Chapter 3 Technical Notes

Chapter Overview

File Formats

- HDF 8-bit Raster Image Format
- HDF 24-bit Raster Image Format
- HDF Scientific Data Format

Geometric Representations used by Height-Color Visualizer

- Polygonal
- Relief
- Wireframe

8-bit vs. 24-bit Color Interpolation

Lighting Model Equations

- Ambient Light
- Diffuse Light
- Specular Light

Illumination Models

- Illumination Model 0
- Illumination Model 1

Perspective Transformation Information

Future Enhancements

Chapter Overview

This chapter provides detailed technical insight into the inner workings of HCVIS. 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 on current technical

limitations of HCVIS are included.

File Formats

The only file format supported by HCVIS 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. Currently, HCVIS supports the 8-bit and 24-bit raster image formats under HDF as well as the HDF 2D scientific data format. You may use any combination of these files for the height and color arguments as long as they have the same 2D XY dimensions (sometimes referred to as resolution or rank=2).

A current limitation of HCVIS is that it can only handle one data set per HDF file. In other words, while it is possible under HDF to store multiple data sets in the same physical HDF file on disk, it is not currently possible to fully utilize these files with HCVIS. When an HDF file containing more than one data set is specified as the <height_file> or <color_file> argument, HCVIS extracts the first data set it encounters within that file and ignores all others. Therefore, you must explicitly specify separate height and color file names on the command line. The height and color file name may be the same, in which case, height and color redundantly display the same information in the rendered image.

HDF 8-bit Raster Image Format

When you specify an HDF 8-bit raster image file for height or color, HCVIS tries to extract a color palette from one of these files. If both the height file and the color file have palettes, HCVIS uses the palette in the color file. The -c parameter overrides any color palettes it encounters within the height or color files. If no palettes are found in any of the files and you have not specified any on the command line, the internal palette used by HCVIS defaults to a linear ramp of gray (i.e. a linear ramp of total black to total white).

HDF 24-bit Raster Image Format

When you specify an HDF 24-bit raster image file for height, HCVIS internally converts the image to its equivalent intensity representation from which the program derives the height geometry. The formula used is:

```
scalar intensity = (0.30 * Red) + (0.59 * Green) +
(0.11 * Blue)
```

This computation is performed at every height node.

HDF Scientific Data Format

You should be aware of some important facts when using HCVIS with the HDF 2D scientific data format (hereafter referred to as DFSD). The range of values in a DFSD file with floating-point values can be any valid floating-point value. Internally HCVIS normalizes this data into integers in the range [0...255] using a straight forward linear transformation.

Geometric Representations used by Height-Color Visualizer

HCVIS uses three main geometric representations. They include the following:

Polygonal

HCVIS uses the polygonal representation of the data as its default. Internally, HCVIS reads in the data as a 2D array. The indices of this array form the XY coordinates of the geometry which is used to represent the data. The Z coordinate is derived from the scalar variable at every node of the height file. The geometry is centered about the origin in X and Y and the base of the geometry (where array values equal zero) lies in the Z=0 plane. Since the input data is essentially a regular 2D grid with a variable at every node, the polygonal representation is nothing more than a triangulated mesh of that grid with the height being derived from the variable at every node. The polygons are colored using a bi-linear color interpolation scheme between polygon vertex colors. This process is more commonly known as “Gouraud Shading.” For a definition of Gouraud Shading, see the section in this chapter entitled, “8-bit vs. 24-bit Color Interpolation.”

Relief

The polygonal and internal relief representation of the data are the same. Since the relief representation requires shading with a light source, shading takes place on geometry that is exactly the same as the polygonal geometry would be with the same parameters. However, when the relief polygons are drawn to the screen, the Z-coordinate of all polygon vertices sets to zero whereas in the polygonal representation, the drawn Z-coordinate is unaltered.

Wireframe

The wireframe representation of the data differs from the polygonal and relief representations in that it is not shaded with a light source. The wireframe representation is not a triangulated mesh but rather a rectangulated mesh connecting all adjacent nodes. Color intensities are linearly interpolated across a given line if the particular hardware supports it.

8-bit vs. 24-bit Color Interpolation

Both the polygon and relief representations of the data use the polygon primitive to draw the object to the screen. Specifically, all polygons in these representations are triangles. There are several ways to fill a given triangle based on the colors associated with each of the triangle’s three vertices. Obviously, if all the color intensities are the same at every vertex, the triangle would have uniform intensity in its interior. However, this usually is not the case with HCVIS. Therefore a method has been derived in computer graphics to interpolate color intensities across polygon vertices. This method is known as “Gouraud Shading.” Note that

"shading" in this instance refers to the coloring of a polygon regardless of any lighting calculations. Basically, Gouraud shading linearly interpolates color intensities along each edge. The edge values are then linearly interpolated along each horizontal scan line.

Colors can be assigned to triangle vertices in one of two methods. One method assigns a red, green, and blue value to each of the three triangle vertices. Each of these components is independently bi-linearly interpolated within the triangle boundaries to give the resultant shading. The other method assigns only a color index to each triangle vertex (this occurs with the -O8 and -i 0 parameter settings). These single scalar color indices are bi-linearly interpolated within the triangle boundaries using color indices to fill the triangle as opposed to using actual color intensities. The two methods can be visually different.

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. HCVIS 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 and its data gradations. 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 HCVIS 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.

To create polygons, HCVIS 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.

The terms used in the following equations are defined as follows:

```
C           unmodified original color from color field file
K           coefficients from -k parameter
L           light source (Lr,Lg,Lb are obtained from the -lc
parameter)
I           illumination
M           material
Ns          specular exponent from the -ns parameter
V           viewing vector direction
R           reflected light vector direction
a           ambient
```

```
d           diffuse
s           specular
r           red
g           green
b           blue
^           exponentiation
•           dot product
MAX(a,b)    returns the maximum of (a,b)

examples:
Mga  ambient green component of material
Ird  diffuse red component of illumination
```

Ambient light

Consider 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. The ambient reflection for a given material color is computed as follows:

```
Mra = Ka * Cr
Mga = Ka * Cg
Mba = Ka * Cb
```

The ambient contribution of the light source is:

```
Lra = Ka * Lr
Lga = Ka * Lg
Lba = Ka * Lb
```

The resulting equation for ambient reflectance is:

```
Ira = Mra * Lra
Iga = Mga * Lga
Iba = Mba * Lba
```

As these last three equations demonstrate, when ambient light illuminates a surface alone, the surface is shaded with a uniform illumination 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 (the cosine of this angle is just (N•L)). The diffuse reflection for a given material is computed as follows:

```
Mrd = Kd * Cr
Mgd = Kd * Cg
Mbd = Kd * Cb
```

The resulting equation for ambient and diffuse reflectance is:

```
Ird = Ira + Lr * Mrd * MAX((N•L),0)
Igd = Iga + Lg * Mgd * MAX((N•L),0)
```

```
Ibd = Iba + Lb * Mbd * MAX((N•L),0)
```

Notice that the MAX((N•L),0) term returns a zero when (N•L) is less than zero thus zeroing out the last three multiplied terms and illuminating the polygon with only ambient light. This illumination occurs when the polygon faces away from the light source. The surface appears brightest when a given polygon is perpendicular to the light direction, but becomes less bright when the surface lies at an angle to the light direction.

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 high-light moves, indicating that the specular high-light 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 cosine of this angle is just (R•V)). 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. The specular reflection for a given material color is computed as follows:

```
Mrs = Ks * Lr
Mgs = Ks * Lg
Mbs = Ks * Lb
```

The resulting equation for ambient, diffuse, and specular reflectance is:

```
Irs = Ird + Lr * Mrs * MAX((R•V),0)^Ns
Igs = Igd + Lg * Mgs * MAX((R•v),0)^Ns
Ibs = Ibd + Lb * Mbs * MAX((R•V),0)^Ns
```

Notice that the (R•V) term is raised to a power Ns. 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. Since (R•V) is the cosine of the angle between these two vectors, MAX((R•V),0) returns values in the range [0.0...1.0]. When these values are raised to the exponent Ns, the larger Ns is, the smaller the specular high-light; conversely the smaller Ns is, the larger the specular high-light. In other words, Ns controls the specular high-light size by controlling how sharp the equation (R•V)^Ns attenuates the light source color over the surface of a given polygon.

Illumination Models

This section discusses some of the implementation issues and technical limitations of the current illumination models available under HCVIS. The term “illumination model” as used by HCVIS refers to one of many available techniques for illuminating polygons with a light source. The illumination model is set with the -i parameter on the command line for HCVIS. Currently there are only two illumination models available.

Future releases of HCVIS will support additional illumination models.

Illumination Model 0

The illumination models currently supported by HCVIS are labeled 0 and 1 and are set using the -i parameter as mentioned in the section, “Command Line Parameters” located in Chapter One, “Getting Started.” Illumination model 0 indicates that lighting equations are not used in the rendering of data. The triangles of the triangulated mesh are colored using Gouraud shading where the color value at each vertex is identical to the value in the color field and is in no way modified.

Illumination Model 1

Illumination model 1 uses the lighting equations described in the section, “Lighting Model Equations,” located in this chapter. Color values at each triangle vertex of the triangulated mesh are adjusted according to these equations, then Gouraud shading is performed on the triangle using the adjusted color values. It is important to understand that illumination model 1 adjusts colors in a given triangle’s vertices based on that triangle’s normal. Adjacent triangles may have differing triangle normals, but obviously the vertices along the common edge between adjacent triangle have the same original color. These color values along the adjacent edge are adjusted differently because differing triange normals are used to adjust them even though the original color values at the vertices are the same. This technique is among the fastest available for simulating lighting of the surface, but as you can see it does have its limitations.

A better illumination model would find all triangles common to a given vertex and average the normals of these triangles together to come up with what’s known as a “vertex normal.” This vertex normal could then be used in the lighting equations to consistently adjust color values at triangle vertices with none of the adjacent edge limitations of illumination model 1. The new illumination model would produce vastly superior images, but at the cost of drastically slower rendering times. Nonetheless, it most likely will be supported in a future release of HCVIS.

Perspective Transformation Information

The following section uses pseudo code based on the Silicon Graphics Graphics Library (SGI GL) to derive the perspective transformation used by HCVIS. You must have knowledge of the perspective transform used by HCVIS in order to use HCVIS in advanced applications.

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

lrsize = 1.3333;      /* left,right size of window
btsize = 1.0;         /* bottom,top size of window
near = 5.0;           /* location of near clipping plane
far = 100.0;          /* far clipping plane
tz = -6.5;            /* view Z translation

/* The window (left,right,bottom,top,near,far) subroutine defines a perspective projection /*
transformation. The 4x4 homogeneous matrix used is:
/*
/* | 2*near                                                                    |
```

```
/* | ------    0                    0                    0             |
/* | right-left                                                         |
/* |                                                                    |
/* |                  2*near                                            |
/* | 0                ---            0                    0             |
/* |                  top-bottom                                        |
/* |                                                                    |
/* | right+left       top+bottom           far+near                     |
/* | ------    ------       - ----          -1             |
/* | right-left       top-bottom           far-near                     |
/* |                                                                    |
/* |                                                       2*far*near   |
/* | 0                0                    0                - ------   |
/* |                                                       far-near     |
/*
/* The above information is that which is documented in the Silicon Graphics GT GL Users
/* Guide V1.0, page B-3.

window(-lrsize,lrsize,-btsize,btsize,near,far);

/* premultiply the current matrix (i.e., window matrix) by a translate(x,y,z) 4x4
/* homogeneous matrix:
translate(0.0,0.0,tz);

/* compute some geometry normalization scaling parameters:
scl_norm = MAX((data_x_size - 1),(data_y_size - 1));
sx_norm = 1.0 / scl_norm;
sy_norm = sx_norm;
sz_norm = 1.0 / 255.0;

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

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

/* premultiply current matrix by geometry normalization scaling:
scale(sx_norm,sy_norm,sz_norm);

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

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

Future Enhancements

Many enhancements could be developed to improve the functionality and usability of HCVIS. For various reasons and design philosophies, HCVIS 1.1 was released in its current form without these enhancements. However HCVIS, 2.0 may contain many of the following capabilities in addition to the current capabilities in HCVIS 1.1:

- **Additional Illumination Models**

HCVIS 2.0 will support additional illumination models which will greatly improve the appearance of rendered images, but at the expense of reduced rendering speeds.

- **Interactive Interrogation of Data**

In HCVIS 2.0 a user will be able to use the mouse to “point and click” on the data to retrieve specific data values from areas of particular interest.

- **Interactive Setup and Manipulation**

Currently HCVIS 1.1 is command-line driven only. HCVIS 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 HCVIS 2.0.

- **Palette Display Option**

An option will be included in HCVIS 2.0 to overlay the palette being used to color the data. In other words, a small window specified by the user will display the current palette. This palette-window will appear within the window which the rendered image is drawn to.

- **Simultaneous Display of Multiple Height-Color Datasets**

With HCVIS 2.0 two or more datasets can be displayed simultaneously with different representations. For example, one dataset can be represented in Wireframe mode and overlayed on another dataset represented in Polygon mode, or two datasets can be displayed side-by-side. Most combinations of datasets and modes will be supported.