SHADOW PATTERN
SIMULATOR

by

Roberto A. Brown

A thesis submitted in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in
Computer Science

Approved:

UTAH STATE UNIVERSITY
Logan, Utah

1988
ACKNOWLEDGMENTS

I wish to express sincere thanks and deep appreciation to Dr. Del Dyreson, my major professor, for his invaluable suggestions, comments and support throughout the course of this study. Similarly, I want to thank Dr. Robert Campbell and Dr. Ron Canfield, committee members.

Thanks go to Professor E. Gregory McPherson for the data supplied and his help during the modeling stages of the study.

To my wife and parents, for their continuous support and encouragement throughout my college career and always, goes the most profound appreciation.

Roberto A. Brown
## TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGMENTS</td>
<td></td>
<td>ii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td></td>
<td>v</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td></td>
<td>vi</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td></td>
<td>vii</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Chapter</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II</td>
<td>BASIC GEOMETRY</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Coordinate Systems</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Types of Geometric Models</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Point, Planes, Lines and Rays</td>
<td>5</td>
</tr>
<tr>
<td>III</td>
<td>THE SOLAR GEOMETRY</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>Solar Declination</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Time</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>Longitude of the Sun</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Local Azimuth and Elevation</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Sunrise and Sunset</td>
<td>12</td>
</tr>
<tr>
<td>IV</td>
<td>GEOMETRY OF THE TREE’S SHADE</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td>Program Overview</td>
<td>15</td>
</tr>
<tr>
<td></td>
<td>Calculating Shadows on Normal Surfaces</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Calculating Shadows on Non-Normal Surfaces</td>
<td>31</td>
</tr>
<tr>
<td>V</td>
<td>VALIDATION OF THE PROGRAM</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Description of Scale Model</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Description of the Validation Procedure</td>
<td>32</td>
</tr>
<tr>
<td>VI</td>
<td>CONCLUSIONS AND RECOMMENDATIONS</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>Model Limitations</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Alternative Method</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Recommendations for Future Study</td>
<td>38</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>(a) The Celestial Sphere; (b) A Spherical Triangle from the Celestial Sphere Defining the Zenith Distance and the Azimuthal Angle.</td>
<td>11</td>
</tr>
<tr>
<td>2.</td>
<td>Once the Displacement Angle is Entered for Structures Oriented other than East (above), SPS Normalizes All Surfaces (below).</td>
<td>16</td>
</tr>
<tr>
<td>3.</td>
<td>Program Diagram</td>
<td>17</td>
</tr>
<tr>
<td>4.</td>
<td>SPS Graphic Output: Shadow Pattern from a Parabolic Tree Canopy.</td>
<td>20</td>
</tr>
<tr>
<td>5.</td>
<td>Sections and Plan Views of the Three Shapes Show a Shadowed Section.</td>
<td>22</td>
</tr>
<tr>
<td>6.</td>
<td>There is no Distortion of the Shadow and the Bottom is Flat when Sun’s Rays are Normal to the Tree Crown and the Receiving Plane.</td>
<td>27</td>
</tr>
<tr>
<td>7.</td>
<td>The Shadow Pattern is Distorted and the Bottom Rounded when Incidence Angles are Not Normal.</td>
<td>28</td>
</tr>
<tr>
<td>8.</td>
<td>Shadow Distortion on Heliodon.</td>
<td>34</td>
</tr>
</tbody>
</table>
LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Percent Surface Shaded by Heliodon Model and Calculated by the Program.</td>
<td>35</td>
</tr>
</tbody>
</table>
ABSTRACT

Shadow Pattern Simulator

by

Roberto A. Brown, Master of Science
Utah State University, 1988

Major Professor: Dr. Del Dyreson
Department: Computer Science

This paper describes an interactive computer program that can be used as a design tool in assessing the shading effects of trees in relation to buildings. The program determines the area and position of the shadow cast by a tree or group of trees on a surface of a building which can have any orientation. The program outputs numerical and graphical hourly results at any time of the day, for any day of the year, at any location on the earth, for any relative positioning between the tree and building, and estimates percent irradiation reductions on building surfaces resulting from tree shade. The program allows three different shapes of tree (elliptical, cylindrical, and parabolic) to be used. This program is being interfaced with a commercially available building energy analysis program.

(127 pages)
Presently, shadow algorithms for point sources are identical to hidden-surface algorithms which require large amounts of computer time and memory. The amount of time and space required are limiting factors on most microcomputers. This paper uses an alternative method in an attempt to overcome some of the limitations of previous work by cutting the number of calculations required to approximate a shadow cast by a tree.

Shadow Pattern Simulator (SPS) is a program designed to interface with a commercially available microcomputer-based building energy analysis program (MICROPAS).

Many variables affect the shadow’s shape and position. Solar variables include solar declination, solar altitude and solar azimuth, angle of incidence, and times of sunrise and sunset. Other factors that affect the shadow’s shape include tree shape, height, and building surface orientation.

Some authors (McPherson 1984, Terjung & Louie 1972) suggest that trees can be typified either as having a spherical, parabolic, or cylindrical shape. That assumption was adopted in the development of the shadow-casting algorithm.

Both increased energy costs and our growing awareness that trees modify our environment have created interest in
potential savings with trees. Trees can reduce energy costs by controlling how much sunlight strikes and enters a building. Shade from trees can reduce the need for air-conditioning in summer, and sun through windows can provide considerable heat in winter. This report develops a computer model for determining the area, position, shape, and shade coefficients of the shadows cast by a tree or group of trees on a surface of a building with any orientation.

SPS was first written in 1985 in FORTRAN for the VAX II. Since then, it has been transported to a microcomputer first using MS-FORTRAN and finally translated to TURBO PASCAL.

This program is actually being used for research at the University of Arizona by E. Gregory McPherson, professor in the landscape architecture program. Two papers have been published (McPherson, Brown, & Rowntree, 1985 and McPherson, Rowntree, 1986) describing the program and its applications.
CHAPTER II
BASIC GEOMETRY

Coordinate Systems

To represent a point in a three-dimensional world, we need a three-dimensional coordinate system. We choose one that requires three mutually perpendicular (orthogonal) axes. The three-dimensional coordinate system can be viewed as the two-dimensional coordinate system (XY plane) augmented by a Z axis through the origin, resulting in two additional planes (the XZ plane and the YZ plane). Two orientations are possible when we elect to augment the Z axis. If the Z axis points upward, the associated coordinate system is referred to as the right-handed system. Similarly, a coordinate system with the Z axis points down is referred to as the left-handed system. As a matter of convention, we use the right-handed coordinate system as our world coordinate system.

Types of Geometric Models

To model the geometry of three-space we need mathematical models of volumes, surfaces and curves. Two basic methods exist to do this, classification and enumeration.

The basic idea of the first method is that there exists a point-membership classification function. This is given by
either a formula or a three-valued procedure, that given the coordinates of a point, returns whether the point is inside, outside, or on the shape. Mathematically, this can be expressed as

\[
F(x, y, z) =
\begin{align*}
< 0 & \quad \text{inside} \\
= 0 & \quad \text{on} \\
> 0 & \quad \text{outside}
\end{align*}
\]

The second method enumerates the points of interest by explicitly generating them. In this case, the function is a mapping from a set of parameters to a set of points. Calling the function with a given parameter set generates a point on the shape. If the object is a curve there is only one parameter, if it’s a surface there are two parameters, and if it’s a volume there are three parameters. In the case of surfaces, this can be mathematically expressed as

\[
(x(u, v), y(u, v), z(u, v), w(u, v))
\]

where \( x, y, z \) and \( w \) are independent functions of the parameters \( u \) and \( v \). A surface defined in this way is often called an explicit or parametric surface.

A mathematical model of a shape is not unique and the two methods described above are not mutually exclusive. For many
primitive shapes both types of descriptions can be used.

**Point, Planes, Lines and Rays**

In projective three-space a point is represented as the four-component vector

\[ x = (x, y, z, w) \]

and its coordinates in Euclidean three-space are

\[ \left( \frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right) \]

If \( w \) equals 1, we may safely ignore the fourth component. One of the advantages of the homogeneous representation is that points and planes are the duals of each other. This means that all calculations involving operations between points can be replaced by calculations involving operations between planes. The representation of a plane is

\[ a = (a, b, c, d) \]

A point with a homogeneous coordinate of zero corresponds to a point at infinity. These points can also be considered direction vectors. One advantage of the homogeneous representation is that directions and finite-points can be
used interchangeably. This is not always true when using vector arithmetic.

A line is a linear combination of two points,

\[ p = sx + ty, \]

or two planes,

\[ p = sa + tb. \]

In the above equations, the same point or plane is generated for all parameter values which give the same ratio of s/t. The point form of the line equation is more commonly interpreted in the following way

\[ p = o + td \]

where \( o \) is a finite point (homogeneous coordinate equal to 1), and \( d \) is a direction (homogeneous coordinate equal to 0). If we restrict \( t \) to only positive values, the above equation represents a ray, where \( o \) is its origin and \( d \) is the direction in which it's moving.
CHAPTER III
THE SOLAR GEOMETRY

For the sake of simplicity, the Earth is assumed to be static; the position of the sun at any time can be described by its coordinates, altitude and azimuth, and its path in the sky can be determined by means of some trigonometric equations that take into account variables like the latitude and longitude of the site and the time of the day and year.

The American Ephemeris and Natural Almanac, published yearly, contains tables for determining the position of the sun. It would be possible to store this information in an array for each year of interest, but this approach would require a huge amount of data to be entered for a small number of years. A better approach is to calculate the position of the sun directly (Walraven 1978).

The solar variables included in the program to characterize the position of the sun in the sky or in relation to the surface to be shaded are shown in Figure 1 and described below.
Solar Declination

The solar declination is the angle between the earth's equatorial plane and the line joining the earth and sun centers. This angle is approximately 23.5 degrees. It is this tilt that causes the elevation of the sun at noon to vary throughout the year.

Time

For purpose of calculating the position of the sun, time shall be measured in units of days from midnight at Greenwich, England, December 31, 1979.

To determine the position of the sun in the sky at a given time, it is necessary to know not only the position of the sun on the celestial sphere, but the position of the celestial sphere with respect to the earth at that time. The rotation of the celestial sphere is measured in sidereal time relative to the local celestial meridian.

The sidereal time in radian from Greenwich, England, is given by

\[ S_G = 1.7248732 + 3.6723 \times 10^{-7} \times \text{time} + 2 \times \left( \frac{\text{time}}{365.25} - \text{delyr} \right) \]

where delyr is the number of years from 1988.

The local sidereal time at the site of the observer is
obtained from SG by adding the time of day, expressed in radian, and subtracting the local longitude in radian west of Greenwich.

Longitude of the Sun

The position of the sun on the celestial sphere can be found by a single parameter, the longitude LONG. Since the earth’s orbit about the sun is not circular, the speed of travel of the sun on the ecliptic varies slightly throughout the year, so the expression for the longitude is somewhat complicated. The longitude of the sun, in radian, is calculated as follows:

\[
\begin{align*}
\text{LONG} &= 2 \times \text{time} / 365.25 \\
&= -0.06564994 - 4.541864\times10^{-7} \times \text{time} + \\
&= 4.866563352 + 3.67658\times10^{-7} \times \text{time} + \\
&\quad (0.0334339 - 2.288\times10^{-9} \times \text{time}) \sin(2 ) + 3.49303\times10^{-4} \sin(2 )
\end{align*}
\]

Local Azimuth and Elevation

The position of the sun in the sky for a particular site can be specified by two angular coordinates: the solar azimuth and the solar elevation. The solar azimuth (a) is the angle measured clockwise between the true north and the projection of the solar ray on a plane tangent to the earth
at the site. The solar elevation (e) is the angle between a direct solar ray and the above-mentioned plane.

The local elevation of the sun is just $90^\circ$ minus the zenith distance

$$e = 90 - z$$

The desired expression for the local azimuth and elevation angles follows directly from the spherical triangle shown in Figure 1b, where $z$ is the local latitude, and the hour angle $H$ is defined as right ascension minus local sidereal time, as shown in Figure 1c,

$$H = - S$$

From spherical trigonometry, the triangle in Figure 1b yields

$$\cos(z) = \sin(e) = \sin( ) \sin( ) + \cos( ) \cos(\ )$$

$$\sin(\ ) = \cos( ) \sin( ) / \sin(z)$$

A problem arises when $\alpha$ is computed. The arcsin function that is used in computers returns a value that is, by convention, in the range $-90^\circ$ to $+90^\circ$. If the true $\alpha$ is outside the range, a correction must be made.

The proper correction to $\alpha$ is
Figure 1. (a) The Celestial Sphere; (b) A Spherical Triangle from the Celestial Sphere Defining the Zenith Distance and the Azimuthal Angle.
if (cos(z) < sin( ) / sin( )) if (a < 0) a = a + 360
a = 180 - a

Sunrise and Sunset

Sunrise and sunset do not occur exactly at the time when the elevation of the sun is 0°, due to atmospheric diffraction and curvature of the earth. The elevation of the sun at sunrise or sunset is given approximately by

\[ e = -(0.833 + 0.0214 \ h) \]

where \( h \) is the local height above sea level in feet.
CHAPTER IV
GEOMETRY OF THE TREE’S SHADE

When using trees to shade buildings one of the problems is where their shadows will be at different times of the day throughout the year. The solution of the problem must take into account three elements: the tree’s geometry, the position of the sun in the sky at the required times and the characteristics of the surfaces to be shaded.

Some authors (McPherson 1984, Terjung & Louie 1972) suggest that trees can be typified either as having elliptical, parabolic or cylindrical shapes, or as being a combination of these shapes. That assumption was adopted in this paper with the limitation that the trees are simple; that is, they are not composed of more than one shape.

The tree should be geometrically defined by the coordinates of some characteristic points (such as the center of the canopy and the radius of a spherical-shaped tree, or the positions of the top and base together with the radius of this base for a parabolic- or cylindrical-shaped tree) in relation to a chosen system of coordinate axes. In this paper the coordinate axes have their origin on the bottom left-hand corner of the surface to be shaded. The sun, the trees and the surface being studied are referred to this coordinate system. If, for the sake of simplicity, the Earth is assumed to be static, the position of the sun at any time
can be described by its coordinates, altitude and azimuth, and its path in the sky can be determined by means of some trigonometric equations that take into account variables like the latitude of the site and the time of the day and year.

The surface, representing the different components of the building envelope, are defined in terms of their orientation in relation to the true north (wall azimuth angle) and to the horizontal (inclination). Moreover, the relationship between the solar position and the different building surfaces is described by the wall-solar azimuth angle (horizontal shadow angle) and by the angle of incidence of the sun's rays on the surface.

SPS uses a complex algorithm to calculate shadows on a building surface produced by objects other than the building itself. Some approximations and assumptions are used to reduce calculation complexity.

Calculations occur for objects located within a right-handed rectangular coordinates system in three-dimensional space. To describe the sun's position and remain consistent within this coordinates system, the y axis is defined as parallel to the north wind direction.

The idea behind the algorithm is to calculate shadows on a vertical surface parallel to the y axis and facing the positive x axis (that is, it is facing east), and with the shading objects in front of this surface. These surfaces are
called "normal" (section A on Figure 2). With this idea in mind, calculations of shadows on any other surface, with any slope, and which may or may not be parallel to the y axis are possible, provided that slope and displacement of the surface is known.

The position of the sun is calculated using equations developed by R. Walraven (1978).

Program Overview

SPS was developed to estimate the percent of available total solar radiation reduced by tree shade at opaque and glazed exterior building surfaces. MICROPAS, a microcomputer-based building analysis program, is being modified to accept this data.

The system can be described as being subdivided into three distinct sections (Figure 3). The first section provides the capability of entering, modifying, and saving the input information necessary for a simulation. The second section provides a graphical plan view of the information entered to the model. The analysis and output graphing operations comprise the last section. Although the first and last sections must be completed sequentially before starting the next, the sections themselves contain a number of procedures which may be used in any order, eliminating the rigidity of a single-path process.
Figure 2. Once the Displacement Angle is Entered for Structures Oriented other than East (above), SPS Normalizes All Surfaces (below).
Figure 3. Program Diagram
A single building and surrounding trees are located by grid coordinates. Input parameters include

1. Building orientation
2. Building length
3. Building width
4. Building wall height
5. Roof pitch
6. Base coordinates of shading objects
7. Height of shading object
8. Trunk height
9. Crown diameter
10. Canopy shape
11. Canopy shading coefficients

The user specifies the simulated day and site location to create an array of solar angles that SPS obtains. Using this data, SPS calculates the percentage of surface area shaded for each one-half hour on a surface by surface basis. Wall/roof surfaces are perceived as grid cells that, when shaded, are filled with the tree’s shading coefficient (the percent of available solar radiation transmitted through the tree crown). If shadows from several trees overlap on the surface, shading coefficients are multiplied. The three half-hour values encompassing each hour are averaged to derive an hourly value for each cell of the surface in
question. This array can be displayed graphically to illustrate the areal reduction of direct solar radiation due to tree shade (Figure 4).

To modify direct solar radiation values to account for tree shade, SPS averages all of the hourly shading coefficients in each cell to derive a single hourly value for each irradiated surface. This value is multiplied by the actual amount of available direct solar radiation generated by the MICROPAS weather processor. Thus, the direct solar radiation values used to calculate space conditioning costs are reduced as a function of tree shade.

SPS was fine-tuned with a heliodon and three-dimensional model. SPS uses the same algorithm to determine reductions of direct and diffuse solar radiation. Sky view factors are not calculated for diffuse radiation, and as a result shading effects are increasingly overestimated as the relative magnitude of the diffuse components increase, refer to Figure 3. Because the yearly diffuse component is generally less than 25% of the total solar flux (McPherson 1984), this omission is not of major importance in most regions. The small amount of radiation reflected off leaves to building surfaces are not considered. SPS assumes that the building is on a flat horizontal surface, which is used as datum for the location of the shading objects.
Figure 4. SPS Graphic Output: Shadow Pattern from a Parabolic Tree Canopy.
Calculating Shadows on Normal Surfaces

A mathematical model is used to calculate shadows. It permits one to describe the building surfaces, the shading objects, and the sun’s rays.

A plane in space is used to represent the building surfaces. Parabolic, elliptic, and cylindrical shapes are used to represent tree canopies, tree trunks, and walls. These three-dimensional geometric shapes are decomposed into two, two-dimensional equations, which simplify calculations. In section, the objects are represented by a parabola, ellipse, and rectangle, respectively (Figure 5). In plan view they are represented by concentric circles.

The sun’s rays are represented by sets of straight parallel lines.

The algorithm assumes that the input parameters previously noted are known. It then calculates the shaded position of the tree crown. The boundary between the illuminated and shadowed portions is defined by points on the crown where the sun’s rays are tangent. It is not necessary to calculate all the boundary points. The only points needed are a point at the tip of the shaded section of the crown, another at the bottom, and another two that delimit the width of the shaded portion (points on the boundary of the widest section of the shaded portion). These points are calculated using the two-dimensional
Figure 5. Sections and Plan Views of the Three Shapes Show a Shadowed Section.
approximation for the geometric shape that describes the shape of the shading object in question.

For a parabola \( y^2 = -4pz \)
For an ellipse \( \frac{y^2}{b^2} + \frac{z^2}{a^2} = 1 \)
For a rectangle \( y = y_1 \) or \( y_2 \)
\( z = z_1 \) or \( z_2 \)

where \( y_1 \) or \( z_1 \) are the left and right limits on the \( y \) axis, \( z_1 \) and \( z_2 \) are the bottom and top limits of the rectangle.

A circle is used for the second two-dimensional equation.

For a circle \( (y^2 + z^2 = r^2) \)

To represent the sun's rays it is necessary to calculate a unit vector that describes the direction in space. This is done using the two-dimensional angle, azimuth and altitude.

It is now possible to calculate the four points on the shading object. Knowing the equation that describes the shape of the object and the direction of the sun's rays, the algorithm calculates the points on the object that are tangent to a straight line parallel to the unit vector.

The equations that calculate these points are the following:
Parabola

\[ \begin{align*}
PT_p &= (x - r \cos(\beta), y - r \sin(\beta), -\tan^2(\theta) p + z_t) \\
PMR_p &= (x - D \sin(\beta)/2, y + D \cos(\beta)/2, z_h) \\
PML_p &= (x + D \sin(\beta)/2, y - D \cos(\beta)/2, z_h) \\
PB_p &= (x + D/(2 \sin(\beta)), y + D/(2 \sin(\beta), z_b) \\
\end{align*} \]

Ellipsoid

\[ \begin{align*}
PT_e &= (x - r \cos(\beta), y - r \sin(\beta), z_h - a(1 - (r/b)^2)^{\frac{1}{4}}) \\
PMR_e &= (x - D \sin(\beta)/2, y + D \cos(\beta)/2, z_h) \\
PML_e &= (x + D \sin(\beta)/2, y - D \cos(\beta)/2, z_h) \\
PB_e &= (x + r \cos(\beta), y + r \sin(\beta), z_h - a(1 - (r/b)^2)^{\frac{1}{4}}) \\
\end{align*} \]

Cylinder

\[ \begin{align*}
PT_c &= (x - r \cos(\beta), y - r \sin(\beta), z_t) \\
PMR_c &= (x - D \sin(\beta)/2, y + D \cos(\beta)/2, z_h) \\
PML_c &= (x + D \sin(\beta)/2, y - D \cos(\beta)/2, z_h) \\
PB_c &= (x + D/(2 \sin(\beta)), y + D/(2 \sin(\beta), z_b) \\
\end{align*} \]

where

PT is the top point
PMR is the right point on the widest section
PML is the left point on the widest section
PB is the bottom point
x, y coordinates of the object
r the radius of the widest section
\( z_t \) height of the tip of the object

\( z_b \) height of the bottom of the object

\( z_h \) height of the widest section of the object

a crown length (major axis of an ellipse)

b crown diameter (minor axis of an ellipse)

ß sun's azimuth angle

sun's elevation angle

The second step is to project these points to a plane in space which contains the building surface. The plane can be described with three points and the following equations

\[
\begin{align*}
Ax_1 + By_1 + Cz_1 - 1 &= 0 \\
Ax_2 + By_2 + Cz_2 - 1 &= 0 \\
Ax_3 + By_3 + Cz_3 - 1 &= 0
\end{align*}
\]

where \( x_n, y_n, z_n \) are coordinates of a points on the building surface \( (n = 1, 2, 3) \), and \( A, B, C \) are constants.

To calculate the projection of the four points, straight lines that are tangent to the object at each of the four points and parallel to the unit vector are used. The following equations describe such lines

\[
\begin{align*}
x_o &= x + ta \\
y_o &= y + tb
\end{align*}
\]
\[ z_0 = z + tc \]

where \( x, y, z \) are the coordinates of the point on the object; \( t, a, b, c \) are coordinates; and \( x_0, y_0, z_0 \) are coordinates of a point on the plane.

By solving the three equations that describe the plane in space for the parameters \( A, B, \) and \( C \) and inserting these results in the general equation for a plane we obtain

\[ Ax_0 + By_0 + Cz_0 - 1 = 0 \]

The projection is simulated by obtaining the intersection of the lines with the plane. This is done by equating the equation of a plane and a line. The intersection will generate a point whose coordinates are those of the projected point.

\[ T = (1 - Cz - By - Az)/(Aa + Bb + Cc) \]
\[ x_{on} = x + Ta \]
\[ y_{on} = y + Tb \]
\[ z_{on} = z + Tc \]

where \( n \) can be "t" for top, "m" for middle, "r" for right, or "l" for left. In this manner all four points are projected.

Two checks are made before the shadow is approximated. The first verifies if the shadow bottom is flat (Figure 6). If it is not flat, a curve (half ellipse) is used. To
Figure 6. There is no Distortion of the Shadow and the Bottom is Flat when Sun's Rays are Normal to the Tree Crown and the Receiving Plane.
Figure 7. The Shadow Pattern is Distorted and the Bottom Rounded when Incidence Angles are Not Normal.
accomplish this a fifth point \((x_{om}, y_{om}, z_{om})\) serves as a delimiter between the top portion and the bottom of the shadow. This fifth point is the midpoint of the widest section of the shading object, and it is projected using the method described above.

The second check detects if the shadow is twisted. The angle, \(\theta\), is calculated. \(\theta\) is formed between an imaginary line that goes through the top and bottom points of the shadow and the \(z\) coordinate. The shadow is twisted if \(\theta\) is not equal to 0.

\[
= - \arctan\left(\frac{y_{ot} - y_{ob}}{z_{ot} - z_{ob}}\right)
\]

With the above information the algorithm can approximate the shape of the shadow on the building surface. Using one of the three two-dimensional equations (parabola, ellipse, or rectangle) that describe the shape of the shading object, the algorithm fits a curve that goes through the first four projected points and approximates the bottom with a halfellipse if required. If the shadow is twisted, all points on the shadow boundary are rotated using the angle \(\theta\).

The equations used to approximate the coordinates of a point that falls on the shadow boundary and make the above checks and corrections are
\[ d = y_\text{or} \cos(\ ) + z_\text{or} \sin(\ ) - y_\text{ol} \cos(\ ) + z_\text{ol} \sin(\ ) \]

For a parabolic shape

\[ p = 2 (d/2)/(4 [(y_\text{ot} + y_\text{om})^2 + (z_\text{ot} - z_\text{om})^2]^{\frac{1}{2}}) \]

\[ y_\text{o}^2 \cos^2(\ ) \sin(\ ) + 2 y_\text{o} z_\text{o} \cos(\ ) \sin(\ ) + z_\text{o} \sin^2(\ ) = 4 p z_\text{o} \cos(\ ) + 4 p y_\text{o} \sin(\ ) \]

For an elliptic shape

\[ a = [(y_\text{ot} - y_\text{ob})^2 + (z_\text{ot} - z_\text{ob})^2]^{\frac{1}{2}} / 2 \]
\[ b = d/2 \]
\[ a^2 y_\text{o}^2 \cos^2(\ ) + 2 y_\text{o} z_\text{o} \cos(\ ) \sin(\ ) + a^2 z_\text{o}^2 \sin^2(\ ) + b^2 z_\text{o}^2 \sin^2(\ ) + b^2 z_\text{o}^2 \cos^2(\ ) - 2 b^2 y_\text{o} z_\text{o} \cos(\ ) \sin(\ ) + b^2 y_\text{o}^2 \sin^2(\ ) = a^2 b^2 \]

Given a \( z_\text{o} \) value, the approximate coordinate of a point on the shadow boundary can be found by solving the above equations for \( x_\text{o} \) and \( z_\text{o} \).

The equation is much easier for a rectangular shape

\[ y_\text{o} = y_\text{om} + d/2 - z_\text{o} \sin(\ ) \]
\[ x_\text{o} = x_\text{om} \]
The algorithm uses these equations for all points of the fitted curve to reproduce an approximated shadow. Points lying between the projected boundary points are shaded.

Calculating Shadows on Non-Normal Surfaces

Use of the algorithm described above requires rearrangement of the environment if non-normal surfaces are simulated. That is, it is necessary to rotate the section of the building under analysis and all shading objects in front of it so that the section surface is normal and the objects are located in a plane such that they reproduce the same shadows as if they were in their original locations. The solar azimuth angle is also modified so that it will generate the same shading effect as if it were in its original position.

The transformation of the building surface is achieved using the standard transformation equations.
CHAPTER V
VALIDATION OF THE PROGRAM

In order to test the output of the program scale models of four trees and a house, suitable to be placed on a heliodon, were built.

Description of Scale Model

The house was made from cardboard and covered with paper which had a grid on it. The trees were plastic. The heliodon had a wooden plate where the objects were placed. Distributed along the border of the plate, several threaded holes were cut, thus defining alternative positions for trees.

The model of the house was built having a fixed roof with no overhangs at the roof level.

The trunks of the trees are simulated by cylindrical metallic rods with threads in both ends, one end to be screwed into the plate and the other into the canopy of the "tree".

Description of the Validation Procedure

The validation of the program was made by comparing the results given by the computer program with those obtained on the heliodon. The program was run on a Leading Edge model M, with no math-coprocessor and 640K of memory. When this
computer is connected to an Epson printer, the latter can produce hard copies of the graphics displayed on the screen of the terminal. It can be seen that the shadows of the trees cast on the heliodon show a slight difference when compared to those output by the program. The difference can be partly attributed to variations in the dimensions of the tree models. Another source of error is thought to be the constructional limitations of the heliodon (Markus & Morris 1980). Figure 8 illustrates the distortion that can be expected to be found in the shadow on a heliodon due to the limited distance between the models and the light source. It can be seen that instead of being a parallel beam, like the sun rays, the light diverges from the source.

In order to check the numerical result, the area of the shadows produced by the heliodon model were traced with a marker and the area outlined were measured using a planimeter. These areas were compared to those output by the program. The results are shown in Table 1.

These results show that, although increasing for small areas, the relative difference can be considered negligible.
Figure 8. Shadow Distortion on Heliodon.
Tree shape: Parabola
Simulation date: March/Sept. 21

<table>
<thead>
<tr>
<th>Time</th>
<th>East Wall</th>
<th>East Roof</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Percent shade</td>
<td>Heliodon</td>
</tr>
<tr>
<td>7:00</td>
<td>14.8</td>
<td>13.8</td>
</tr>
<tr>
<td>7:30</td>
<td>20.6</td>
<td>18.5</td>
</tr>
<tr>
<td>8:00</td>
<td>25.3</td>
<td>22.4</td>
</tr>
<tr>
<td>8:30</td>
<td>29.9</td>
<td>27.6</td>
</tr>
<tr>
<td>9:00</td>
<td>33.6</td>
<td>31.3</td>
</tr>
<tr>
<td>9:30</td>
<td>32.6</td>
<td>31.3</td>
</tr>
<tr>
<td>10:00</td>
<td>26.0</td>
<td>21.9</td>
</tr>
<tr>
<td>10:30</td>
<td>15.6</td>
<td>8.1</td>
</tr>
<tr>
<td>11:00</td>
<td>7.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>West Wall</th>
<th>West Roof</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Percent shade</td>
<td>Heliodon</td>
</tr>
<tr>
<td>2:00</td>
<td>1.6</td>
<td>0.0</td>
</tr>
<tr>
<td>2:30</td>
<td>12.5</td>
<td>4.9</td>
</tr>
<tr>
<td>3:00</td>
<td>23.4</td>
<td>19.3</td>
</tr>
<tr>
<td>3:30</td>
<td>32.3</td>
<td>31.0</td>
</tr>
<tr>
<td>4:00</td>
<td>31.8</td>
<td>29.9</td>
</tr>
<tr>
<td>4:30</td>
<td>29.7</td>
<td>27.1</td>
</tr>
<tr>
<td>5:00</td>
<td>24.5</td>
<td>22.1</td>
</tr>
<tr>
<td>5:30</td>
<td>19.3</td>
<td>18.2</td>
</tr>
<tr>
<td>6:00</td>
<td>15.9</td>
<td>13.3</td>
</tr>
<tr>
<td>6:30</td>
<td>11.7</td>
<td>8.1</td>
</tr>
</tbody>
</table>

Table 1. Percent Surface Shaded by Heliodon Model and Calculated by the Program.
CHAPTER VI
CONCLUSIONS AND RECOMMENDATIONS

The theoretical background and application of a computer program concerning the use of trees for shading are described. The graphical output of the program was validated on a heliodon using scale models of trees and building.

The major strength of the entire project is that it succeeds in modeling an extremely complex and important process at a level not previously attained. The interactive input method, graphical displays, ease of use, and speed make SPS a useful tool.

The shadows of the trees cast on the heliodon show a slight difference when compared to those output by the program. The difference can be partly attributed to variations in the dimensions of the tree models. Another source of error is thought to be the constructional limitations of the heliodon.

The results given by the program may be used by building designers in order to determine the position of trees’ shadows on buildings for different relative positionings of trees and buildings. This information is important either to prevent the shading of windows, walls and solar collectors when solar gains must be maximized or, on the contrary, to benefit from it, using trees as shading devices when the local climate so requires. SPS can also be used to (1)
determine the optimal height for existing trees in order to provide winter sun and summer shade; (2) predict the future impact of plantings on solar access and solar control; (3) locate the best places for solar collectors given existing and future shadow patterns; and (4) calculate the amount of solar access blocked by trees.

Model Limitations

The model uses some assumptions which may not hold in some circumstances and thus place limitations on the applicability of the model. One of the principal assumptions was that a building has four flat walls and a roof with no overhangs. This assumption limits the model to small, one-storied houses.

Tree canopies are assumed to be simple, limiting the program only to ellipsoidal, parabolic, or cylindrical trees.

The modeling of the shadows is done at 30-minute intervals. Actual field conditions might require shorter or longer intervals.

Alternative Method

The method described in this paper uses the idea of forward ray tracing; that is, the viewer is the light source (the sun), and the ray is followed until it hits the building surface or until we are satisfied that it would not intersect
the surface being analyzed. This method introduces various limitations and increases the number of calculations to determine the shadow locations on the projected plane.

The alternative method uses the backward ray-tracing idea. In this case, the viewers are each cell on the surface being analyzed and looking in the direction of the sun. Each cell traces an individual ray to its source. If the ray intersects an object before arriving at the light source, then the cell is shaded.

For modeling the trees, constructive model geometry can be used. One of the most popular methods constructs solid objects from simpler objects by performing set operations, usually union or difference. A composite object can be represented as a binary tree where a node contains a set operation and two child nodes, which may also be composite solids. The leaves of this tree are primitive objects such as ellipsoids, planes, etc. One of the advantages of this representational scheme is its completeness; another advantage is the ease with which it can be ray-traced.

Recommendations for Future Study

The following recommendations are made:

1. The programs can be enhanced by adding the capability of combining the basic shapes to form a composite tree canopy.
2. A graphical output of percent surface shaded for an entire run would give the user a better idea of the amount of shade a surface received. This output can be in the form of a bar graph.

3. The model can be improved by allowing the user to specify the number, shape and location of the building’s surfaces.

4. Diffuse solar radiation calculations can be added to the model to increase the accuracy of results.

5. Implementation of surface details such as windows, doors, etc., would improve the design applicability of the model.
REFERENCES


Program: SPS (Shadow Pattern Simulator)

This is an interactive program which calculates solar radiation reductions and produces shadow patterns on buildings surfaces.

Main Module

program sps;
{$V-,C-,U-}
{$I graph.p}
type
  CharSet = Set of Char;
  CharArty = array[1..9] of Char;
const
  MaxRecSize = 48;
  Nrow = 15;
  Ncol = 60;
  Pcol = 480;
  Prow = 120;
  MaxListLen = 30;
  rad = 0.0174533;
  intervalY = 40;
  OriginX = 142;
  OriginY = 12;
  IntervalX = 80;
  Constant = 1.03;
  LastSeason = 8;
  EndOfDay = 24;
  ColorGlb = 255;
  GrafBase = $B800;

days_in_month: array[1..12] of byte=
  (31,28,31,30,31,30,31,31,30,31,30,31);
AbortSet : CharSet = [^[,^,C];
FileChar : CharSet =
  ['A'..'Z','O'..'9','$','#','&','%', '(',')','-',',','@',
  '!','(',')'','/'];
AcceptM : CharArty = ('C','E','S','T','D','Q',#0,#0,#0);
type
  AnyStr = String[255];
  Str80 = String[80];
  Str64 = String[64];
  Str50 = String[50];
  Str12 = String[12];
  Str7 = String[7];
  Ary1x3 = array[1..3] of real;
Ary3x3 = array[1..3,1..3] of real;
Ary5x3 = array[1..5,1..3] of real;
surface = array[1..Nrow,1..Ncol] of byte;

find_first_template_type = string[64];
find_filename_type = string[12];

FileType = file;

RegPack = record
    ax,bx,cx,dx,bp,si,di,ds,es,flags : integer
end;
result = record
    AL,AH: Byte;
    BX,CX,DX,BP,SI,DI,DS,ES,Flags: Integer;
end;
results = record
    AL,AH,BL,BH,CL,CH,DL,DH: byte;
    BP,SI,DI,DS,ES,Flags: Integer;
end;

ShadingFactors = record
    Sv,Wv,Nv,Ev,St,Wt,Nt,Et : real
end;

ObjPointer = ^ObjRec;

BldRec = record
    lngth,hgt,wdth,RoofAng,DispAng : real;
    chk : byte
end;
ObjRec = record
    Base,Top,Dia,Xo,Yo,pct : real;
    shape : char;
    num : integer;
    next : ObjPointer;
    prior : ObjPointer;
    chk : byte;
end;
LocRec = record
    lat,long,ZTime : real;
    DLS : char;
    chk : byte
end;
DateRec = record
    Year,Month,Day : real;
    chk : byte
end;
SunPosition = record
    t : String[5];
    azm,elv : real
end;
SunRec = array [0..47] of SunPosition;

var
  BldGlb        : BldRec;
  LocGlb        : LocRec;
  DateGlb       : DateRec;
  LastNumGlb    : integer;
  startGlb, lastGlb : ObjPointer;
  LastDispGlb   : ObjPointer;
  FirstDispGlb  : ObjPointer;
  savedGlb      : boolean;
  IOstat        : integer;
  FileNameGlb   : Str12;
  CurrDirGlb    : Str64;
  OverlayPathGlb: Str64;
  SunGlb        : SunRec;
  sec           : char;
  GridGlb       : surface;
  AveGlb        : array[0..47] of real;
  GrdGlb        : boolean;
  DrawGlb       : boolean;
  AsciiGlb      : boolean;
  ShdF          : file of ShadingFactors;
  SeasonCodeGlb : integer;
  CommandLineGlb: String[6];
  NewDate, NewLoc: boolean;
  EdMode        : boolean;
  OutHrAve      : boolean;
  PlotGlb       : boolean;
  Time1, Time2 : integer;
  choice        : integer;
  Time1Str, Time2Str: String[5];
  DelayGlb      : integer;
  PrintGlb      : boolean;
  GraphCount    : byte;
  QuitGlb       : boolean;

(*
  Sun Position Module
*)

overlay procedure AzmElv;
var
  hr, min, sec, dasvtm, jday : real;
  count                      : integer;
  Thr                         : String[2];

function JulianDay(month, day, year: integer): real;
  { This function converts a Gregorian day to a Julian day.}
  var
i : integer;
temp : real;

begin
  temp := 0;
  { Add the number of days for each complete month in the year. }
  If month > 1 then
    For i := 1 to month - 1 do
      temp := temp + days_in_month[i];
  { If the year is a leap year and the month is greater than 2, subtract a day. }
  If (month > 2) and (year mod 4 = 0) then
    temp := temp - 1;
  JulianDay := temp + day; { Add the days. }
end;

(*
  Calculate sun azimuth and elevation for one day and half hour intervals
*)

procedure sunpos( year, day, hr, min, sec,
  zone, Dasvtm, lat, long : real;
  var sa, se : real);

var
twoPi, rad, Delyr, t, time,
theta, g, el, eps, sel, al, a2, ra : real;
decl, st, s, h, phi : real;
leap, temp : real;

begin
  twoPi := 2.0*pi;
  rad := pi/180.0;

  Delyr := year - 1980.0;
  leap := trunc(delyr/4.0);
  t := hr + (min + sec/60.0)/60.0 + zone - Dasvtm;
  time := delyr*365.0 + leap + day - 1.0 + t/24.0;
  if delyr = (leap*4.0) then time := time-1.0;
  if (delyr < 0.0) and (delyr <> leap*4.0) then
    time := time-1.0;
  theta := (360.0*time/365.25)*rad;
  g := -0.031271 - 4.53963E-7*time + theta;
  el := 4.900968 + 3.6747E-7*time + (0.033434 - 2.3E-9*time)*
    sin(g) + 0.000349*sin(2.0*g) + theta;
  eps := 0.409140 - 6.2149E-9*time;
  sel := sin(el);
  al := sel*cos(eps);
  a2 := cos(el);
  ra := atan2(al,a2);
  if ra < 0.0 then ra := ra + twoPi;
  decl := asin(sel*sin(eps));
st := 1.759335 + twoPi*(time/365.25 - delyr) + 3.694E-7*time;
if st >= twoPi then st := st - twoPi;
s := st + (t*15.0 - long)*rad;
if s >= twoPi then s := s - twoPi;
h := ra - s;
phi := lat*rad;
temp := sin(phi)*sin(decl) + cos(phi)*cos(decl)*cos(h);
se := asin(temp);
sa := asin(cos(decl)*sin(h)/cos(se))/rad;
if sin(se) < sin(decl)/sin(phi) then
begin
  if sa < 0.0 then sa := sa + 360.0;
  sa := 180.0 - sa
end;
se := se/rad;
sa := 180.0 - sa
end;

(*
Sun Position Main procedure
*)

begin
  ClrScr;
  if (DateGlb.chk <> 7) or (LocGlb.chk <> 15) then
  begin
    error('Insufficient information');
    exit
  end;
  DrawBox(26,9,54,14,'');
  Gotoxy(28,11); write('Calculating sun position. ');
  Gotoxy(28,12); write('Please wait a moment. ');
  hr := 0.0;
  min := 0.0;
  sec := 0.0;
  dasvtm := 0.0;
  count := 0;
  NewDate := false;
  NewLoc := false;
  with DateGlb do
  jday := JulianDay(trunc(month), trunc(day), trunc(year));
  if LocGlb.dls = 'Y' then dasvtm := 1.0;
  repeat
    with SunGlb[count], LocGlb, DateGlb do
    begin
      sunpos(year, jday, hr, min, sec, Ztime, dasvtm, lat, long, azm, elv);
      str(trunc(hr):2, Thr); t := Thr;
      str(trunc(min):2, Thr); t := t + ':' + Thr;
      min := min + 30.0;
      if min >= 60 then

begin
  hr:= hr+1.0;
  min:= 0.0
end
end;
count:= count+1;
until hr = 24.0
end;

(*
  Return memory to system, reset screen, and return
  to the system.
*)

procedure Quit;
var
  temp: ObjPointer;
begin
  while StartGlb <> nil do
    begin
      temp:= StartGlb^.next;
      dispose(StartGlb);
      StartGlb:= temp
    end;
 ClrScr;
  CurNorm
end;

(*
  Obtain a new driver path
*)

procedure GetNewDrive;
var
  temp : Str64;
  TC : Char;
begin
  ClrScr;
  CurBlck;
  GotoXY(1,3); Write('Path: ');
  temp:= CurrDirGlb;
  repeat
    InputStr(temp,64,6,2,[^M]+AbortSet,[]),TC);
    if not(TC in AbortSet) then
      begin
        {$I-}ChDir(temp){$I+};
        IOSTat:= IOresult;
        if IOSTat = 0 then
          CurrDirGlb:= temp
        else error('Path not found')
      end
  end
until (IOstat = 0) or (TC in AbortSet);
CurBlnk
end;

(*
Contract a MACROPAS compatible file
*)

procedure CreateMicroPasFile;
{$I-}
label
escape;
const
blank = ' ';
Season_Name: array[1..8] of string[14] =
('WINTER','WINTER/SPRING','SPRING/SUMMER',
'SUMMER','SUMMER/FALL','FALL/WINTER',
'PEAK HEATING','PEAK COOLING');
var
season,hr,index  : integer;
ShadingFactor    : ShadingFactors;
MPF              : Text;
comment          : Str64;
TC                : Char;
begin
ClrScr;
if not exist(FileNameGlb+'.TMP') then
begin
  error('Shading file does not exist');
goto escape
end;
GotoXY(2,11);
Write('Comment:');
comment:='';
inputstr(comment,64,10,10,[A MJ,[J,TC);
GotoXY(1,11); ClrEol;
DrawBox(15,9,63,14,'');
GotoXY(17,11);
Write('Creating MicroPas Hourly Shading Factors File');
GotoXY(35,12);Write('Please Wait');
assign(ShdF,FileNameGlb+'.TMP');
ReSet(ShdF);
assign(MPF,FileNameGlb+'.SHD');
ReWrite(MPF);
IOcheck;
if IOstat <> 0 then goto escape;
Writeln(MPF,comment);
IOcheck;
if IOstat <> 0 then goto escape;
Writeln(MPF, LastSeason);
IOcheck;
if IOstat <> 0 then goto escape;
index:= 0;
for season:= 1 to LastSeason do
begin
  GotoXY(33,24); ClrEol;
  Write(Season_Name[season]);
  Writeln(MPF, Season_Name[season]);
  IOcheck;
  if IOstat <> 0 then goto escape;
  for Hr:= 0 to EndOfDay-1 do
  begin
    seek(ShdF,index);
    Read(ShdF,ShadingFactor);
    with ShadingFactor do
    begin
      Write(MPF,Hr:2,blank,Sv:5:3);
      Write(blank,Wv:5:3,blank,Nv:5:3,blank);
      Write(Ev:5:3,blank,St:5:3,blank,Wt:5:3);
      Writeln(blank,Nt:5:3,blank,Et:5:3);
    end;
    IOcheck;
    if IOstat <> 0 then goto escape;
    index:= succ(index);
  end {for Hr}
end {for season};
GotoXY(33,24); ClrEol;
Write('MicroPas File Created');
close(ShdF);
close(MPF);
erase ( ShdF) ;
ext;
escape:
close(ShdF);
close(MPF);
{$I+}
end {procedure CreateMicropasFile};

(* Convert time from a string to two integers *)

function ObtainTime( time : Str12;
var t1,t2 : integer): boolean;
var
  p,code : integer;
  r : real;
  temp : string[5];
  first : boolean;
begin
  p:= 1;
  temp:= '';
first := true;
while p < length(time)+1 do
begin
  case time[p] of
    '0'..'9':
      temp := temp + time[p];
    
    :' if (p+1 <= length(time)) and (time[p+1] <> '-')
      and (time[p+1] >= '3') then
      temp := temp + '.5'
    else if time[p+1] = '-' then
      p := p-1;
    '-'
    begin
      val(temp, r, code);
      if first then
        begin
          tl := trunc(2.0*r);
          temp := ''; first := false
        end
      else
        t2 := trunc(2.0*r)
      end
      p := p+1
    end;
end;

ObtainTime :=
  (t1 >= 0) and (t1 <= 47)
  and (t2 >= 0) and (t2 <= 47)
  and (t1 <= t2)

end;

(*
  Display default setting
*)

procedure SetOptions;
const
  accept: CharAry = ('C', 'D', 'P', 'T', 'M', 'Q', #0, #0, #0);
var
  choice, code : integer;
  ok : boolean;
  TC : char;
  temp : string[3];
procedure DispMenu;
begin
  DrawBox(28,5,53,14,'O P T I O N  M E N U');
  DrawBox(10,16,70,24,'O P T I O N  S T A T U S');
  GotoXY(32,7); Write('Create shading file');
  GotoXY(32,8); Write('Display shadows');
  GotoXY(32,9); Write('send to Printer');
GotoXY(32,10); Write('delay Time');
GotoXY(32,11); Write('convert to Micropas');
GotoXY(32,12); Write('Quit');
GotoXY(12,18); Write('Create shading file..');
GotoXY(12,19); Write('Display shadows......');
GotoXY(12,20); Write(' with grid...........');
GotoXY(12,21); Write('Send to printer.......');
GotoXY(12,22); Write('Delay time............');
end;
begin
ClrScr;
choice:= 1;
DispMenu;
repeat
GotoXY(33,18);
if OutHrAve then
  Write(' ON ')
else
  Write(' OFF ');
GotoXY(33,19);
if DrawGlb then
  Write(' ON ')
else
  Write(' OFF ');
Write(' INTERVAL FROM ',TimelStr+
  ConstStr(' ',5-length(Timelstr)),' TO ',
  Time2Str+ConstStr(' ',5-length(Time2str)));
GotoXY(33,20);
if GrdGlb then
  Write(' YES')
else
  Write(' NO ');
GotoXY(33,21);
if PrintGlb then
  Write(' YES')
else
  Write(' NO ');
GotoXY(33,22);
Write(DelayGlb);
choice:= Query(6,7,30,choice,accept);
case choice of
  1: OutHrAve:= not OutHrAve;
  2: begin
    DrawGlb:= not DrawGlb and HardwarePresent;
    if DrawGlb then
      begin
        repeat
          inputstr(TimelStr,5,53,18,[A MJ,'0'..'9',
            ':'],TC);
          inputstr(Time2Str,5,62,18,[A MJ,'0'..'9',
            ':'],TC);
        end;
  end;
ok := ObtainTime(TimelStr+'-'+Time2Str+'-',
                      Timel,Time2);
if not ok then
  begin
    error('Invalid time interval. ');
    DispMenu
  end;
until ok;
if GrdGlb then temp := 'YES' else temp := 'NO';
inputStr(temp,3,33,19,['M'],
          ['Y','E','S','N','O'],TC);
GrdGlb := temp[1] in ['Y','E','S'];
end
end;

3: PrintGlb := Not PrintGlb and PtrWorking;
4: begin
  str(DelayGlb,temp);
  inputStr(temp,2,32,21,['M'],['0'..'9'],TC);
  val(temp,DelayGlb,code);
  if code <> 0 then
    begin
      error('Invalid delay time');
      DispMenu
    end
end;
5: begin
  CreateMicroPasFile;
  ClrScr;
  DispMenu
end
end
until choice = 6
end;

(*
  Display copy right creen
*)

procedure FrontPage;
const
  C = #223;
  blank = ' ';
  MaxNameLen = 16;
  name : array[1..MaxNameLen] of byte =
     (82,111,98,101,114,116,111,32,
      65,46,32,98,114,111,119,110);
var
  i : integer;
  author : string[MaxNameLen];
  ch : char;
begin
  if not exist('sps.000') then begin
    Writeln('Overlay file must be present on default directory path.');
    beep;
    halt
  end;
  author := '';
  for i := 1 to MaxNameLen do author := author + chr(name[i]);
  ClrScr;
  GotoXY(17,2); Write(ConstStr(C,4),blank,ConstStr(C,2));
  Write(blank:10,ConstStr(C,8),blank:15);
  Write(ConstStr(C,4),blank,ConstStr(C,2));
  GotoXY(14,3); Write(ConstStr(C,11));
  Write(blank:9,ConstStr(C,11),blank:10);
  Write(ConstStr(C,11));
  GotoXY(13,4); Write(ConstStr(C,4),blank:6,ConstStr(C,3));
  Write(blank:8,ConstStr(C,3));
  Write(blank:6,ConstStr(C,3),blank:8);
  Write(ConstStr(C,4),blank:6,ConstStr(C,3));
  GotoXY(12,5); Write(ConstStr(C,8));
  Write(blank:14,ConstStr(C,3));
  Write(blank:6,ConstStr(C,3),blank:7);
  Write(ConstStr(C,8));
  GotoXY(13,6); Write(ConstStr(C,12));
  Write(blank:9,ConstStr(C,11),blank:9);
  Write(ConstStr(C,12));
  GotoXY(14,7); Write(ConstStr(C,12));
  Write(blank:8,ConstStr(C,9),blank:12);
  Write(ConstStr(C,12));
  GotoXY(19,8); Write(ConstStr(C,8));
  Write(blank:7,ConstStr(C,3),blank:23);
  Write(ConstStr(C,8));
  GotoXY(13,9); Write(ConstStr(C,3),blank:6,ConstStr(C,4));
  Write(blank:8,ConstStr(C,3),blank:17);
  Write(ConstStr(C,3),blank:6,ConstStr(C,4));
  GotoXY(14,10); Write(ConstStr(C,11));
  Write(blank:9,ConstStr(C,3),blank:18);
  Write(ConstStr(C,11));
  GotoXY(15,11); Write(ConstStr(C,2),blank:2,ConstStr(C,4));
  Write(blank:11,ConstStr(C,3),blank:19);
  Write(ConstStr(C,2),blank:2,ConstStr(C,4));
  DrawBox(20,15,60,23,'Shadow Pattern Simulator');
  GotoXY(26,17); Write('Written by: ',Author);
  GotoXY(26,19); Write(' Version 2.0');
  GotoXY(21,21); Write('Copyright (C) 1986 by ',Author);
  GotoXY(26,24); Write('Press any key to continue...');
  Read(kbd,ch)
end {procedure FrontPage};
procedure init_all;
begin
  FillChar(DateGlb, SizeOf(DateGlb), 0);
  FillChar(LocGlb, SizeOf(LocGlb), 0);
  FillChar(BldGlb, SizeOf(BldGlb), 0);
  FillChar(SunGlb, SizeOf(SunGlb), 0);
  StartGlb := nil;
  LastGlb := nil;
  FirstDispGlb := nil;
  LastDispGlb := nil;
  SavedGlb := true;
  NewLoc := true;
  NewDate := true;
  FileNameGlb := '';
  GrdGlb := false;
  DrawGlb := false;
  OutHrAve := false;
  PlotGlb := false;
  PrintGlb := false;
  DelayGlb := 0;
  Time1Str := '';
  Time2Str := '';
  Time1 := 0;
  Time2 := 0;
  sec := '@';
  GetDir(0, CurrDirGlb);
  OverlayPathGlb := CurrDirGlb;
  CommandLineGlb := '';
  LastNumGlb := 0;
  AsciiGlb := false;
  CurBlnk
end;

(*
  Main procedure
*)

begin
  init_all;
  FrontPage;
  choice := 1;
  repeat
    EdMode := false;
    C1rScr;
    DrawBox(28,5,53,14,'S P S MENU');
    GotoXY(32,7); Write('Change default drive');
GotoXY(32,8); Write('Editor');
GotoXY(32,9); Write('Set options');
GotoXY(32,10); Write('Trees planted');
GotoXY(32,11); Write('Do simulation');
GotoXY(32,12); Write('Quit');
choice:= Query(6,7,30,choice,AcceptM);
if choice in [2..5] then OvrPath(OverlayPathGlb);
case choice of
  1: GetNewDrive;
  2: SPS_Editor;
  3: SetOptions;
  4: DispSite;
  5: begin
      if NewDate or NewLoc then AzmElv;
      if not(NewDate or NewLoc) then Simulator;
    end;
  6: Quit;
end;
until choice = 6
end.

(*
 Editor Module
*)

overlay procedure SPS_Editor;

(*
 This procedure gets the address (segment and offset) of
 the current Disk Transfer Area using DOS function call
 hex 2F.
 After the call the DTA segment is returned in the ES
 register and the DTA offset is returned in the BX
 register.
 The DTA is used by DOS as a buffer for disk I/O. It is
 128 bytes long and is normally located at offset hex 80
 in the program segment prefix.
*)

Procedure Get_Disk_Transfer_Area_Address(var DTA_segment,
  DTA_offset: integer);
var
  registers: result;
begin
  With registers do begin
    AH := $2F; ( Place the function call number in AH. )
    MsDos(registers); ( Invoke the function. )
    DTA_segment := ES; ( Save the Disk Transfer Area
                         segment. )
    DTA_offset := BX ( Save the Disk Transfer Area
This procedure finds the first entry in the specified directory or subdirectory that matches the specified file name. The calling parameters are as follows.

FIND_FIRST_TEMPLATE: A string of upto 64 characters which must contain the path including the file name template to be found. For example,
if find_first_template contains b:\xxxxx\*.inc the subdirectory xxxxx on drive b: will be searched for the first file with the extension .inc.

FIND_FIRST_ATTRIBUTE: A byte which specifies the attribute(s) of the files to be found. Each of the low order six bits in the attribute byte represents a different attribute as follows.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Value</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>Read Only.</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>Hidden.</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>System.</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>This entry contains the volume label.</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>This entry is a subdirectory.</td>
</tr>
<tr>
<td>32</td>
<td></td>
<td>Archive. This file has been modified since the last time it was saved using the backup command.</td>
</tr>
</tbody>
</table>

If the attribute byte is zero, only normal file entries are found. Entries for volume label, subdirectory, hidden and system entries will not be found.

If the attribute byte is set for hidden, system, or subdirectory entries an inclusive search will be performed. All normal file entries plus all entries matching the specified attribute(s) are found.

For example, to find all normal files plus all hidden and system files set the attribute to 6 (2 + 4).

If the attribute is set to 8 for the volume label, only the volume label entry will be found.

DTA SEGMENT: This is the segment address of the disk transfer area obtained by using the procedure Get_Disk_Transfer_Area_Address.
DTA_OFFSET: This is the offset address of the disk transfer area obtained by using the procedure Get_Disk_Transfer_Area_Address.

FIRST_FILENAME: The first filename found by the directory search is returned in this 12 character string.

FIRST_ERROR: Contains one of the following error codes after the call.

0 = A file name was found which matches the search criteria.
2 = File not found.
18 = No more files.

This procedure uses DOS function call hex 4E to perform the directory search. Registers DS:DX contain the address of the string containing the drive, path, and filename of the the file to be found. This string must be terminated by a byte containing hex 0.

The CX register contains the attribute byte to be used in the search.

If the DOS function call is successful the Disk Transfer Area contains the following information.

21 bytes - reserved for DOS use.
1 byte - the file's attribute.
2 bytes - the file's creation/update time.
2 bytes - the file's creation/update date.
2 bytes - low order word of the file's size.
2 bytes - high order word of the file's size.
13 bytes - the name and extension of the file followed by a byte containing hex 0. All characters following the first blank are dropped. If an extension is present it is separated from the filename by a period.

*)

Procedure Find_first_file(find_first_template :  
    find_first_template_type;  
    find_first_attribute,  
    DTA_segment, DTA_offset: integer;  
    var first_filename :  
    find_filename_type;  
    var first_error : byte);  

var
    registers: result;
i: integer;

begin

{ Insert the required null byte (hex 0) after the last character of the filename. }

find_first_template[Length(find_first_template)+1]:= Char(0);

With registers do

begin

AH:= $4E;
CX := find_first_attribute;
DS := Seg(find_first_template);
DX := Ofs(find_first_template) + 1;
MsDos(registers);

If Flags and 1 = 1 then {If the carry flag is on an error }

first_error := AL { has occurred. Save the error code. }

else begin

first_error := 0; { No error. Set error code to zero. }

{ Extract file name from the Disk Transfer Area. }
i := 30;
first_filename := '';  
While Mem[DTA_segment:DTA_offset + i] <> 0 do
begin

first_filename:= first_filename +
Char(Mem[DTA_segment:DTA_offset + i]);
i:= i + 1;
end;
end {If Flags}
end {With registers}
end;

(*

This procedure finds the next entry in the specified directory or subdirectory that matches the file name specified in a previous Find_first_file call. The calling parameters are as follows.*

| DTA_SEGMENT: |
| DTA_OFFSET: |
| NEXT_FILENAME: These parameters have the same meaning |
| NEXT_ERROR: | as the corresponding parameters in the procedure Find_first_file. |

| -----> | after the call. |
0 = A file name was found which matches the search criteria.
18 = No more files.

This procedure uses DOS function call hex 4F to continue the directory search.

If the DOS function call is successful the Disk Transfer Area contains the same information as after Find_first_file.

*)

Procedure Find_next_file(DTA_segment, DTA_offset : integer;
var next_filename :
find_filename_type;
var next_error : byte);

var
  registers: result;
i: integer;
begin
  With registers do
    begin
      AH := $4F; { Put the function number in AH }
      MsDos(registers); { Invoke the function. }
      If Flags and 1 = 1 then {If the carry flag is on an error }
        next_error := AL { has occurred. Save the error code.}
      else begin
        next_error := 0; { No error. Set error code to zero.}
        i := 30;
        next_filename := '';
        While Mem[DTA_segment:DTA_offset + i] <> 0 do
          begin
            next_filename := next_filename +
            Char(Mem[DTA_segment:DTA_offset + i]);
            i := i + 1;
          end
        end {If Flags}
      end {With registers}
    end
end;

(*
  Test if a bit set
*)

function TestBit(var target;
bit_num: integer): boolean;

var
  s : integer absolute target;
  d : integer;
begin
  d:= s;
  d:= d shr bit_num;
  TestBit:= odd(d)
end;

(*
   Set a bit on
*)

procedure SetBit(var target;
  bit_num: integer);

var
  s : integer absolute target;
  m : integer;
begin
  m:= 1 shl bit_num;
  s:= s or m
end;

(*
   Set a bit off
*)

procedure ClearBit(var target;
  bit_num : integer);

var
  s : integer absolute target;
  m : integer;
begin
  m:= not(1 shl bit_num);
  s:= s and m
end;

(*
   Obtain date input
*)

procedure InputDate(var Date: DateRec);
const
term : CharSet = [ ^E, ^I, ^M, ^X, ^Z ];

var
  L,code : integer;
  TC : Char;
  T : Str7;
begin
L:= 1;
NewDate:= true;
ClearInstWindow;
Msg(1,'Date: All information must be entered.');
Msg(3,' <End> or < ~ Z> to exit.');
with Date do
repeat
  case L of
    1: begin
      Msg(2,'Year must be >= 1980');
      if TestBit(chk,0) then
        str(year:4:0,T)
      else T:= '';
      StripLeadingSpaces(T);
      repeat
        InputStr(T,4,8,8,term,['0' .. '9'],TC);
        if T <>'' then
          begin
            val(T,year,code);
            if year >= 1980.0 then
              SetBit(chk,0)
            else error('Year must be greater than 1980')
          end
        else ClearBit(chk,0)
      until (year >= 1980.0) or (T = '')
    end;
    2: begin
      Msg(2,'Month in numeric format (i.e.,Jan = 1)');
      if TestBit(chk,1) then
        str(month:2:0,T)
      else begin
        T := ',
        month:= 1.0
      end;
      StripLeadingSpaces(T);
      repeat
        InputStr(T,2,8,9,term,['0' .. '9'],TC);
        if T <>'' then
          begin
            val(T,month,code);
            if (month >= 1.0) and (month <= 12.0) then
              SetBit(chk,1)
            else error('Month does not exist')
          end
        else ClearBit(chk,1)
      until ((month >= 1.0) and (month <= 12.0))
          or (T = '')
    end;
    3: begin
      Msg(2,'Day of the month');
if TestBit(chk,2) then
  str(day:2:0,T)
else T:= '';
StripLeadingSpaces(T);
repeat
  InputStr(T,2,8,10,term,['0'..'9'],TC);
  if T <> '' then
    begin
      val(T,day,code);
      if (trunc(day) in
          [1..Days_in_month[trunc(month)]])) then
        SetBit(chk,2)
      else
        error('Day does not exist')
    end
  else ClearBit(chk,2)
  until (trunc(day) in
         [1..Days_in_month[trunc(month)]])
       or (T = ' ')
  end;
  if TC in ['A I','A M','A X'] then
    if L = 3 then L:= 1
    else L:= L+1
  else if TC = 'A E' then
    if L = 1 then L:= 3
    else L:= L-1
  until (TC = 'A M') and (L = 1) or (TC = 'A Z')
end;

(*
  Obtain Location information
  *)

procedure InputLoc(var Loc: LocRec);
const
term: CharSet = ['A E','A I','A M','A X','A Z'];
var
  L,code: integer;
  TC: Char;
  T: Str7;
begin
  L:= 1;
  ClearInstWindow;
  Msg(1,'Location: All information must be entered.');
  Msg(3,'<End> or <^Z> to exit.');
  NewLoc:= true;
  with Loc do
    repeat
      case L of
      end;
end;
1: begin
  if TestBit(chk,0) then
    str(lat:7:2,T)
  else T:= ''; StripLeadingSpaces(T);
  Inputstr(T,7,18,2,term,['0'..'9','.','],TC);
  if T <> '' then
    begin
      val(T,lat,code);
      SetBit(chk,0)
    end
  else ClearBit(chk,0)
end;
2: begin
  if TestBit(chk,1) then
    str(long:7:2,T)
  else T:= ''; StripLeadingSpaces(T);
  Inputstr(T,7,18,3,term,['0'..'9','.','],TC);
  if T <> '' then
    begin
      val(T,long,code);
      SetBit(chk,1)
    end
  else ClearBit(chk,1)
end;
3: begin
  Msg(1,'Atlantic Std. time = 4
        Eastern Std. time = 5');
  Msg(2,'Central Std. time = 6
        Mountain Std. time = 7');
  Msg(3,'Pacific Std. time = 8
        Hawaii Std. time = 9');
  if TestBit(chk,2) then
    str(ZTime:1:0,T)
  else T:= ''; StripLeadingSpaces(T);
  Inputstr(T,1,18,4,term,['4'..'9'],TC);
  if T <> '' then
    begin
      val(T,ZTime,code);
      SetBit(chk,2)
    end
  else ClearBit(chk,2);
  ClearInstWindow;
  Msg(1,'Location: All information must be entered.');
  Msg(3,'<End> or <^Z> to exit.');
end;
4: begin
  ClearInstWindow;
Msg(2,' Is daylight saving time in effect? ');
Msg(3,' Respond with "Y" or "N". '); if TestBit(chk,3) then
  T:= DLS
else T:= '';
InputStr(T,1,18,5,term,['Y','N'],TC);
if T <> '' then
begin
  DLS:= T[1];
  SetBit(chk,3)
end
else ClearBit(chk,3);
ClearInstWindow;
Msg(1,'Location: All information must be entered.');
Msg(3,' <End> or <^Z> to exit.');
end;

end;

if TC in ['I','M','X'] then
  if L = 4 then L:= 1
else L:= L+1
else if TC = 'E' then
  if L = 1 then L:= 4
else L:= L-1
until (TC = 'M') and (L = 1) or (TC = 'Z')
end;

(* Obtain building information *)

procedure InputBld(var Bld: BldRec);
const
  term : CharSet = ['E','I','M','X','Z'];
var
  L,code : integer;
  TC : Char;
  T : Str7;
begin
  L:= 1;
  ClearInstWindow;
  Msg(1,'Building: All information must be entered.');
  Msg(3,' <End> or <^Z> to exit.');
  with Bld do
  repeat
    case L of
      1: begin
        Msg(2,' Length of section A. ');
        if TestBit(chk,0) then
          str(length:7:2,T)
        else T:= '';
StripLeadingSpaces(T);
InputStr(T,7,48,1,term,['0'..'9','.'],TC);
if T <> '' then
begin
  val(T,length,code);
  SetBit(chk,0)
end
else ClearBit(chk,0)
end;

2: begin
Msg(2,' Length of section B. ');
if TestBit(chk,1) then
  str(width:7:2,T)
else T:='';
StripLeadingSpaces(T);
InputStr(T,7,48,2,term,['0'..'9','.'],TC);
if T <> '' then
begin
  val(T,width,code);
  SetBit(chk,1)
end
else ClearBit(chk,1)
end;

3: begin
Msg(2,' Height of section A. ');
if TestBit(chk,2) then
  str(height:7:2,T)
else T:='';
StripLeadingSpaces(T);
InputStr(T,7,48,3,term,['0'..'9','.'],TC);
if T <> '' then
begin
  val(T,height,code);
  SetBit(chk,2)
end
else ClearBit(chk,2)
end;

4: begin
Msg(2,' Angle >= 0 and < 90 Deg. ');
if TestBit(chk,3) then
  str(RoofAng:5:2,T)
else T:='';
StripLeadingSpaces(T);
repeat
  InputStr(T,5,48,4,term,['0'..'9','.'],TC);
  if T <> '' then
  begin
    val(T,RoofAng,code);
    if (RoofAng >= 0.0) or (RoofAng < 90.0) then
      SetBit(chk,3)
    else
error('Angle out of range')
end
else ClearBit(chk,3)
until (RoofAng >= 0.0) or (RoofAng < 90.0)
or (T = '')
end;
5: begin
Msg(2,' Angle in Deg. (Counter-Clock-Wise +)');
if TestBit(chk,4) then
str(DispAng:7:2,T)
else T:= ' ';
StripLeadingSpaces(T);
InputStr(T,7,48,5,term,['0'..'9','.','','--'],TC);
if T <> '' then
begin
val(T,DispAng,code);
SetBit(chk,4)
else ClearBit(chk,4)
end;
end;
if TC in [^I,^M,^X] then
if L = 5 then L:= 1
else L:= L+1
else if TC = ^E then
if L = 1 then L:= 5
else L:= L-1
until (TC = ^M) and (L = 1) or (TC = ^Z)
end;

(*
Obtain tree information
*)

procedure InputTree(var Tree: ObjRec);
const
term : CharSet = [^E,^I,^M,^X,^Z];
var
L,code : integer;
TC : Char;
T : Str7;
begin
L:= 1;
with Tree do
repeat
  case L of
  1: begin
    Msg(2,' Enter "P","E",or "C"
if TestBit(chk,0) then
  T:= shape
else T := ''; StripLeadingSpaces(T); InputStr(T, 1, 28, 16, term, ['P', 'E', 'C'], TC); if T <> '' then begin shape := T[1]; SetBit(chk, 0) end else ClearBit(chk, 0) end;

2: begin
    Msg(2, '');
    if TestBit(chk, 1) then 
        str(Base:7:2, T)
    else T := ';
    StripLeadingSpaces(T);
    InputStr(T, 7, 28, 17, term, ['0'..'9', '.'], TC);
    if T <> '' then begin
        val(T, base, code);
        SetBit(chk, 1)
    end else ClearBit(chk, 1)
end;

3: begin
    Msg(2, '');
    if TestBit(chk, 2) then 
        str(top:7:2, T)
    else T := ';
    StripLeadingSpaces(T);
    InputStr(T, 7, 28, 18, term, ['0'..'9', '.'], TC);
    if T <> '' then begin
        val(T, top, code);
        SetBit(chk, 2)
    end else ClearBit(chk, 2)
end;

4: begin
    Msg(2, '');
    if TestBit(chk, 3) then 
        str(dia:7:2, T)
    else T := ';
    StripLeadingSpaces(T);
    InputStr(T, 7, 28, 19, term, ['0'..'9', '.'], TC);
    if T <> '' then begin
        val(T, dia, code);
        SetBit(chk, 3)
    end else ClearBit(chk, 3)
end;

5: begin
  Msg(2,'Value must be >= 0 and <= 1');
  if TestBit(chk,4) then
    str(Xo:7:2,T)
  else T:= ''; StripLeadingSpaces(T);
  InputStr(T,7,28,20,term,['0'..'9','.','-'],TC);
  if T <> '' then
    begin
      val(T,Xo,code);
      SetBit(chk,4)
    end
  else ClearBit(chk,4)
end;

6: begin
  Msg(2,'Value must be >= 0 and <= 1');
  if TestBit(chk,5) then
    str(Yo:7:2,T)
  else T:= ''; StripLeadingSpaces(T);
  InputStr(T,7,28,21,term,['0'..'9','.','-'],TC);
  if T <> '' then
    begin
      val(T,Yo,code);
      SetBit(chk,5)
    end
  else ClearBit(chk,5)
end;

7: begin
  Msg(2,'Value must be >= 0 and <= 1');
  if TestBit(chk,6) then
    str(pct:4:2,T)
  else T:= ''; StripLeadingSpaces(T);
  repeat
    InputStr(T,4,28,22,term,['0'..'9','.','-'],TC);
    if T <> '' then
      begin
        val(T,pct,code);
        if pct <= 1.0 then SetBit(chk,6)
        else error('Invalid shading coefficient')
      end
    else ClearBit(chk,6)
  until (pct <= 1.0) or (T = '')
end;

if TC in ['^I','^M','^X'] then
  if L = 7 then L:= 1
  else L:= L+1
else if TC = '^E' then
if L = 1 then L := 7
else L := L - 1
until (TC = ^M) and (L = 1) or (TC = ^Z)
end;

(*
Find tree in list
*)

function Find(lnum : integer): ObjPointer;
var
  i : ObjPointer;
begin
  i := startGlb;
  Find := nil;
  while i <> nil do
    begin
      if lnum = i^.num then
        begin
          Find := i;
          exit
        end;
      i := i^.next
    end
end;

(*
Renumber list
*)

procedure PatchUp(Lnum, incr : integer);
var
  i : ObjPointer;
begin
  i := Find(lnum);
  while i <> nil do
    begin
      i^.num := i^.num + incr;
      i := i^.next
    end
end;

(*
Store tree in the list
*)

function DLS_Store( info, start : ObjPointer;
                        var last : ObjPointer): ObjPointer;
var
  old, top : ^ObjRec;
  done : boolean;
begin
  top:= start;
  old:= nil;
  done:= false;
  if start = nil then
  begin
    info^.next:= nil;
    last:= info;
    info^.prior:= nil;
    DLS_Store:= info
  end
  else begin
    while(start <> nil) and (not done) do
    begin
      if start^.num < info^.num then
      begin
        old:= start;
        start:= start^.next
      end
      else begin
        if old<> nil then
        begin
          old^.next:= info;
          info^.next:= start;
          start^.prior:= old;
          DLS_Store:= top; {Keep same starting point}
          done:= true
        end
        else begin
          info^.next:= start; {New first element}
          info^.prior:= nil;
          DLS_Store:= info;
          done:= true
        end
      end
    end;
    if not done then
    begin
      last^.next:= info; {Goes on end}
      info^.next:= nil;
      info^.prior:= last;
      last:= info;
      DLS_Store:= top
    end
  end
end;

(*
  Remove tree from the list
*)
function DL_Delete(start : ObjPointer;
    key : byte ) : ObjPointer;

var
    temp,temp2 : ObjPointer;
    done : boolean;
begin
    if start^.num = key then
        begin
            DL_Delete:= start^.next;
            if temp^.next <> nil then
                begin
                    temp:= start^.next;
                    temp^.prior:= nil
                end;
            dispose(start);
        end
    else begin
        done:= false;
        temp:= start^.next;
        temp2:= start;
        while (temp <> nil) and (not done) do
            begin
                if temp^.num = key then
                    begin
                        temp2^.next:= temp^.next;
                        if temp^.next <> nil then
                            temp^.next^.prior:= temp2;
                        done:= true;
                        lastGlb:= temp^.prior;
                        dispose(temp);
                    end
                else begin
                    temp2:= temp;
                    temp:= temp^.next
                end
            end;
        DL_Delete:= start;
        if not done then error('Not found')
        else PatchUp(key+1,-1)
    end
end;

(*
  Clear menu window
*)

procedure ClearEdMenuForm;
var
    i : integer;
begin
    for i:= 2 to 6 do
begin
  GotoXY(64,i); Write('':14)
end
end;

(*
  Clear building window
*)
procedure ClearBldForm;
var
  i : integer;
begin
  for i:= 2 to 6 do
    begin
      GotoXY(49,i); Write('':7)
    end
end;

(*
  Clear date window
*)
procedure ClearDateForm;
var
  i : integer;
begin
  for i:= 9 to 11 do
    begin
      GotoXY(9,i); Write('':7)
    end
end;

(*
  Clear location window
*)
procedure ClearLocForm;
var
  i : integer;
begin
  for i:= 3 to 6 do
    begin
      GotoXY(19,i); Write('':7)
    end
end;

(*
  Clear tree input screen fields
*)
procedure ClearTreeForm;
var
  i : integer;
begin
  GotoXY(9,15); Write('':4);
  for i:= 16 to 23 do
    GotoXY(29,i); Write('':7)
end;

procedure ClearForestForm;
var
  i : integer;
begin
  for i:= 16 to 23 do
    GotoXY(2,i); Write('':76)
end;

procedure ClearTreeWindow;
var
  i : integer;
begin
  for i:= 15 to 23 do
    GotoXY(2,i); Write('':77)
end;

procedure ClearDate;
begin
  FillChar(DateGlb,SizeOf(DateGlb),0);
  ClearDateForm
end;

procedure ClearLoc;
begin
  FillChar(LocGlb,SizeOf(LocGlb),0);
  ClearLocForm
end;

procedure ClearBld;
begin
  FillChar(BldGlb,SizeOf(BldGlb),0);
  ClearBldForm
end;

procedure ClearForest;
var
  i : integer;
temp : ObjPointer;
begin
while startGlb <> nil do
begin
  temp := startGlb^.next;
  dispose(startGlb);
  startGlb := temp
end;
lastGlb := nil; startGlb := nil;
FirstDispGlb := nil;
LastDispGlb := nil;
lastNumGlb := 0;
ClearForestForm
end;

procedure ClearAll;
begin
  ClearDate;
  ClearLoc;
  ClearBld;
  ClearForest
end;

procedure OutBld;
begin
  with BldGlb do
  begin
    GotoXY(49,2);
    if TestBit(chk,0) then Write(lngth:1:2);
    GotoXY(49,3);
    if TestBit(chk,1) then Write(width:1:2);
    GotoXY(49,4);
    if TestBit(chk,2) then Write(hgt:1:2);
    GotoXY(49,5);
    if TestBit(chk,3) then Write(RoofAng:1:2);
    GotoXY(49,6);
    if TestBit(chk,4) then Write(DispAng:1:2);
  end
end;

procedure OutDate;
begin
  with DateGlb do
  begin
    GotoXY(9,9); if TestBit(chk,0) then Write(Year:1:0);
    GotoXY(9,10); if TestBit(chk,1) then Write(Month:1:0);
    GotoXY(9,11); if TestBit(chk,2) then Write(day:1:0);
  end
end;

procedure OutLoc;
begin
  with LocGlb do
  begin
    GotoXY(19,3); if TestBit(chk,0) then Write(Lat:1:2);
    GotoXY(19,4); if TestBit(chk,1) then Write(Long:1:2);
    GotoXY(19,5); if TestBit(chk,2) then Write(ZTime:1:0);
    GotoXY(19,6); if TestBit(chk,3) then Write(DLS);
  end end;

procedure OutTree(Tree : ObjRec);
begin
  with tree do
  begin
    GotoXY(9,15); Write(num);
    GotoXY(29,17); Write(shape);
    GotoXY(29,18); Write(Base:1:2);
    GotoXY(29,19); Write(top:1:2);
    GotoXY(29,20); Write(dia:1:2);
    GotoXY(29,21); Write(Xo:1:2);
    GotoXY(29,22); Write(Yo:1:2);
    GotoXY(29,23); Write(pct:1:2);
  end end;

procedure OutBldForm;
begin
  GotoXY(29,2); Write('House length........');
  GotoXY(29,3); Write('House width........');
  GotoXY(29,4); Write('Wall height.........');
  GotoXY(29,5); Write('Roof angle.........');
  GotoXY(29,6); Write('House displacement..');
end;

procedure OutDateForm;
begin
  GotoXY(2,9);  Write('Year....');
  GotoXY(2,10); Write('Month..');
  GotoXY(2,11); Write('day....');
end;

procedure OutLocForm;
begin
  GotoXY(2,3);  Write('Latitude.........');
  GotoXY(2,4);  Write('Longitude.........');
  GotoXY(2,5);  Write('Zone Time.........');
  GotoXY(2,6);  Write('Daylight Saving..');
end;

procedure OutForestForm;
begin
GotoXY(2,15);
Write('Tree Shape Bole Hgt Crown Hgt Crown Dia X Coord Y Coord');
Write(' Shade Coef')
end;

procedure OutTreeForm;
begin
  GotoXY(3,15); Write('Tree #');
  GotoXY(3,17); Write('Tree shape.................');
  GotoXY(3,18); Write('Bole height...................');
  GotoXY(3,19); Write('Crown height..................');
  GotoXY(3,20); Write('Crown diameter.................');
  GotoXY(3,21); Write('Tree X coordinate.............');
  GotoXY(3,22); Write('Tree Y coordinate.............');
  GotoXY(3,23); Write('Tree shading coefficient..');
end;

function Open(var f: filetype; n: Str12; c: Char ): boolean;
begin
  {$I-}
  assign(f,n+'.SPS');
  IOcheck;
  if IOstat = 0 then
  begin
    case c of
      'O' : Rewrite(f,MaxRecsize);
      'I' : Reset(f,MaxRecsize);
    end;
    IOcheck;
    Open:= IOstat = 0;
    if IOstat <> 0 then
    begin
      close(f);
      IOcheck
    end
  end
  {$I+}
end;

procedure GetRec(var f : Filetype;
  var x: integer;
  var ok : boolean);
var Buffer : array[1..MaxRecSize] of byte;
begin
  {$I-}
  seek(f,R);
  {$I+}
end;
IOcheck;
if IOstat = 0 then
begin
  BlockRead(f,Buffer,1);
  IOcheck;
  if IOstat = 0 then
    case R of
      0: move(Buffer,x,SizeOf(DateGlb));
      1: move(Buffer,x,SizeOf(LocGlb));
      2: move(Buffer,x,SizeOf(BldGlb));
      else move(Buffer,x,SizeOf(ObjRec))
    end;
  ok:= IOstat = 0
{$I+}
end;
end;

procedure PutRec(var f : FileType;
  var x;
  R : integer;
  var ok : boolean);{
var
  Buffer : array[1..MaxRecSize] of byte;
begin
{$I-}
  seek(f,R);
  IOcheck;
  if IOstat = 0 then
    begin
      case R of
        0: move(x,Buffer,SizeOf(DateGlb));
        1: move(x,Buffer,SizeOf(LocGlb));
        2: move(x,Buffer,SizeOf(BldGlb));
        else move(x,Buffer,SizeOf(ObjRec))
      end;
      BlockWrite(f,Buffer,1);
      IOcheck
    end;
  ok:= IOstat = 0
{$I+}
end;
end;

procedure Display(first : ObjPointer);
var
  ci : integer;
  cr : real;
begin
  ClearForestForm;
  FirstDispGlb:= first;
  ci:= 1; cr:= 1.0;
  while (first <> nil) and (int(ci div 7) <> (cr/7.0)) do
with first^ do
begin
  GotoXY(2,ci+16);
  Write(num:3);
  Write(shape:6);
  Write(base:12:2);
  Write(top:11:2);
  Write(dia:11:2);
  Write(Xo:10:2);
  Write(Yo:10:2);
  Write(pct:12:2);
  first:= first^.next;
  ci:= ci+1; cr:= cr+1.0
end;
if first <> nil then
  LastDispGlbl:= first^.next
else
  LastDispGlbl:= startGlbl
end;

(*
  Query the user for tree to be removed
*)

procedure Remove;
var
  num, code : integer;
  T : Str7;
  TC : Char;
begin
  T:= '';
  ClearInstWindow;
  Msg(1,'Remove: Enter the number of the tree to be removed');
  Msg(2,' followed by <Ret>. Use <Esc> to cancel.');
  Msg(3,'Tree number =');
  InputStr(T,3,42,10,\^[M]+AbortSet,\['0'..'9'],TC);
  if TC in AbortSet then exit;
  StripLeadingSpaces(T);
  val(T,num,code);
  startGlbl:= DL_Delete(startGlbl,num);
  Display(FirstDispGlbl)
end;

(*
  Add a new tree into the list
*)

procedure Add;
label
  exitproc;
var
  info : ObjPointer;
  num : integer;
  done : boolean;
begin
  done:= false;
  num:= LastNumGlb+1;
  ClearInstWindow;
  Msg(1,'Add: All information must be entered.');
  Msg(3,' <End> or <^Z> with no entry to exit.');
  ClearTreeWindow;
  OutTreeForm;
  repeat
    if RamSp < int(SizeOf(info^)) then
      begin
        error('Out of memory space');
        goto exitproc;
      end
    else begin
      new(info);
      fillchar(info^,sizeof(info^),0);
      info^.num:= num;
      ClearTreeForm;
      GotoXY(9,15); Write(num);
      InputTree(info^);
      if info^.chk = 0 then done:= true
    else begin
      if Find(num) <> nil then PatchUp(num,1);
      startGlb:= DLS_store(info,startGlb,lastGlb);
      num:= num+1
    end
  end
  until done;
  dispose(info);
exitproc:
  LastNumGlb:= num;
  ClearTreeWindow;
  OutForestForm;
  FirstDispGlb:= startGlb;
  Display(FirstDispGlb)
end;

(*
  Edit a specific tree
*)

procedure Update;
var
  num,code : integer;
  s : String[3];
TC : Char;
info : ObjPointer;
begin
ClearInstWindow;
Msg(1,'Modify: Enter tree to be edit followed by <Ret> ');
Msg(2,' Press <Esc> to cancel.' );
Msg(3,'Tree number =');
s:= '';
InputStr(s,3,42,10,^[M]+AbortSet,[0..9],TC);
if TC in AbortSet then exit;
ClearInstWindow;
Msg(1,'Modify: ');
Msg(3,' <End> or <A Z> to exit');
val(s,num,code);
info:= Find(num);
if info <> nil then
begin
  ClearTreeWindow;
  OutTreeForm;
  OutTree(info);
  InputTree(info);
  if info^.chk = 0 then
    startGlb:= DL_Delete(startGlb,num);
  ClearTreeWindow;
  OutForestForm;
  firstDispGlb:= startGlb;
  Display(FirstDispGlb)
end
else error('Tree not found')
end;

(*
  Save information on a disk file
*)

procedure Save(var Filename : Str12;
               start : ObjPointer);
var
  f         : FileType;
  ok        : boolean;
  TC        : Char;
  FN        : Str12;
  protect, unprotect: boolean;
  err       : byte;
begin
ClearInstWindow;
protect:= false; unprotect:= false;
Msg(1,'Save: /U = Unprotect, /P = Protect ');
Msg(2,' <Esc> to abort command.');
Msg(3,'Destination file =');
InputStr(FileName, 12, 47, 10, [^M]+AbortSet, FileChar, TC);
if TC in AbortSet then exit;
if pos(’/U’, FileName) > 0 then
begin
   unprotect := true;
   delete(FileName, pos(’/U’, FileName), 2);
end;
if pos(’/P’, FileName) > 0 then
begin
   protect := true;
   delete(FileName, pos(’/P’, FileName), 2);
end;
while (FileName <> '') and (pos(’/’, FileName) > 0) do
   delete(FileName, pos(’/’, FileName), 1);
if FileName = '' then
begin
   error(’Invalid file name.’);
   exit
end;
FN := FileName + ’.sps’;
if not exist(FN) and (DskSpc(’*’) < int((LastGlb^.*.num+3)*MaxRecSize)) then
begin
   error(’Disk is full.’);
   exit
end;
if Unprotect then
   change_file_attribute(’r’, ’c’, FN, err);
if Open(f, FileName, ’O’) then
begin
   ClearInstWindow;
   Msg(1, ’Saving file...’);
   PutRec(f, DateGlb, 0, ok);
   if ok then
      begin
         PutRec(f, LocGlb, 1, ok);
         if ok then
            begin
               PutRec(f, BldGlb, 2, ok);
               while (start <> nil) and ok do
                  begin
                     PutRec(f, start^.start^.num+2, ok);
                     start := start^.next
                  end
            end
         end
      end;
   close(f);
   if protect then
      change_file_attribute(’r’, ’s’, FN, err);
end;
savedGlb := ok
(*
  Load information from a disk file
*)

procedure Load(var FileName : Str12;
    var start    : ObjPointer);

var
  temp    : ObjPointer;
  f       : FileType;
  ok      : boolean;
  protect : boolean;
  TC      : Char;
  FN      : Str12;
  R       : integer;
  err     : byte;
begin
  FileName:= ''; ClearInstWindow;
  Msg(1,'Load: Use cursor key to enter name and  '); if TC in AbortSet then exit;
  InputStr(FileName,8,42,[^M]+AbortSet,FileChar-['/'],TC);
  if Open(f,FileName,'I') then
    begin
      ClearInstWindow;
      Msg(l,'Loading file...');
      GetRec(f,DateGlb,0,ok);
      if ok then
        begin
          GetRec(f,LocGlb,1,ok);
          if ok then begin
            GetRec(f,BldGlb,2,ok);
            R:= 2;
            while not eof(f) and ok
              and (RamSpc > int(SizeOf(Temp^))) do
              begin
                new(temp);
                R:= R+1;
                GetRec(f,temp^,R,ok);
                if ok then begin
                  start:= DLS_Store(temp,start,lastGlb)
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;

else
    dispose(temp)
end;
savedGlbl:= true;
OutDate;
OutLoc;
OutBld;
FirstDispGlbl:= start;
Display(FirstDispGlbl)
end
end;
close(f);
if protect then
    change_file_attribute('r','s',FN,err);
end
end;

procedure VerticalTab;
var
    i : integer;
begin
    for i:= 1 to 3 do
        if PtrWorking then Writeln(lst)
end;

(*
Format and send information to a line printer
*)

procedure print;
const
    zone : array[4..9] of string[8] =
        ('Atlantic','Eastern','Central','Mountain','Pacific','Hawaii');
var
    info : ObjPointer;
    line : integer;
begin
    ClearInstWindow;
    if not PtrWorking then begin
        error('Printer not Ready. Command canceled.');
        exit
    end;
    Msg(2,'Printing...');
    VerticalTab;
    with DateGlbl do
    begin
        if PtrWorking then begin
            for i:= 1 to 3 do
                if not PtrWorking then Writeln(lst)
end;

(* Format and send information to a line printer *)

procedure print;
const
    zone : array[4..9] of string[8] =
        ('Atlantic','Eastern','Central','Mountain','Pacific','Hawaii');
var
    info : ObjPointer;
    line : integer;
begin
    ClearInstWindow;
    if not PtrWorking then begin
        error('Printer not Ready. Command canceled.');
        exit
    end;
    Msg(2,'Printing...');
    VerticalTab;
    with DateGlbl do
    begin
        if PtrWorking then begin
            for i:= 1 to 3 do
                if not PtrWorking then Writeln(lst)
end;
Write(lst,'Month.............. '); if testbit(chk,1) then     Writeln(lst,Month:1:0) else     Writeln(lst) end;
if PtrWorking then begin
    Write(lst,'Day................. '); if testbit(chk,2) then     Writeln(lst,Day:1:0) else     Writeln(lst) end;
if PtrWorking then begin
    Write(lst,'Year............. '); if testbit(chk,0) then     Writeln(lst,Year:1:0) else     Writeln(lst) end;
if PtrWorking then Writeln(lst);
end;
with LocGlb do begin
    if PtrWorking then begin
        Write(lst,'Zone Time........ '); if testbit(chk,2) then     Writeln(lst,zone[trunc(ZTime)]) else     Writeln(lst) end;
    if PtrWorking then begin
        Write(lst,'Daylight savings.... '); if testbit(chk,3) then     Writeln(lst,DLS) else     Writeln(lst) end;
    if PtrWorking then begin
        Write(lst,'Local Latitude..... '); if testbit(chk,0) then     Writeln(lst,Lat:1:2) else     Writeln(lst) end;
end; if PtrWorking then
begin
  Write(lst,'Local Longitude..... ');  
  if testbit(chk,1) then
      Writeln(lst,Long:1:2)
  else
      Writeln(lst)
  end;
  if PtrWorking then Writeln(lst);
end;
with BldGlb do
begin
  if PtrWorking then
  begin
    Write(lst,'Length of sec A..... ');
    if testbit(chk,0) then
      Writeln(lst,lngth:1:2)
    else
      Writeln(lst)
  end;
  if PtrWorking then
  begin
    Write(lst,'Length of sec B..... ');
    if testbit(chk,1) then
      Writeln(lst,wdth:1:2)
    else
      Writeln(lst)
  end;
  if PtrWorking then
  begin
    Write(lst,'Height of sec A..... ');
    if testbit(chk,2) then
      Writeln(lst,hgt:1:2)
    else
      Writeln(lst)
  end;
  if PtrWorking then
  begin
    Write(lst,'Roof angle .......... ');
    if testbit(chk,3) then
      Writeln(lst,RoofAng:1:2)
    else
      Writeln(lst)
  end;
  if PtrWorking then
  begin
    Write(lst,'Displacement angle .. ');
    if testbit(chk,4) then
      Writeln(lst,DispAng:1:2)
    else
      Writeln(lst)
  end;
if PtrWorking then Writeln(lst);
end;
if PtrWorking then
begin
  Write(lst,'Tree Shape Bole Hgt');
  Write(' Crown Hgt Crown Dia X Coord.');
  Writeln(lst,' Y Coord. Shade Coef.');
end;
line:= 16;
info:= StartGlb;
while info <> nil do with info^ do
begin
  if PtrWorking then
  begin
    Write(lst,num:3);
    if TestBit(chk,0) then
      Write(lst,shape:6)
    else
      Write(lst,':6);
    if TestBit(chk,1) then
      Write(lst;base:12:2)
    else
      Write(lst,':12:2);
    if TestBit(chk,2) then
      Write(lst,top:11:2)
    else
      Write(lst,':11:2);
    if TestBit(chk,3) then
      Write(lst,dia:11:2)
    else
      Write(lst,':11);
    if TestBit(chk,4) then
      Write(lst,Xo:10:2)
    else
      Write(lst,':10);
    if TestBit(chk,5) then
      Write(lst,Yo:10:2)
    else
      Write(lst,':10);
    if TestBit(chk,6) then
      Write(lst,pct:10:2)
    else
      Write(lst,':10);
    Writeln(lst);
  end;
  info:= info^.next;
  line:= line+1;
  if line = 55 then
  begin
    if PtrWorking then Writeln(lst,#12);
    VerticalTab;
  end;
  info:= info^.next;
  line:= line+1;
  if line = 55 then
  begin
    if PtrWorking then Writeln(lst,#12);
    VerticalTab;
  end;
line:= 0
end
end;
if PtrWorking then Writeln(lst,#12);
end (procedure print);

(*
Display disk directory
*)

procedure Directory;
var
   DTAseg, DTAoff : integer;
   search_name    : string[64];
   search_attribute : byte;
   filename       : string[12];
   err            : byte;
   ch             : char;
   escape         : boolean;
   col,row        : integer;
begin
   escape:= false;
   ClearInstWindow;
   ClearTreeWindow;
   drawbox(1,14,79,24,'DIRECTORY');
   Msg(1,'Disk directory:');
   search_name:= '*.sps';
   search_attribute := 63;  
      { Find files with any attribute }
   Get_Disk_Transfer_Area_Address(DTAseg, DTAoff);

   ( Find the first file name that meets the search criteria. )
   Find_first_file(search_name,search_attribute,
      DTAseg,DTAoff,filename,err);

   ( If the search for the first file is successful, display the file information and look for the next filename that satisfies the search criteria. )
   col:= 4;
   row:= 16;
   if err = 0 then
      begin
         Repeat
            GotoXY(col,row); Write(filename);
            col:= col+14;
            if col = 74 then
begin
  col:= 4;
  row:= row+1
end;
{ Find the next filename. }
Find_next_file(DTAseg, DTAoff, filename, err);
if ((err <> 0) or (row = 24)) then
begin
  beep;
  Read(kbd, ch);
  escape:= (Upcase(ch) = 'Q') or (err <> 0);
  row:= 16;
  ClearTreeWindow;
end
Until escape;
end {If err = 0}
else error('Files not found');
drawbox(l,14,79,24,'TREES');
OutForestForm;
Display(FirstDispGlb)
end;

(*
Setup the screen for the editor
*)

procedure SetEditorScr;
begin
  clrscr;
  drawbox(1,1,27,7,'LOCATION'); {location}
  drawbox(1,8,27,12,'DATE'); {date}
  drawbox(28,1,56,7,'BUILDING'); {building}
  drawbox(63,1,79,7,'MENU'); {menu}
  drawbox(28,8,79,12,'INSTRUCTIONS'); {instructions}
  drawbox(1,14,79,24,'TREES'); {trees}
  Msg(l,'One moment please.');
  OutLocForm;
  OutDateForm;
  OutBldForm;
  OutForestForm
end;

(*
Clear forms
*)

procedure ClearForms;
const
  Accept: CharAry = ('D','L','B','T','Q','^P','#0','#0','#0');
var
choice : integer;

begin
  choice:= 1;
  ClearInstWindow;
  Msg(1,'Clear: Will destroy the information of the');
  Msg(2,' selected window.');
  Msg(3,'  
    + <Ret> or first cap letter to select.');
  ClearEdMenuForm;
  GotoXY(69,2); Write('Date');
  GotoXY(69,3); Write('Location ');
  GotoXY(69,4); Write('Building ');
  GotoXY(69,5); Write('Trees ');
  GotoXY(69,6); Write('Quit ');
  repeat
    choice:= Query(5,2,67,choice,accept);
    case choice of
      1: ClearDate;
      2: ClearLoc;
      3: ClearBld;
      4: ClearForest;
      6: begin print; choice:= 1 end;
    end;
  until choice = 5
end;

procedure TreesMenu;
const
  Accept : CharAry = ('N','M','A','R','Q','P',0,0,0);
var
  choice : integer;

begin
  choice:= 1;
  ClearEdMenuForm;
  repeat
    ClearInstWindow;
    Msg(1,'Trees: Select appropriate option.');
    Msg(3,'  
      + <Ret> or first cap letter to select.');
    GotoXY(67,2); Write('Next page');
    GotoXY(67,3); Write('Modify ');
    GotoXY(67,4); Write('Add ');
    GotoXY(67,5); Write('Remove ');
    GotoXY(67,6); Write('Quit ');
    choice:= Query(5,2,65,choice,accept);
    case choice of
      1: Display(LastDispGlb);
      2: UpDate;
      3: Add;
      4: Remove;
      6: begin print; choice:= 1 end;
procedure Edit;
const
  Accept : CharAry = ('D','L','B','T','Q','^P,#0,#0,#0);
var
  choice : integer;
begin
  choice:= 1;
  ClearEdMenuForm;
  repeat
    ClearInstWindow;
    Msg(1,'Edit: Select appropriate option.');
    Msg(3,' '+<Ret> or first cap letter to select.');  
    gotoxy(68,2); Write('Date');
    gotoxy(68,3); Write('Location');
    gotoxy(68,4); Write('Building');
    gotoxy(68,5); Write('Trees');
    gotoxy(68,6); Write('Quit');
    choice:= Query(5,2,66,choice,accept);
    case choice of
      1: InputDate(DateGlb);
      2: InputLoc(LocGlb);
      3: InputBld(BldGlb);
      4: begin
         TreesMenu;
         ClearEdMenuForm;
      end;
      6: begin print; choice:= 1 end;
    end;
  until choice = 5
end;

procedure EdMenu;
const
  Accept : CharAry = ('C','E','L','S','Q','^P,'^D,#0,#0);
var
  choice : integer;
begin
  choice:= 1;
  repeat
    ClearInstWindow;
    Msg(1,'Editor: Select appropriate option.');
    Msg(3,' '+<Ret> or first cap letter to select.');
    gotoxy(69,2); Write('Clear');
    gotoxy(69,3); Write('CleanEdMenuForm');
    gotoxy(69,4); Write('Quit');
    gotoxy(69,5); Write('Print');
    gotoxy(69,6); Write('Close');
    gotoxy(69,7); Write('Dir');
    choice:= Query(5,2,66,choice,accept);
    case choice of
      1: ClearEdMenuForm;
      2: gotoxy(69,2); Write('Clean');
      3: gotoxy(69,3); Write('Close');
      4: TreesMenu;
      5: gotoxy(69,4); Write('Dir');
    end;
  until choice = 5
end;
GotoXY(69,3); Write('Edit ');
GotoXY(69,4); Write('Load ');
GotoXY(69,5); Write('Save ');
GotoXY(69,6); Write('Quit ');
choice:=Query(5,2,67,choice,accept);
case choice of
  1: ClearForms;
  2: Edit;
  3: Load(FileNameGlb,startGlb);
  4: Save(FilenameGlb,startGlb);
  6: begin print; choice:= 1 end;
  7: begin Directory; choice:= 1 end;
end;
until choice = 5
end;

(*
  Initialize the Editor
*)

procedure Initialize;
begin
  OutDate;
  OutLoc;
  OutBld;
  Display(FirstDispGlb)
end;

(*
  main Editors' procedure
*)

begin
  SetEditorscr;
  EdMode:=true;
  Initialize;
  EdMenu;
  EdMode:=false
end;

(*
  Display site module
*)

overlay procedure DispSite;
var
  info : ObjPointer;
  Xmax,Xmin,Ymax,Ymin : real;
  ch : char;

  function YPoint(y : real): integer;
begin
  if $y > Y_{\text{max}}$ then $y := Y_{\text{max}}$
  else if $y < Y_{\text{min}}$ then $y := Y_{\text{min}}$
  $Y_{\text{Point}} := \text{originY} + \text{Prow} - \text{round}\left(\frac{|y - Y_{\text{min}}|}{Y_{\text{max}} - Y_{\text{min}}} \times \text{Prow}\right)$
end;

function XPoint(x : real): integer;
begin
  if $X > X_{\text{max}}$ then $X := X_{\text{max}}$
  else if $X < X_{\text{min}}$ then $X := X_{\text{min}}$
  $X_{\text{Point}} := \text{originX} + \text{round}\left(\frac{|x - X_{\text{min}}|}{X_{\text{max}} - X_{\text{min}}} \times \text{Pcol}\right)$
end;

procedure DrawAxis;
begin
  draw(XPoint(0.0), OriginY, XPoint(0.0), OriginY + Prow, 1);
  draw(OriginX, YPoint(0.0), OriginX + Pcol, YPoint(0.0), 1)
end;

procedure DrawBld;
var
  ang, r, x1, y1, x2, y2, x3, y3, x4, y4 : integer;
begin
  with BldGlb do begin
    ang := ArcTan(width/length);
    r := sqrt(length*length + width*width);
    x1 := XPoint(0.0);
    y1 := YPoint(0.0);
    x2 := XPoint(length*cos((90.0 + DispAng)*rad));
    y2 := YPoint(length*sin((90.0 + DispAng)*rad));
    x3 := XPoint(r*cos((90.0 + DispAng)*rad + ang));
    y3 := YPoint(r*sin((90.0 + DispAng)*rad + ang));
    x4 := XPoint(width*cos((180.0 + DispAng)*rad));
    y4 := YPoint(width*sin((180.0 + DispAng)*rad));
    Draw(x1, y1, x2, y2, 1);
    Draw(x2, y2, x3, y3, 1);
    Draw(x3, y3, x4, y4, 1);
    Draw(x4, y4, x1, y1, 1);
  end
end;

procedure DrawTrees;
var
  x, y, r : integer;
begin
  info := StartGlb;
  while info <> nil do with info^ do begin
    end
end;
if chk = 127 then
begin
  x:= XPoint(Xo);
  y:= YPoint(Yo);
  r:= (Ypoint(Yo+dia/2.0)-y);
  draw(x-r,y-r,x+r,y-r,1);
  draw(x+r,y-r,x,y+r,1);
  draw(x,y+r,x-r,y-r,1);
  plot(x,y,1);
end;
end;

(*
  Main display site procedure
*)

begin
  if HardwarePresent then
    if BldGlb.chk = 31 then
      begin
        Hires;
        draw(142,12,622,12,1);
        draw(142,12,142,132,1);
        draw(622,12,622,132,1);
        draw(142,132,622,132,1);
        GotoXY(37,1); Write('TREES PLANTED');
        draw(75,40,75,64,l);
        GotoXY(10,3); Write('N');
        GotoXY(10,5); Write(#30);
        Xmax:= 0.0; Xmin:= 0.0;
        Ymax:= 0.0; Ymin:= 0.0;
        info:= StartGlb;
        while info <> nil do with info^ do
          begin
            if chk = 127 then
              begin
                if Xo+dia > Xmax then
                  Xmax:= Xo+dia
                else if Xo-dia < Xmin then
                  Xmin:= Xo-dia;
                if Yo+dia > Ymax then
                  Ymax:= Yo+dia
                else if Yo-dia < Ymin then
                  Ymin:= Yo-dia;
              end;
            info:= info^.next
          end;
        DrawAxis;
        DrawBld;
        DrawTrees;
repeat
    Read(kbd,ch);
    if ch = 'P then HardCopy
until ch <> 'P;
TextMode;
end
else error(‘Incomplete building information’)
else error(‘Graphics hardware not present’)
end;

(*
  Simulator Module
*)

Overlay procedure Simulator;
var
    CommandLineLoc : String[6];

procedure ParabolicObj(alpha,beta : real;
    Xo,Yo,Base,Top,Dia : real;
    var xyz : Ary5x3);
var
    p,r : real;
begin
    alpha:= alpha*rad;
    beta:= beta*rad;
    Xo:= Xo*constant;
    Yo:= Yo*constant;
    Dia:= Dia*constant;
    Base:= Base*constant;
    top:= top*constant;
    p:= sqr(Dia/2.0)/(4.0*Top);
    r:= 2.0*p*tan(beta);
    xyz[1,1]:= Xo-r*cos(alpha);
    xyz[1,2]:= Yo-r*sin(alpha);
    xyz[1,3]:= Base+Top-p*sqr(tan(beta));
    xyz[2,1]:= xyz[1,1];
    xyz[2,2]:= Yo;
    xyz[2,3]:= Base;
    xyz[3,1]:= Xo+Dia/2.0*cos(alpha);
    xyz[3,2]:= Yo+Dia/2.0*sin(alpha);
    xyz[3,3]:= Base;
    xyz[4,1]:= Xo-Dia/2.0*sin(alpha);
    xyz[4,2]:= Yo-Dia/2.0*cos(alpha);
    xyz[4,3]:= Base;
    xyz[5,1]:= Xo+Dia/2.0*sin(alpha);
    xyz[5,2]:= Yo-Dia/2.0*cos(alpha);
    xyz[5,3]:= Base
end;

procedure EllipticObj(alpha,beta : real;

var
  majorx, minorx, r : real;
begin
  alpha := alpha * rad;
  beta := beta * rad;
  Xo := Xo * constant;
  Yo := Yo * constant;
  Dia := Dia * constant;
  Base := Base * constant;
  top := top * constant;
  majorx := Top / 2.0;
  minorx := Dia / 2.0;
  r := sqrt(minorx) * tan(beta) / sqrt(sqrt(majorx) +
            sqrt(tan(beta) * minorx));
  xyz[1,1] := Xo - r * cos(alpha);
  xyz[1,2] := Yo - r * sin(alpha);
  xyz[1,3] := majorx + Base + majorx * sqrt(1.0 - sqrt(r / minorx));
  xyz[2,1] := xyz[1,1];
  xyz[2,2] := Yo;
  xyz[2,3] := Base + majorx;
  xyz[3,1] := Xo + r * cos(alpha);
  xyz[3,2] := Yo + r * sin(alpha);
  xyz[3,3] := majorx + Base - majorx * sqrt(1.0 - sqrt(r / minorx));
  xyz[4,1] := Xo - Dia / 2.0 * sin(alpha);
  xyz[4,2] := Yo + Dia / 2.0 * cos(alpha);
  xyz[4,3] := Base + majorx;
  xyz[5,1] := Xo + Dia / 2.0 * sin(alpha);
  xyz[5,2] := Yo - Dia / 2.0 * cos(alpha);
  xyz[5,3] := Base + majorx
end;

procedure CylindricalObj(alpha, beta : real;
                         Xo, Yo, Base, Top, Dia : real;
                         var xyz : Ary5X3);

var
  r : real;
begin
  alpha := alpha * rad;
  beta := beta * rad;
  Xo := Xo * constant;
  Yo := Yo * constant;
  Dia := Dia * constant;
  Base := Base * constant;
  top := top * constant;
  r := Dia / 2.0;
  xyz[1,1] := Xo - r * cos(alpha);
  xyz[1,2] := Yo - r * sin(alpha);
  xyz[1,3] := Base + Top;
  xyz[2,1] := xyz[1,1];
xyz[2,2] := Yo;
xyz[2,3] := Base;
xyz[3,1] := Xo+r*cos(alpha);
xyz[3,2] := Yo+r*sin(alpha);
xyz[3,3] := Base;
xyz[4,1] := Xo-Dia/2.0*sin(alpha);
xyz[4,2] := Yo+Dia/2.0*cos(alpha);
xyz[4,3] := Base;
xyz[5,1] := Xo+Dia/2.0*sin(alpha);
xyz[5,2] := Yo-Dia/2.0*cos(alpha);
xyz[5,3] := Base
end;

procedure BottomShd( z,t1,t2,t3,t4 : real;
S : Ary5x3;
var yl, yr
var a,b,c,major,minor : real;
t1,tr,h,k : real;
err : boolean;
begin
if AsciiGlb then Writeln('bottom');
minor := dist(S[2,2],S[2,3],S[3,2],S[3,3]);
major := dist(S[4,2],S[4,3],S[2,2],S[2,3]);
h := S[2,2]; k := z-S[2,3];
a := sqr(minor*t1)+sqr(major*t3);
b := 2.0*k*(sqr(minor)*t1*t2+sqr(major)*t3*t4)-2.0*h*a;
c := sqr(minor*k*t2)+sqr(major*k*t4)+sqr(h)*a -
h*(2.0*k*(sqr(minor)*t1*t2 +
sqr(major)*t3*t4)) - sqr(major*minor);
quad(a,b,c,yl, yr, err);
if err then
begin
  yl := -1.0;
  yr := -1.0
end
end;

procedure ParabolocShd( z : real;
S : Ary5x3;
var yl, yr : real);
var a,b,c,p,wp,w,theta,t1,t2,t3,t4 : real;
tl,tr,h,k : real;
err : boolean;
begin
wp := pi/2.0;
w := AngLin(S);
theta := Atan2((S[4,3]-S[5,3]),(S[4,2]-S[5,2]));
t1 := sin(w+theta)/sin(w);
t2 := sin(w-wp+theta)/sin(w);
t3 := \sin(-\theta)/\sin(w);
t4 := \sin(wp-\theta)/\sin(w);
h := S[1,2]; k := z-S[1,3];
p := -\sqrt{((S[4,2]-h)*t1 + (S[4,3]-S[1,3])*t2)/(4.0* ((S[4,2]-h)*t3 + (S[4,3]-S[1,3])*t4))};
a := tl*t1;
b := 2.0*k*t1*t2+4.0*p*t3-2.0*h*a;
c := \sqrt{k*t2+4.0*p*k*t4+\sqrt{h*t1} - h*(2.0*k*t1*t2+4.0*p*t3)};
quad(a,b,c,yl,yr,err);
if err then
begin
  yl := -1.0;
yr := -1.0
end;
if (S[2,3]-S[3,3] > 0.0) and ((Z < S[4,3])
or (z < S[5,3])) then
begin
  BottomShd(z,tl,t2,t3,t4,S,tl,tr);
  if (z < S[5,3]) (and (tl > yl)) then yl := tl;
  if (z < S[4,3]) (and (tr < yr)) then yr := tr
end;
if AsciiGlb then
  Writeln('z=',z:10:3,' yl=',yl:10:3,
            'yr=',yr:10:3,' O=',theta:10:3,' p=',p:10:3)
end;

procedure EllipticShd(z, S, var yl, yr, var a, b, c, p, wp, w, theta, tl, t2, t3, t4, var tl, tr, h, k, major, minor, err var minor, major);
quad(a,b,c,yl,yr,err);
if err then
begin
  yl:= -1.0;
  yr:= -1.0
end;
if (S[2,3]-S[3,3] > 0.0) and ((z < S[4,3]) or (z < S[5,3])) then
begin
  BottornShd(z,tl,t2,t3,t4,S,tl,tr);
  if (z < S[5,3]) {and (tl > yl)} then yl:= tl;
  if (z < S[4,3]) {and (tr < yr)} then yr:= tr
end;
end;

procedure CylindricalShd( z
var yl,yr
var a,b,c,p,wp,w,theta,t1,t2,t3,t4
var tl,tr,h,k
begin
  wp:= pi/2.0;
  w:= AngLin(S);
  theta:= Atan2((S[4,3]-S[5,3]),(S[4,2]-S[5,2]));
  t1:= sin(w+theta)/sin(w);
  t2:= sin(w-wp+theta)/sin(w);
  t3:= sin(-theta)/sin(w);
  t4:= sin(wp-theta)/sin(w);
  h:= S[2,2]; k:= z-S[2,3];
  if (z <= S[1,3]) and (z >= S[3,3]) then
  begin
    yl:= (z*(t2-t4)-h*(t3-tl))/(tl-t3);
    yr:= (z*(t2-t4)-h*(tl-t3))/(tl-t3);
  end
  else begin
    yl:= -1.0;
    yr:= -1.0
  end;
  if (S[2,3]-S[3,3] > 0.0) and
    ((z < S[4,3]) or (z < S[5,3])) then
  begin
    BottornShd(z,tl,t2,t3,t4,S,tl,tr);
    if (z < S[5,3]) {and (tl > yl)} then yl:= tl;
    if (z < S[4,3]) {and (tr < yr)} then yr:= tr
  end;
end;

procedure SetMat(sec : Char);
var
  ty1,ty2,sech,z : real;
  i,j : integer;
begin
  FillChar(GridGlb,sizeof(GrdGlb),100);
  if sec in ['B','D'] then with BldGlb do
  begin
    sech:= wdth/2.0*tan(RoofAng*rad)+hgt;
    for i:= FindYPoint(l,sech,O.O,sech) to
     FindYPoint(l,sech,O.O,hgt) do
    begin
      z:= FindYpPoint(l,sech,O.O,i);
      ty1:= wdth/2.0-(sech-z)/tan(RoofAng*rad);
      for j:= 1 to FindXPoint(l,wdth,O.O,ty1)-1 do
       GridGlb[i,j]: = 255;
      ty2:= wdth/2.0+(sech-z)/tan(RoofAng*rad);
      for j:= FindXPoint(l,wdth,O.O,ty2)+1 to Ncol do
       GridGlb[i,j]: = 255;
    end
  end
end;

procedure FillMat( sec
  var
    pct
    xyz
    shape
    var GridGlb
begin
  case sec of
    'A'..'C': begin
      sech:= BldGlb.hgt;
      secw:= BldGlb.lngth
      end;
    'B'..'D': begin
      sech:= BldGlb.wdth / 2.0*tan(BldGlb.RoofAng*rad) + BldGlb.hgt;
      secw:= BldGlb.wdth;
      end;
    '0'..'3': begin
      sech := BldGlb.wdth/(2.0*cos(BldGlb.RoofAng*rad));
      secw:= BldGlb.lngth
      end;
  end;
  StartRow:= FindYPoint(1,sech,0.0,xyz[1,3]*1.5);
  LastRow:= FindYPoint(1,sech,0.0,xyz[3,3]);
for i:= startrow to lastrow do
begin
    z:= FindYpPoint(1,sech,0.0,i);
    case shape of
        'P': ParabolocShd(z,xyz,yl,yr);
        'E': EllipticShd(z,xyz,yl,yr);
        'C': CylindricalShd(z,xyz,yl,yr);
    end;
    if ((yl >= 0.0) and (yr >= 0.0)) and
        (sec in ['B','D']) and
        (z > BldGlb.hgt) then
begin
    tyl:= secw/2.0-(sech-z)/tan(BldGlb.RoofAng*rad);
    ty2:= secw/2.0+(sech-z)/tan(BldGlb.RoofAng*rad);
    if (yl > ty2) or (yr < tyl) then
begin
        yl:= -1.0;
        yr:= -1.0;
    end
    else begin
        if yl < tyl then yl:= tyl;
        if yr > ty2 then yr:= ty2;
    end
end;
end;
if (yl >= 0.0) and (yr >= 0.0) then
begin
    startcol:= FindXPoint(1,secw,0.0,yl);
    Lastcol:= FindXPoint(1,secw,0.0,yr);
    for j:= startcol to lastcol do
begin
    case sec of
        'A','C',
        'B','D',
        '0' : begin
            row:= i;
            col:= j
        end;
        '1' : begin
            row:= 1-(j-Ncol);
            col:= i
        end;
        '2' : begin
            row:= abs(i-Nrow-1);
            col:= abs(j-Ncol-1)
        end;
        '3' : begin
            row:= j;
            col:= 1-(i-Nrow)
        end;
end;
GridGlb[row,col]:= 100
round(GridGlb[row,col]*0.01*pct*100.0)

(*
    Calculate hourly average
*)

function GetAve(   sec : char;
    GridGlb : surface;
    var prcnt_shaded : real): real;
var
    shaded_cells,cell_used,sum,temp,z : real;
    i,j,y : integer;
begin
    shaded_cells:= 0.0;
    cell_used:= 0.0;
    sum:= 0.0;
    for i:= 1 to Nrow do
        for j:= 1 to Ncol do
            if GridGlb[i,j] < 255 then
                begin
                    cell_used:= cell_used+1.0;
                    sum:= sum+GridGlb[i,j]*0.01;
                    if GridGlb[i,j] < 100 then
                        shaded_cells:= shaded_cells+1.0
                    end;
                prcnt_shaded:= shaded_cells*100.0/cell_used;
                GetAve:= sum/cell_used
            end;

(*
    Draw surface border
*)

procedure PutBorder(Sec : char);
var
    i,x,j,t1,t2 : integer;
    temp,Yindex,Xindex,Xincr : real;
begin
    draw(142,12,142,132,1);
    draw(142,132,622,132,1);
    if sec in ['B','D'] then with BldGlb do
        begin
            temp:= wdth/2.0*tan(RoofAng*rad)+hgt;
            draw(FindXPoint(142,wdth,0.0,0.0),
                FindYPoint(12,temp,0.0,hgt),
                FindXPoint(142,wdth,0.0,wdth/2.0),
                FindYPoint(12,temp,0.0,hgt),
            );
        end;
    end;
FindYPoint(12,temp,0.0,temp),l);
draw(FindXPoint(142,wdth,0.0,wdth),
    FindYPoint(12,temp,0.0,hgt),
    FindXPoint(142,wdth,0.0,wdth/2.0),
    FindYPoint(12,temp,0.0,temp),l);
draw(622,FindYPoint(12,temp,0.0,hgt),622,132,1);
end
else begin
    draw(622,12,622,132,1);
    draw(142,12,622,12,1);
end;
x:= originx;
for i:= 1 to 25 do
begin
    if ((i-l)mod 4) = 0 then draw(x,132,x,135,1);
x:= x+intervalx div 4
end;
t1:= OriginX-8; t2:= OriginY;
for i:= 1 to 2 do
begin
    t2:= t2+IntervalY;
    Draw(t1,t2,originx,t2,1)
end;
case sec of
    'A','C': begin
        Yindex:= BldGlb.hgt;
        Xincr:= BldGlb.lngth/24.0
    end;
    'B','D': begin
        Yindex:= BldGlb.hgt+
            BldGlb.wdth/2.0*tan(BldGlb.RoofAng*rad);
        Xincr:= BldGlb.wdth/24.0
    end;
    'E','F': begin
        Yindex:= BldGlb.wdth /
            (2.0*cos(BldGlb.RoofAng*rad));
        Xincr:= BldGlb.lngth/24.0
    end;
end;
temp:= Yindex/3.0;
GotoXY(1,2);
for i:= 1 to 4 do
begin
    Writeln(Yindex:16:2);
    if i <> 4 then for j:= 1 to 4 do Writeln;
    if i < 3 then
        yindex:= Yindex-temp
    else
        Yindex:= 0.0
end;
Xindex:= 0.0;
Write(Xindex:19:1);
for i:= 1 to 6 do
begin
  Xindex:= Xindex+Xincr*4.0;
  Write(Xindex:10:1)
end;
Writeln
end;

(*
  Clear display
*)

procedure cls(sec : char);
var
temp : real;
begin
  GraphWindow(143,13,621,131);
  ClearScreen;
  GraphWindow(0,0,639,199);
  if sec in ['B','D'] then with BldGlb do
  begin
    temp:= wdth/2.0*tan(RoofAng*rad)+hgt;
    draw(FindXPoint(142,wdth,0.0,0.0),
         FindYPoint(12,temp,0.0,hgt),
         FindXPoint(142,wdth,0.0,wdth/2.0),
         FindYPoint(12,temp,0.0,temp),1);
    draw(FindXPoint(142,wdth,0.0,wdth),
         FindYPoint(12,temp,0.0,hgt),
         FindXPoint(142,wdth,0.0,wdth/2.0),
         FindYPoint(12,temp,0.0,temp),1)
  end;
end;

(*
  Draw grid on screen
*)

procedure PutGrid;
const
  Ver : array[0..7] of byte =
    ($80,$80,$80,$80,$80,$80,$80,$80);
  Hor : array[0..7] of byte =
    ($00,$00,$00,$10,$00,$00,$00,$00);
begin
  Pattern(Hor);
  FillPattern(142,7,622,127,1);
  Pattern(Ver);
  FillPattern(142,12,621,132,1)
end;
procedure DrawShade(  
  sec, shape : char;
  Grd, init : Boolean;
  xyz : Ary5x3);

var
  i, j1, j2, row1, col1, row2, col2 : integer;
  z, sech, secw, yl, yr, tyl, ty2 : real;

begin
  if init then
    begin
      cls(sec);
      if Grd then PutGrid
    end;
  case sec of
    'A', 'C': begin
      sech := BldGlb.hgt;
      secw := BldGlb.lngth
    end;
    'B', 'D': begin
      sech := BldGlb.width / 2.0*tan(BldGlb.RoofAng*rad) + BldGlb.hgt;
      secw := BldGlb.width;
    end;
    '0'..'3': begin
      sech := BldGlb.width/(2.0*cos(BldGlb.RoofAng*rad));
      secw := BldGlb.lngth
    end;
  end;

  for i := FindYPoint(12, sech, 0.0, xyz[1,3]*1.5) to
       FindYPoint(12, sech, 0.0, xyz[3,3]) do
    begin
      z := FindYPoint(12, sech, 0.0, i);
      case shape of
        'P': ParabolocShd(z, xyz, yl, yr);
        'E': EllipticShd(z, xyz, yl, yr);
        'C': CylindricalShd(z, xyz, yl, yr);
      end;
      if ((yl >= 0.0) and (yr >= 0.0)) and
         (sec in ['B', 'D']) and (z > BldGlb.hgt) then
        begin
          tyl := secw/2.0-(sech-z)/tan(BldGlb.RoofAng*rad);
          ty2 := secw/2.0+(sech-z)/tan(BldGlb.RoofAng*rad);
          if (yl > ty2) or (yr < tyl) then
            begin
              yl := -1.0;
              yr := -1.0;
            end;
        end;
  end;
end
else begin
if yl < tyl then yl:= tyl;
if yr > ty2 then yr:= ty2;
end
end;
if (yl >= 0.0) and (yr >= 0.0) then
begin
jl:= FindXPoint(l42,secw,0.0,yl);
j2:= FindXPoint(l42,secw,0.0,yr);
case sec of
'A','C',
'B','D',
'0'
: begin
row1:= i;
coll:= jl;
row2:= i;
col2:= j2;
end;
'1'
: begin
row1:= 1-(jl-Ncol);
coll:= i;
row2:= 1-(j2-Ncol);
col2:= i;
end;
'2'
: begin
row1:= abs(i-Nrow-1);
coll:= abs(jl-Ncol-1);
row2:= abs(i-Nrow-1);
col2:= abs(j2-Ncol-1);
end;
'3'
: begin
row1:= j1;
coll:= i-(i-Nrow);
row2:= j2;
col2:= i-(i-Nrow);
end;
draw(coll,row1,col2,row2,l)
end
end
end;

(*
Modify azimuth angle to that it is between -90 and 90
degrees
*)

function TransformAng(sec : char; 
DispAng : real; 
azm : real): real;
var
Ang : real;
begin
  case sec of
    'A': Ang:= 90.0-azm-DispAng;
    'B': if (azm >= 0.0) and (azm < 180.0) then
        Ang:= -(azm+DispAng)
        else
        Ang:= 360.0-azm-DispAng;
    'C': Ang:= 270.0-azm-DispAng;
    'D': Ang:= 180.0-azm-DispAng;
  end;
  TransFormAng:= Ang
end;

procedure TransformCoord( sec : char;
Xo,Yo : real;
var x,y : real);
var
  h,k,r : real;
  alpha,beta : real;
  M,N : Arylx3;
  A : Ary3x3;
begin
  fillchar(A,SizeOf(A),0);
  with BldGlb do
    case sec of
      'A': begin
        alpha:= -DispAng*rad;
        h:= 0.0;
        k:= 0.0
      end;
      'B': begin
        alpha:= (270.0-DispAng)*rad;
        h:= lngth*cos((90.0+DispAng)*rad);
        k:= lngth*sin((90.0+DispAng)*rad)
      end;
      'C': begin
        alpha:= (180.0-DispAng)*rad;
        beta:= atan2(wdth,lngth);
        r:= sqrt(sqr(wdth)+sqr(lngth));
        h:= r*cos((90.0+DispAng)*rad+beta);
        k:= r*sin((90.0+DispAng)*rad+beta)
      end;
      'D': begin
        alpha:= (90.0-DispAng)*rad;
        h:= wdth*cos(pi+DispAng*rad);
        k:= wdth*sin(pi+DispAng*rad)
      end;
    end;
  A[1,1]:= cos(alpha);
\[ A[1,2] := \sin(\alpha); \]
\[ A[2,1] := -\sin(\alpha); \]
\[ A[2,2] := \cos(\alpha); \]
\[ A[3,3] := 1.0; \]
\[ M[1] := Xo-h; \]
\[ M[2] := Yo-k; \]
\[ M[3] := 0.0; \]
solve(A, M, N);
\[ x := N[1]; \]
\[ y := N[2] \]
end;

(*
Define imaginary 3-D planes where surface is contained *)

procedure DefPlane(sec : char;
var P : Ary3x3);
begin
  fillchar(P, SizeOf(P), 0);
  with BldGlb do
  case sec of
    'A', 'C', 'B', 'D': begin
      P[2,2] := lstch;
      P[3,2] := lstch;
      P[3,3] := hgt;
    end;
    '0': begin
      P[1,3] := hgt;
      P[2,2] := lstch;
      P[2,3] := hgt;
      P[3,1] := -wdth/2.0;
      P[3,2] := lstch;
      P[3,3] := wdth/2.0*tan(RoofAng*rad)+hgt;
    end;
    '1': begin
      P[1,3] := hgt;
      P[2,2] := wdth/2.0;
      P[2,3] := wdth/2.0*tan(RoofAng*rad)+hgt;
      P[3,1] := -lstch;
      P[3,2] := wdth/2.0;
      P[3,3] := wdth/2.0*tan(RoofAng*rad)+hgt;
    end;
    '2': begin
      P[1,1] := -wdth/2.0;
      P[1,3] := wdth/2.0*tan(RoofAng*rad)+hgt;
      P[2,1] := -wdth/2.0;
      P[2,2] := lstch;
      P[2,3] := wdth/2.0*tan(RoofAng*rad)+hgt;
    end;
  end;
P[3,1] := -wdth;
P[3,2] := lngth;
P[3,3] := hgt;
end;

'3': begin
    P[1,2] := wdth/2.0;
P[1,3] := wdth/2.0*tan(RoofAng*rad)+hgt;
P[2,2] := wdth;
P[2,3] := hgt;
P[3,1] := -lngth;
P[3,2] := wdth;
P[3,3] := hgt;
end;

end
end;

(* Check if shadow exists *)

function ShdCheck( sec var ShadeExists temp,Thgt i,j begin

    P : Ary3x3;
    azm,elv : real;
    var xyz : Ary5x3 ): boolean;

    ShadeExists := check(P,xyz[1,1],xyz[1,2],xyz[1,3]);
    if ShadeExists then
      begin
        for i:= 1 to 5 do
          if i <> 2 then
            begin
              project(P,azm,90.0-elv,xyz[i,1],xyz[i,2],xyz[i,3]);
              for j:= 1 to 3 do
                RoundUp(xyz[i,j],4)
            end;
        case sec of
          'A','C': if (xyz[1,3] > 0.0) and
                        (xyz[3,3] < BldGlb.hgt) and
                        (xyz[4,2] > 0.0) and
                        (xyz[5,2] < BldGlb.lngth) then
                        ShadeExists := true;
          'B','D': begin
            Thgt := BldGlb.wdth/
            2.0*tan(BldGlb.RoofAng*rad)+
            BldGlb.hgt;
            if (xyz[1,3] > 0.0) and (xyz[3,3] < Thgt)
            and (xyz[4,2] > 0.0)
and (xyz[5,2] < BldGlb.wdth) then
  ShadeExists:= true
end;
'0': begin
  Thgt:= BldGlb.wdth/
    2.0*tan(BldGlb.RoofAng*rad) +
    BldGlb.hgt;
  if (xyz[1,3] > BldGlb.hgt) and
      (xyz[3,3] < Thgt) and
      (xyz[4,2] > 0.0) and
      (xyz[5,2] < BldGlb.lngth) then
    ShadeExists:= true
  end;
'1','3': begin
  if (xyz[1,1] < 0.0) and
      (xyz[3,1] > -BldGlb.lngth) then
    ShadeExists:= true
  end;
'2': begin
  Thgt:= BldGlb.wdth /
    2.0*tan(BldGlb.RoofAng*rad) +
    BldGlb.hgt;
  if (xyz[1,3] < Thgt) and
      (xyz[3,3] > BldGlb.hgt) and
      (xyz[4,2] > 0.0) and
      (xyz[5,2] < BldGlb.lngth) then
    ShadeExists:= true
  end
end
if ShadeExists then
begin
 inter(xyz[1,1],xyz[1,2],xyz[1,3],
    xyz[3,1],xyz[3,2],xyz[3,3],
    xyz[5,1],xyz[5,2],xyz[5,3],
    xyz[4,1],xyz[4,2],xyz[4,3],
    xyz[2,1],xyz[2,2],xyz[2,3]);
 for j:= 1 to 3 do
   RoundUp(xyz[2,j],4)
end;
ShdCheck:= ShadeExists;
end;

(*
  Rotate/Translate coordinates
  *)

procedure NormalizeCoord(  sec : char;
                          var x  : Ary5x3);
var
  xcos,xsin,Thgt : real;
begin
  with BldGlb do
  begin
    case sec of
    '0' : begin
      xcos := cos((90.0-RoofAng)*rad);
      xsin := sin((90.0-RoofAng)*rad);
      x[1,3] := (x[1,3] - hgt)*xcos - x[1,1]*xsin;
      x[2,3] := (x[2,3] - hgt)*xcos - x[2,1]*xsin;
      x[3,3] := (x[3,3] - hgt)*xcos - x[3,1]*xsin;
      x[4,3] := (x[4,3] - hgt)*xcos - x[4,1]*xsin;
      x[5,3] := (x[5,3] - hgt)*xcos - x[5,1]*xsin;
    end;
    '1' : begin
      xcos := cos(RoofAng*rad);
      xsin := sin(RoofAng*rad);
      x[1,2] := x[1,2]*xcos + (x[1,3] - hgt)*xsin;
      x[2,2] := x[2,2]*xcos + (x[2,3] - hgt)*xsin;
      x[3,2] := x[3,2]*xcos + (x[3,3] - hgt)*xsin;
      x[4,2] := x[4,2]*xcos + (x[4,3] - hgt)*xsin;
      x[5,2] := x[5,2]*xcos + (x[5,3] - hgt)*xsin;
      x[1,3] := -x[1,1];
      x[2,3] := -x[2,1];
      x[3,3] := -x[3,1];
      x[4,3] := -x[4,1];
      x[5,3] := -x[5,1];
    end;
    '2' : begin
      xcos := cos((90.0+RoofAng)*rad);
      xsin := sin((90.0+RoofAng)*rad);
      Thgt := width/2.0*tan(RoofAng*rad) + hgt;
      x[1,3] := (x[1,3] - Thgt)*xcos - (x[1,1] - width/2.0)*xsin;
      x[2,3] := (x[2,3] - Thgt)*xcos - (x[2,1] - width/2.0)*xsin;
      x[3,3] := (x[3,3] - Thgt)*xcos - (x[3,1] - width/2.0)*xsin;
      x[4,3] := (x[4,3] - Thgt)*xcos - (x[4,1] - width/2.0)*xsin;
      x[5,3] := (x[5,3] - Thgt)*xcos - (x[5,1] - width/2.0)*xsin;
    end;
    '3' : begin
      xcos := cos(-RoofAng*rad);
      xsin := sin(-RoofAng*rad);
      Thgt := width/2.0*tan(RoofAng*rad) + hgt;
      x[1,2] := (x[1,2] - width/2.0)*xcos + (x[1,3] - Thgt)*xsin;
      x[2,2] := (x[2,2] - width/2.0)*xcos + (x[2,3] - Thgt)*xsin;
      x[3,2] := (x[3,2] - width/2.0)*xcos + (x[3,3] - Thgt)*xsin;
    end;
  end;
end;
end;
end;
end;

(* Obtain season where data should be saved *)

procedure GetSeason;
const
  accept : CharAry = ('W','P','S','U','F','A','H','C','Q');
begin
  ClrScr;
  DrawBox(31,5,49,17,'SEASON MENU');
  GotoXY(33,7); Write('Winter');
  GotoXY(33,8); Write('winter/sPring');
  GotoXY(33,9); Write('Spring/summer');
  GotoXY(33,10);Write('sUmmer');
  GotoXY(33,11);Write('summer/Fall');
  GotoXY(33,12);Write('fAll/winter');
  GotoXY(33,13);Write('peak Heating');
  GotoXY(33,14);Write('peak Cooleing');
  GotoXY(33,15);Write('Quit');
  case Query(9,7,35,1,accept) of
    1: SeasonCodeGlb:= 0;
    2: SeasonCodeGlb:= 24;
    3: SeasonCodeGlb:= 48;
    4: SeasonCodeGlb:= 72;
    5: SeasonCodeGlb:= 96;
    6: SeasonCodeGlb:= 120;
    7: SeasonCodeGlb:= 144;
    8: SeasonCodeGlb:= 168;
    9: SeasonCodeGlb:= -1;
  end
end;

(* Create/Open a file to save percent reduction *)
procedure InitShadingFile;
var
  rec     : ShadingFactors;
  Hr, season  : integer;
begin
  if not exist(FileNameGlb+'.Tmp') and (DskSpc('*') >= 195) then
  begin
    @$-
    assign(ShdF,FileNameGlb+'.Tmp');
    IOcheck;
    if IOstat = 0 then
    begin
      ReWrite(ShdF);
      IOcheck;
      if IOstat <> 0 then
      begin
        close(ShdF)
      end
      else begin
        fillchar(rec,sizeof(rec),0);
        for Season:= 1 to lastSeason do
          for Hr:= 1 to EndOfDay do
            Write(ShdF,Rec);
        end
      end
    end
    else if exist(FileNameGlb+'.Tmp') then
    begin
      assign(ShdF,FileNameGlb+'.Tmp');
      IOcheck;
      if IOstat = 0 then
      begin
        Reset(ShdF);
        IOcheck;
        if IOstat <> 0 then close(ShdF)
      end
      else begin
        error('Disk is full')
      end
    end
  end
end;

(*
 Save on file percent reductions by seasons
 *)

procedure UpdateShadingFile(sec : Char);
var
  hr, i, SurfaceOrientation : integer;
  ave                     : real;
  ShadeFactor             : ShadingFactors;
begin
with BldGlb do
begin
  if (DispAng >= 315.0) and
      (DispAng <= 360.0) then SurfaceOrientation:= 1;
  if (DispAng >= 0.0 ) and
      (DispAng <= 45.0) then SurfaceOrientation:= 1;
  if (DispAng >= 46.0 ) and
      (DispAng <= 134.0) then Surfaceorientation:= 2;
  if (DispAng >= 135.0) and
      (DispAng <= 224.0) then Surfaceorientation:= 3;
  if (DispAng >= 225.0) and
      (DispAng <= 314.0) then Surfaceorientation:= 1;
end;
hr:= SeasonCodeGlb;
i:= 1;
while i <= 47 do
begin
  if i = 1 then
    ave:= (AveGlb[i]+AveGlb[i+1])/2.0
  else
    ave:= (AveGlb[i-1]+AveGlb[i]+AveGlb[i+1])/3.0;
  seek(ShdF,hr);
  Read(ShdF,ShadeFactor);
  with ShadeFactor do
    case SurfaceOrientation of
    1: case sec of
        'A': Ev:= ave;
        'B': Sv:= ave;
        'C': Wv:= ave;
        'D': Nv:= ave;
        'E': Et:= ave;
        'F': Wt:= ave
    end;
    2: case sec of
        'A': Nv:= ave;
        'B': Ev:= ave;
        'C': Sv:= ave;
        'D': Wv:= ave;
        'E': Nt:= ave;
        'F': St:= ave
    end;
    3: case sec of
        'A': Wv:= ave;
        'B': Nv:= ave;
        'C': Ev:= ave;
        'D': Sv:= ave;
        'E': Wt:= ave;
        'F': Et:= ave
    end;
    4: case sec of
        'A': Sv:= ave;
'B': Wv := ave;
'C': Nv := ave;
'D': Ev := ave;
'E': St := ave;
'F': Nt := ave
end;

seek(ShdF, hr); Write(ShdF, ShadeFactor);
hr := hr + 1;
i := i + 2
end
end;

(*
   Cast shadows on vertical surfaces
*)

procedure WallABCD(sec : char;
   Start : ObjPointer);

var
   Plane : Ary3x3;
   xyz : Ary5x3;
   X, Y, az : real;
   prcnt_shaded : real;
   i, time : integer;
   ok, Clean : boolean;
   info : ObjPointer;
   ch : char;
begin
   GotoXY(43,1); Write('WALL ', sec);
   DefPlane(sec, Plane);
   QuitGlb := false;
   for time := 0 to 47 do with sunGlb[time] do
      begin
         if keypressed then
            begin
               Read(kbd, ch);
               if ch = #27 then
                  begin
                     QuitGlb := true;
                     exit
                  end;
            end;
         SetMat(sec);
         AveGlb[time] := 0.0;
         if DrawGlb then cls(sec);
         GotoXY(1, 23);
         Write('TIME: ', t: 5, ' AZM: ', azm: 6:2,
               ' ELV: ', elv: 6:2);
         Writeln(' PERCENT SHADEN: ');
         if (elv > 0.0) and (elv < 90.0) then
begin
AveGlb[time] := 1.0;
Clean := true;
fillChar(GridGlb, sizeof(GridGlb), 100);
az := TransformAng(Sec, BldGlb.DispAng, azm);
if (az < 90.0) and (az > -90.0) then
begin
  GotoXY(2, 25);
  Write('');
  info := Start;
  while info <> nil do with info do
  begin
    if chk = 127 then
    begin
      TransformCoord(Sec, Xo, Yo, X, Y);
      case shape of
        'P': ParabolicObj(az, elv, X, Y, Base, Top, Dia, xyz);
        'E': EllipticObj(az, elv, X, Y, Base, Top, Dia, xyz);
        'C': CylindricalObj(az, elv, X, Y, Base, Top, Dia, xyz);
      end;
    end;
    if X < 0.0 then
      if Y < 0.0 then
        ok := check(Plane, xyz[4, 1],
                     xyz[4, 2], xyz[4, 3])
      else
        ok := check(Plane, xyz[5, 1],
                     xyz[5, 2], xyz[5, 3])
    else
      ok := true;
    if ok and ShdCheck(sec, Plane, az, elv, xyz) then
    begin
      AveGlb[time] := 0.0;
      FillMat(sec, pct, xyz, shape, GridGlb);
      GotoXY(2, 25);
      Write('SHADE OF TREE # ', num: 4,
            '); if DrawGlb and
      (time >= Timel) and
      (time <= Time2) then
        DrawShade(sec, shape, GrdGlb, Clean, xyz);
      Clean := false;
    end
  end;
  info := info^.next
end;
if AveGlb[time] = 0.0 then
  AveGlb[time] := GetAve(sec, GridGlb, prcnt_shaded)
else begin
  prcnt_shaded := 0.0;
if DrawGlb then cls(sec)
end;
GotoXY(1,23);
Write('TIME: ',t:5,'
  AZM: ',
  azm:6:2,'
  ELV: ',elv:6:2);
Write(' PERCENT SHADED:',prcnt_shaded:6:2);
GotoXY(2,25);
Write('');
if (DrawGlb and (time >= Timel) and
  (time <= Time2)) and PrintGlb then
  HardCopy
else
  wait(DelayGlb);
end else begin
  GotoXY(2,25);
  Write('SECTION HAS NO DIRECT SUN LIGHT');
end
end
end
end;

(*
Cast shadows on the roof of the building
*)

procedure RoofEF( sec : char;
                Start : ObjPointer);

var
  Plane : Ary3x3;
  xyz : Ary5x3;
  X,Y,az : real;
  prcnt_shaded : integer;
  i,time : real;
  ok,Clean : boolean;
  LocSec,TS : char;
  info : ObjPointer;
  ch : char;
begin
  GotoXY(43,1); Write('ROOF ',sec);
  QuitGlb:= false;
  for time:= 0 to 47 do with sunGlb[time] do
    begin
      if keypressed then
        begin
          Read(kbd,ch);
          if ch = #27 then
            begin
QuitGlb := true;
exit
end;
end;
AveGlb[time] := 0.0;
if DrawGlb then cls(sec);
GotoXY(1,23);
Write('TIME:', t:5,' ','AZM:', azm:6:2,' ','ELV:', elv:6:2);
WriteLn('PERCENT SHADED:');
if (elv > 0.0) and (elv < 90.0) then
begin
AveGlb[time] := 1.0;
Clean := true;
fillChar(GridGlb,sizeof(GridGlb),100);
for Locsec:= '0' to '3' do
begin
 case LocSec of
 '0': if sec = 'E' then TS := 'A' else TS := 'C';
 '1': if sec = 'E' then TS := 'B' else TS := 'D';
 '2': if sec = 'E' then TS := 'C' else TS := 'A';
 '3': if sec = 'E' then TS := 'D' else TS := 'B';
end;
DefPlane(LocSec,Plane);
az := TransformAng(TS,BldGlb.DispAng,azm);
if not((az < 90.0) and (az > -90.0)) then
begin
GotoXY(2,25);
Write(' ');  
end;
if (az < 90.0) and (az > -90.0) and
( AngPlaneLine(Plane,azm,elv) > 0.0 ) then
begin
info := Start;
while info <> nil do with info^ do
begin
 if chk = 127 then
 begin
 TransformCoord(TS,Xo,Yo,X,Y);
 ok := (X >= 0.0) and
 ((LocSec in ['0','2']) and
  (Y >= 0.0) and (Y <= BldGlb.lngth))
or
 ((LocSec in ['1','3']) and (X > 0.0));
 if ok then
 begin
 case shape of
 'P': ParabolicObj(az,elv,X,Y,Base,
 Top,Dia,xyz);
 'E': EllipticObj(az,elv,X,Y,Base,
Top, Dia, xyz);

'C': CylindricalObj(az, elv, X, Y, Base, Top, Dia, xyz);

end;
if ShdCheck(LocSec, Plane, az, elv, xyz) then
begin
  AveGlb[time] := 0.0;
  NormalizeCoord(LocSec, xyz);
  FillMat(LocSec, pct, xyz, shape, GridGlb);
  GotoXY(2, 25);
  Write('SHADE OF TREE # ', num: 4, ', ');
  if DrawGlb and (time >= Timel) and (time <= Time2) then
     DrawShade(LocSec, shape, GrdGlb, Clean, xyz);

  Clean := false;
end

info := info^.next

end
end end

if AveGlb[time] = 0.0 then
  AveGlb[time] := GetAve(sec, GridGlb, prcnt_shaded)
else
  prcnt_shaded := 0.0;
  GotoXY(l, 23);
  Writeln('PERCENT SHADED: ', prcnt_shaded: 6: 2);
  GotoXY(2, 25);
  Write('');
  if (DrawGlb and (time >= Timel) and (time <= Time2)) and PrintGlb then
    HardCopy
  else
    wait(DelayGlb);
end
else begin
  GotoXY(2, 25);
  Write('SECTION HAS NO DIRECT SUN LIGHT');
end
end

(*
   Display options and obtain choice
*)

function GetOptions: boolean;
const
accept : CharSet = ['A'..'F','*'];
term : CharSet = ['^M','^[]';

var
TC : char;

begin
ClrScr;
DrawBox(30,5,50,15,'SECTION MENU');
DrawBox(30,18,50,20,'SECTIONS SELECTED');
DrawBox(1,22,79,24,'');
GotoXY(33,7); Write('Section Name');
GotoXY(33,8); Write(' Wall A');
GotoXY(33,9); Write(' Wall B');
GotoXY(33,10); Write(' Wall C');
GotoXY(33,11); Write(' Wall D');
GotoXY(33,12); Write(' Roof E');
GotoXY(33,13); Write(' Roof F');
GotoXY(33,19);
Write('Sections:');
GotoXY(19,23);
Write('To select ALL enter "*", <Esc> to Quit.');
InputStr(CommandLineGlb,6,41,18,term,accept,TC);
if pos('*',CommandLineGlb) > 0 then
CommandLineGlb:= 'ABCDEF';
GetOptions:= (TC <> '[]') and (CommandLineGlb <> '');
end;

(*
  Main Simulator procedure
 *)

begin {Simulator}
if PrintGlb then
begin
  GraphCount:= 0;
end;
QuitGlb:= false;
if (DateGlb.chk <> 7) or (LocGlb.chk <> 15)
  or (BldGlb.chk <> 31)
  or (StartGlb = nil) then
  error('Can’’t proceed with current information.' )
else begin
  if OutHrAve then
  begin
    GetSeason;
    if SeasonCodeGlb >= 0 then
    begin
      InitShadingFile;
      if IOstat <> 0 then exit
    end
    else exit
  end
end;
if not GetOptions then exit;
if DrawGlb then
  HiRes
else
  clrscr;
CommandLineLoc:= CommandLineGlb;
while not(CommandLineLoc = '') do
begin
  if DrawGlb then
    PutBorder(CommandLineLoc[1]);
  case CommandLineLoc[1] of
    'A'..'D': WallABCD(CommandLineLoc[1],StartGlb);
    'E', 'F': RoofEF(CommandLineLoc[1],StartGlb);
  end;
  if not QuitGlb and OutHrAve then
    UpdateShadingFile(CommandLineLoc[1]);
  delete(CommandLineLoc,1,1);
  if QuitGlb then CommandLineLoc:= ''
end
TextMode;
if OutHrAve then Close(ShdF)
end {Simulator}: