Chapter 3 Technical Notes

Chapter Overview

File Formats

- HDF Scientific Data Format
- HDF 24-bit Raster Image Format
- HDF VSet Format
- Object Format

Lighting Model Equations

- Ambient Light
- Diffuse Light
- Specular Light
- Specular Exponent

Perspective Transformation Information

Non-Silicon Graphics Use

Future Enhancements

Chapter Overview

This chapter provides detailed technical insight into the inner workings of ISOVIS. It will help you to understand exactly how your data is being represented and what modifications (if any) are made to the data as it is displayed. Definitions of terms as well as discussions of current technical limitations of ISOVIS are included.

File Formats

The only input file format supported by ISOVIS is HDF. HDF files are data files which you create using NCSA's Hierarchical Data Format for file storage. *HDF* is a multi-object file format for the transfer of graphical and floating-point data between machines. Refer to Appendix A, "Obtaing NCSA HDF and Other Software," for information on obtaining NCSA HDF.

HDF Scientific Data Format

Currently, ISOVIS supports the SDS format in HDF for the input data. Due to the nature of the isosurface, the SDS must be three-dimensional. The range of valid floating-point values in a SDS file is unlimited.

A current limitation of ISOVIS is that it can only handle one dataset per HDF file. In other words, while it is possible under HDF to store multiple datasets in the same physical HDF file on disk, it is not currently possible to fully utilize these files with ISOVIS. When an HDF file containing more than one dataset is specified, ISOVIS extracts the first dataset it encounters within that file and ignores all others.

HDF 24-Bit Raster Image Format

Images rendered by ISOVIS can be saved as raster images for future viewing or for recording into some other media. ISOVIS saves these raster images in HDF files using the raster 24-bit format.

If 8-bit raster images are desired, then you can easily convert the 24-bit images to 8-bit images with hdf24hdf8, a utility provided with the general HDF software.

HDF Vset Format

By using vset format (NCSA Vset) with ISOVIS, you can store the polygons that represent the isosurface in an HDF file. In particular, the Vset will contain a root node named '/' with four children named 'px', 'py', 'pz', and 'plist3'. Px, py and pz contain the x, y, z vertices and plist3 contains the connectivity. (Refer to Appendix A, "Obtaing HDF," for information on obtaining NCSA HDF and other NCSA products like NCSA HDF Vset.)

NOTE: ISOVIS v.1.0 may create duplicate vertices since the current algorithm does not remember vertices during its search for an isosurface. A future version will probably do away with the redundant vertices.

Object Format

For portability, ISOVIS can store the polygons representing the isosurface in plain ASCII format. This format is also readable by programs such as the Personal Visualizer from Wavefront Technologies, Inc. You would import this type of file into Personal Visualizer as a Wavef*ront* file.

The actual format is simply a list of vertices followed by a list of connectivity. The following example is a short listing of a simple object file:

```
# This line is a comment line
# This file is an object file containing three vertices
# and one polygon. All are contained in the default group.
#
g d
v -10.0 5.0 2.1
v 3.14 15.92 6.7
v 0.0 3.2 1.2
fo 2 1 3
```

The vertices are all implicitly numbered from one, and they all begin with the letter ‘v’ followed by the x, y, and z floating-point coordinates. The connectivity lines all begin with the letters ‘fo’ followed by a list of vertices that make up the polygon. In the above example, one polygon is formed by connecting vertices 2 to 1 to 3. You can safely ignore all other lines in the object file that do not begin with either a ‘v’ or a ‘fo’.

NOTE: As with the Vset files, the object files will also contain duplicate vertices. (See the previous NOTE.)

Lighting Model Equations

A lighting model can be useful in scientific visualization, but its intent is not to make a rendered object more realistic since many objects are in fact abstract representations. ISOVIS uses lighting to augment and expedite the visual interpretation of data. It does this by enhancing otherwise subtle hues of the object such as its three-dimensionality. Since a lighting model needs to adjust polygon color values to simulate the effects of light shining on it, a user should understand what modifications take place so as not to misinterpret a visual representation as the distortion of data.

Some general assumptions have been included in the lighting model ISOVIS currently uses. For example, the light source emits infinite parallel rays of light which do not attenuate in intensity over distance. The light rays are parallel in a single vector direction specified with the -ll parameter. Therefore, sidedness of polygons is observed, meaning that the only side of a polygon which is influenced by the light source is the side whose normal points towards the light source — the other side is given the color of the ambient light. Currently, multiple light sources are not allowed.

In rendering polygons, ISOVIS uses an incremental rendering technique which implies that any given polygon does not influence any other polygon (i.e., there are no shadows or reflections of one polygon off of another). With these constraints, the lighting model does not deviate from that which is described in many computer graphics texts and is usually implemented in hardware.

Ambient Light

You may think of ambient light as background light. It results from the scattering of light off all objects in the immediate vicinity of a scene. As such it can be modeled as a uniform intensity of light in all directions. When ambient light illuminates a surface alone, the surface is shaded with a uniform color independent of view or position.

Diffuse Light

Diffuse light is light that is scattered in all directions due to surface “roughness.” This effect is modeled using Lambert’s cosine law which adjusts surface color values proportional to the cosine of the angle between the surface normal and the light source vector. This causes the

polygon's surface to be the brightest if it faces the light directly. The diffuse contribution will go to zero as the surface is angled away from the light, thus a surface facing away from the light will only be illuminated by the ambient light.

Specular Light

Specular light can be described as the bright spot you see when illuminating a shiny blue ball with a white light (the blue ball reference is an arbitrary example). The white part of the blue ball appears because of the specularly reflected light from the light source which shines off the ball and to your eye. If you alter your view, you notice that the specular highlight moves, indicating that the specular highlight is the color of the light source, and it is dependent on your viewing angle as well. This effect is simulated by adjusting surface color values proportional to the cosine of the angle between the reflected light and view direction. The computation of the reflected light direction is based on the fact that the angle of incidence is equal to the angle of reflection for shiny surfaces.

Specular Exponent

The specular component of the light is raised to a power Ns, the specular exponent. In the real world, most objects are less than perfect reflectors producing a range of reflected light for a single incident light direction. The Ns exponent term simulates this effect. The larger Ns is, the smaller the specular highlight; conversely the smaller Ns is, the larger the specular highlight.

Perspective Transformation Information

Figure 3.1 uses pseudo code based on the Silicon Graphics Graphics Library (SGI GL) to derive the perspective transformation used by ISOVIS. Knowledge of the perspective transform used by ISOVIS is useful in order to use ISOVIS in advanced applications.

Figure 3.1 Perspective Transformation Code

```
/* Pseudo code for isovis perspective transformation. These transformations are applied on
/* geometry which has been previously geometrically centered in X and Y.

fov = 450;            /* field of view in tenths of degrees
aspect = 1.0;         /* window aspect ratio
near = 0.1;           /* location of near clipping plane
far = 500.0;          /* far clipping plane

/* The perspective (fov,aspect,near,far) subroutine defines a perspective projection
/* transformation. The 4x4 homogeneous matrix used is:
/*
/* | cot(fov/2)                                                                  |
/* | ------     0                     0                     0             |
/* | aspect                                                                      |
/* |                                                                             |
/* |                                                                             |
/* | 0                cot(fov/2)            0                     0              |
/* |                                                                             |
/* |                                                                             |
/* |                                          far+near                           |
```

```
/* | 0                                  - ----          -1            |
/* |                                      far-near                          |
/* |                                                                        |
/* |                                                          2*far*near    |
/* | 0              0                     0                 - -----  |
/* |                                                          far-near      |
/*
/* The lookat (vx,vy,vz,px,py,pz,twist) subroutine defines a transformation to
/* position the camera at (vx,vy,vz) and looks at (px,py,pz) with twist.
/*
/* lookat(vx,vy,vz,px,py,pz,twist) = trans(-vx,-vy,-vz) x roty(a) x rotx(b) x rotz(-twist)
/* 'trans', 'roty', 'rotx', and 'rotz' are standard 4 by 4 transformation matrices
/* where      sin(a) = (px-vx) / (sqrt((px-vx)^2 + (pz-vz)^2))
/*            cos(a) = (vz-pz) / (sqrt((px-vx)^2 + (pz-vz)^2))
/*            sin(b) = (vy-py) / (sqrt((px-vx)^2 + (py-vy)^2 + (pz-vz)^2))
/*            cos(b) = (sqrt((px-vx)^2 + (pz-vz)^2))
/*                          / (sqrt((px-vx)^2 + (py-vy)^2 + (pz-vz)^2))
/*
/* The above information is that which is documented in the Silicon Graphics GT GL Users
/* Guide V1.0, page C-3.

perspective(fov,aspect,near,far);
```

Figure 3.1 Perspective Transformation Code (Continued)

```
/* premultiply the current matrix (i.e., window matrix) by a lookat() to position
/* the camera
lookat((xdim/2),(ydim/2),(zdim+100),0,0,0,0);

/* premultiply current matrix by scale(x,y,z) matrix from -s command line parameter
/* arguments:
scale(sx,sy,sz);

/* premultiply current matrix by rotate(x,y,z) matrix from -r command line parameter
/* arguments using a Z->Y->X rotation priority:  (premultiplying order is opposite
/* effective rotation order)
rotate_x(rx);
rotate_y(ry);
rotate_z(rz);

/* premultiply current matrix by translate(tx,ty,tz) matrix from -t command line parameter
/* arguments:
translate(tx,ty,tz);

/* current matrix at this point holds the perspective transformation matrix used by isovis
```

Non-Silicon Graphics Use

NCSA Isosurface Visualizer can be used on computers other than Silicon Graphics Irises, although with reduced functionality. On non-SGI systems, you only can create HDF Vsets and/or object files with ISOVIS. Display of the isosurface and capture of the resulting raster image are not available.

To build a version of ISOVIS for some other computer, just compile the source code without defining the C preprocessor macro name `SGI`. The makefile accompanying ISOVIS includes two sets of lines, one for SGI systems and one for other systems. See the makefile for further information.

If you wish to modify ISOVIS to support graphic output on computers other than Silicon Graphics Irises, you will need to modify the source code in the `graphics.c` file, but be forewarned that that file makes extensive use of a three-dimensional graphics library (currently GL). Also, you should look at the rest of the files and possibly modify the lines where the `SGI` preprocessor macro name is used.

Future Enhancements

Many enhancements could be developed to improve the functionality and usability of ISOVIS. For various reasons and design philosophies, ISOVIS 1.0 was released in its current form without these enhancements. However ISOVIS, 2.0 may contain many of the following capabilities in addition to the current capabilities in ISOVIS 1.0:

- **Elimination of Redundant Vertices**

Currently ISOVIS 1.0 will include redundant vertices into the Vset files and/or Object files due to algorithm simplicity and speed. A future version will probably do away with these extra vertices.

- **Multiple Surfaces**

ISOVIS 1.0 only allows you to display one isosurface at a time. A future version may allow multiple surfaces from a single dataset to be display at one time.

- **Interactive Setup and Manipulation**

Currently ISOVIS 1.0 is only command-line driven. ISOVIS 2.0 will add interactive menu's to allow many of the command line parameters to be set up interactively. All of the current batch-mode capabilities will remain in ISOVIS 2.0.