Some Other Useful Concepts

Prerequisites

Some knowledge of OpenGL modeling and rendering to be able to understand the concepts in these techniques and to see how they can affect images.

Introduction

OpenGL offers many features that allow the programmer to control the image creation process. In this module we discuss some of these that seem to be fairly independent of the more generalized concepts described in other areas. Placing some of these here may be an artifact of the piecemeal way these first drafts were created, but it seems important to mention them somewhere in these materials. In the long run it is likely that many of these will be moved into other areas, but for now, enjoy the potpourri!

Definitions

Window: the portion of the screen (or other drawing space, in case you are working on a different kind of graphic device) on which the drawing of your program is done. The details of window management are well hidden from the programmer so that OpenGL can work across many different platforms. It is easiest to delegate the window setup to the GLUT toolkit where much of the system-dependent parts of OpenGL are defined; the functions to do this are:

- `glutInitWindowSize(width, height);`
- `glutInitWindowPosition(topleftX, topleftY);`
- `glutCreateWindow("window title goes here");`

Here the variables are integers and represent values in screen coordinates. The window will be created in the place you designate on the screen and all drawing will then occur in this window.

One of the difficult problems in window management (difficult if you are working with the window system directly, that is) is reshaping the window — moving it to another place or making the window larger or smaller. These are handled easily by OpenGL because the computer generates an event whenever any of these window reshapes happens, and there is an event callback for window reshaping. We will discuss events and event callbacks in more detail later, but the reshape callback is registered by the function `glutReshapeFunc(reshape)` where the callback function `reshape(GLint w, GLint h)` does what is necessary to regenerate the image in the window. This can involve defining the projection and the viewing environment, it can include defining the viewport(s) for the image(s), or can delegate some of these to the display function. An example is included in the code sections below, but this is only one way the reshape callback function can be organized.

Viewport: a portion of the window in which drawing will be done. The default viewport is the entire window, but if you define a viewport before you define your drawing, that drawing will be done in the viewport region only. This has been used to create stereo pairs, with two side-by-side viewports are defined and images are created in each by defining separate views from each of two eye points. The function that defines a viewport is `glViewport(LLX, LLY, wide, high)` where `LLX` and `LLY` are the lower left x- and y-coordinates of the viewport relative to the window (both `GLint` values) and `wide` and `high` are the width and height of the viewport, respectively (both in `GLsizei` values). The viewport needs to be defined inside the reshape callback function because it must work with resized windows, and probably should be designed in terms relative to the size of the window, so the parameters of the reshape function should be used. The example of viewport use in the clipping figure below will show something of this use.
**Clipping**: the process of drawing with the portion of an image on one side of a plane drawn and the portion on the other side omitted. Recall that a plane is defined by a linear equation

\[ Ax + By + Cz + D = 0 \]

so it can be represented by the 4-tuple \((A, B, C, D)\). The plane divides the space into two parts: that for which \(Ax+By+Cz+D\) is positive and that for which it is negative. Any points for which this value is negative are not displayed; any points for which it is positive or zero are displayed.

You can view the restriction of drawing to the view frustum to be the application of six clipping planes, but OpenGL offers (at least) six additional clipping planes you can define yourself. The clipping planes are named `GL_CLIP_PLANEn` where \(n\) is between 0 and 5, and they are defined by the function `glClipPlane(plane, equation)` where `plane` is one of the pre-defined clipping planes above and `equation` is a vector of four `GLfloat` values. Once you have defined a clipping plane, it is enabled or disabled by a `glEnable(GL_CLIP_PLANEn)` function or equivalent `glDisable(...)` function. Clipping is performed when any modeling primitive is called when a clip plane is enabled; it is not performed when the clip plane is disabled. So you may choose when to clip and when not to clip on a clipping plane, as the example of clipping in the figure below shows.

![Figure: a clipped cone with the axes not clipped.](image)

This stereo pair also shows the use of two viewports (left and right images)

**Some examples**

**Defining a window and drawing to individual viewports**: This is quite straightforward; the code for defining the window is just what is noted above, except that in this particular case the window is defined to be twice as wide as it is high so that we can put two viewports in the window. Then in the display function, two viewports are defined and drawing is done to each. Note that the first viewport starts at the left-hand edge of the window, but the second viewport starts at the center of the window. This relationship is maintained in the window as it is reshaped; the `reshape(...)` function sets two global variables `vpw` and `vph` that are used to set the actual viewport in the `display()` function. This yields a window that is compatible with resizing and contains two side-by-side viewports so you can draw two images beside each other. In the particular example excerpted here and included with these modules, the program is to draw a stereo pair.

```c
void main(void)
{
    ...
    glutInitWindowSize(600,300);
    glutInitWindowPosition(70,70);
    glutCreateWindow("conic sections");
    ...
```
void reshape(int w, int h)
{
    vpw = w; vph = h;
    ...
    glutPostRedisplay();
}

void display( void )
{
    ...
    // left-hand viewport
    glViewport(0,0,vpw/2,vph);
    ...
    // right-hand viewport
    glViewport(vpw/2,0,vpw/2,vph);
    ...
    glutSwapBuffers();
}

Clipping an image against a clipping plane: here we consider some code that draws a cone and lets
the user define various clipping planes to explore conic sections. After the declaration of the
clipping plane as an array of GLdouble values, in the first section you will note that the display
plane is disabled before the axes are drawn but enabled before the cone is drawn. This ensures that
the entire axes are visible but part of the cone may not be. In the second section you will see that
you can set the coordinates of the clipping plane at any time in the program; in this particular case it
is being set in the menu handler as a result of a particular menu choice.

GLdouble myClipPlane[4];

void display( void )
{
    ...
    glDisable(GL_CLIP_PLANE1);
    axes();
    glEnable(GL_CLIP_PLANE1);
    glClipPlane(GL_CLIP_PLANE1, myClipPlane);
    drawCone();
    glutSwapBuffers();
}

void options_menu(int input)
{
    ...
    if (input == Tag) { // menu choice identifies a clip plane
        myClipPlane[0] = 0.0;
        myClipPlane[1] = -4.0;
        myClipPlane[2] = 3.0;
        myClipPlane[3] = -2.0;
    }
    ...
}

A word to the wise...

As we said, OpenGL has many more techniques than we can cover in a set of notes for a beginning
course in graphics, and many times a technique we present will have other applications that we do
not cover. Keep an open mind and keep your eyes open for clever applications whose source code is openly available; there’s a huge set of resources on the Web that we cannot begin to cover.

But as you look over these examples, keep in mind that you should master a good set of simple techniques before you jump into some of the sophisticated ways OpenGL can be used. There are subtle ways that sophisticated things can fail, and unless you have a sound grounding you may find yourself frustrated because a technique you are trying just doesn’t work the way you think it should.

**Code examples**

We include three full code examples. The stereo-pair example is perhaps not as clean as it might be because of the details of managing two images at once, but we hope it is useful.

- **conicStereo.c**: dual-viewport clipping example that also includes menus and keyboard