	return v-nl+NR_END;
}

unsigned long *lvector(nl,nh)
long nh,nl;
/* allocate an unsigned long vector with subscript range v[nl..nh] */
{
	unsigned long *v;

	v=(unsigned long *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(long)));
	if (!v) nrerror("allocation failure in lvector()");
	return v-nl+NR_END;
}

double *dvector(nl,nh)
long nh,nl;
/* allocate a double vector with subscript range v[nl..nh] */
{
	double *v;

	v=(double *)malloc((unsigned int) ((nh-nl+1+NR_END)*sizeof(double)));
	if (!v) nrerror("allocation failure in dvector()");
	return v-nl+NR_END;
}
```

```
float **matrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a float matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        float **m;

        /* allocate pointers to rows */
        m=(float **) malloc((unsigned int)((nrow+NR_END)*sizeof(float*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
        m[nrl]=(float *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(float)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

double **dmatrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a double matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        double **m;

        /* allocate pointers to rows */
        m=(double **) malloc((unsigned int)((nrow+NR_END)*sizeof(double*)));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m += NR_END;
        m -= nrl;

        /* allocate rows and set pointers to them */
        m[nrl]=(double *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(double)));
        if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
        m[nrl] += NR_END;
        m[nrl] -= ncl;

        for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

        /* return pointer to array of pointers to rows */
        return m;
}

int **imatrix(nrl,nrh,ncl,nch)
long nch,ncl,nrh,nrl;
/* allocate a int matrix with subscript range m[nrl..nrh][ncl..nch] */
{
        long i, nrow=nrh-nrl+1,ncol=nch-ncl+1;
        int **m;
```

```
	/* allocate pointers to rows */
	m=(int **) malloc((unsigned int)((nrow+NR_END)*sizeof(int*)));
	if (!m) nrerror("allocation failure 1 in matrix()");
	m += NR_END;
	m -= nrl;


	/* allocate rows and set pointers to them */
	m[nrl]=(int *) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(int)));
	if (!m[nrl]) nrerror("allocation failure 2 in matrix()");
	m[nrl] += NR_END;
	m[nrl] -= ncl;

	for(i=nrl+1;i<=nrh;i++) m[i]=m[i-1]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}

float **submatrix(a,oldrl,oldrh,oldcl,oldch,newrl,newcl)
float **a;
long newcl,newrl,oldch,oldcl,oldrh,oldrl;
/* point a submatrix [newrl..][newcl..] to a[oldrl..oldrh][oldcl..oldch] */
{
	long i,j,nrow=oldrh-oldrl+1,ncol=oldcl-newcl;
	float **m;

	/* allocate array of pointers to rows */
	m=(float **) malloc((unsigned int) ((nrow+NR_END)*sizeof(float*)));
	if (!m) nrerror("allocation failure in submatrix()");
	m += NR_END;
	m -= newrl;

	/* set pointers to rows */
	for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+ncol;

	/* return pointer to array of pointers to rows */
	return m;
}

float **convert_matrix(a,nrl,nrh,ncl,nch)
float *a;
long nch,ncl,nrh,nrl;
/* allocate a float matrix m[nrl..nrh][ncl..nch] that points to the matrix
declared in the standard C manner as a[nrow][ncol], where nrow=nrh-nrl+1
and ncol=nch-ncl+1. The routine should be called with the address
&a[0][0] as the first argument. */
{
	long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1;
	float **m;

	/* allocate pointers to rows */
	m=(float **) malloc((unsigned int) ((nrow+NR_END)*sizeof(float*)));
	if (!m)    nrerror("allocation failure in convert_matrix()");
	m += NR_END;
```

```
	m -= nrl;

	/* set pointers to rows */
	m[nrl]=a-ncl;
	for(i=1,j=nrl+1;i<nrow;i++,j++) m[j]=m[j-1]+ncol;
	/* return pointer to array of pointers to rows */
	return m;
}

float ***f3tensor(nrl,nrh,ncl,nch,ndl,ndh)
long nch,ncl,ndh,ndl,nrh,nrl;
/* allocate a float 3tensor with range t[nrl..nrh][ncl..nch][ndl..ndh] */
{
	long i,j,nrow=nrh-nrl+1,ncol=nch-ncl+1,ndep=ndh-ndl+1;
	float ***t;

	/* allocate pointers to pointers to rows */
	t=(float ***) malloc((unsigned int)((nrow+NR_END)*sizeof(float**)));
	if (!t) nrerror("allocation failure 1 in f3tensor()");
	t += NR_END;
	t -= nrl;

	/* allocate pointers to rows and set pointers to them */
	t[nrl]=(float **) malloc((unsigned int)((nrow*ncol+NR_END)*sizeof(float*)));
	if (!t[nrl]) nrerror("allocation failure 2 in f3tensor()");
	t[nrl] += NR_END;
	t[nrl] -= ncl;

	/* allocate rows and set pointers to them */
	t[nrl][ncl]=(float *) malloc((unsigned int)((nrow*ncol*ndep+NR_END)*sizeof(float)));
	if (!t[nrl][ncl]) nrerror("allocation failure 3 in f3tensor()");
	t[nrl][ncl] += NR_END;
	t[nrl][ncl] -= ndl;

	for(j=ncl+1;j<=nch;j++) t[nrl][j]=t[nrl][j-1]+ndep;
	for(i=nrl+1;i<=nrh;i++) {
		t[i]=t[i-1]+ncol;
		t[i][ncl]=t[i-1][ncl]+ncol*ndep;
		for(j=ncl+1;j<=nch;j++) t[i][j]=t[i][j-1]+ndep;
	}

	/* return pointer to array of pointers to rows */
	return t;
}

void free_vector(v,nl,nh)
float *v;
long nh,nl;
/* free a float vector allocated with vector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_ivector(v,nl,nh)
int *v;
long nh,nl;
```

```
/* free an int vector allocated with ivector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_cvector(v,nl,nh)
long nh,nl;
unsigned char *v;
/* free an unsigned char vector allocated with cvector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_lvector(v,nl,nh)
long nh,nl;
unsigned long *v;
/* free an unsigned long vector allocated with lvector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_dvector(v,nl,nh)
double *v;
long nh,nl;
/* free a double vector allocated with dvector() */
{
	free((FREE_ARG) (v+nl-NR_END));
}

void free_matrix(m,nrl,nrh,ncl,nch)
float **m;
long nch,ncl,nrh,nrl;
/* free a float matrix allocated by matrix() */
{
	free((FREE_ARG) (m[nrl]+ncl-NR_END));
	free((FREE_ARG) (m+nrl-NR_END));
}

void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
long nch,ncl,nrh,nrl;
/* free a double matrix allocated by dmatrix() */
{
	free((FREE_ARG) (m[nrl]+ncl-NR_END));
	free((FREE_ARG) (m+nrl-NR_END));
}

void free_imatrix(m,nrl,nrh,ncl,nch)
int **m;
long nch,ncl,nrh,nrl;
/* free an int matrix allocated by imatrix() */
{
	free((FREE_ARG) (m[nrl]+ncl-NR_END));
	free((FREE_ARG) (m+nrl-NR_END));
}
```

```
void free_submatrix(b,nrl,nrh,ncl,nch)
float **b;
long nch,ncl,nrh,nrl;
/* free a submatrix allocated by submatrix() */
{
        free((FREE_ARG) (b+nrl-NR_END));
}

void free_convert_matrix(b,nrl,nrh,ncl,nch)
float **b;
long nch,ncl,nrh,nrl;
/* free a matrix allocated by convert_matrix() */
{
        free((FREE_ARG) (b+nrl-NR_END));
}

void free_f3tensor(t,nrl,nrh,ncl,nch,ndl,ndh)
float ***t;
long nch,ncl,ndh,ndl,nrh,nrl;
/* free a float f3tensor allocated by f3tensor() */
{
        free((FREE_ARG) (t[nrl][ncl]+ndl-NR_END));
        free((FREE_ARG) (t[nrl]+ncl-NR_END));
        free((FREE_ARG) (t+nrl-NR_END));
}

#endif /* ANSI */
```