3D Stereo Rendering Using OpenGL (and GLUT)

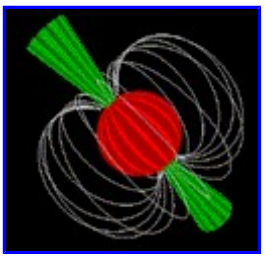
See also, Calculating Stereo Pairs

Source code for the incorrect (but close) "Toe-in" stereo, and the correct Offaxis stereo

Written by Paul Bourke
November 1999
Updated May 2002

Introduction

The following is intended to get someone started creating 3D stereo applications using OpenGL and the associated GLUT library. It is assumed that the reader is both familiar with how to create the appropriate eye positions for comfortable stereo viewing (see link in the title of the page) and the reader has an OpenGL card (or software implementation) and any associated hardware (eg: glasses) needed to support stereo graphic viewing.



The description of the code presented here will concentrate on the stereo aspects, the example does however create a real, time varying OpenGL object, namely the pulsar model shown on the right. The example also contains examples of mouse and keyboard controls of the camera position. The example is not intended to illustrate more advanced OpenGL techniques and indeed it does not do things particularly efficiently, in particular it should, but does not, use display lists. The example does not use textures as that is a large separate topic and would only confuse the task at hand.

Conventions

The example code conforms to a couple of local conventions. The first is that it can be run in a window or full screen (arcade game) mode. By convention the application runs in a window unless the "-f" command line option is specified. Full screen mode is supported in the most recent versions of the GLUT library. The decision to use full screen mode is made with the following snippet.

```
glutCreateWindow("Pulsar model");
glutReshapeWindow(600,400);
if (fullscreen)
   glutFullScreen();
```

It is also useful to be able to run the application in stereo mode or mono mode, the convention is to run in mono unless the command line switch "-s" is supplied. The full usage help information in the example presented here is available by running the application with the "-h" command line option, for example:

```
>pulsar -h
Usage: pulsar [-h] [-f] [-s] [-c]
          -h   this text
          -f   full screen
          -s   stereo
          -c   show construction lines
Key Strokes
  arrow keys   rotate left/right/up/down
  left mouse   rotate
middle mouse   roll
           c   toggle construction lines
           i   translate up
           k   translate down
           j   translate left
           l   translate right
           [   roll clockwise
           ]   roll anti clockwise
           q   quit
```

Stereo

The first thing that needs to be done to support stereo is to initialise the GLUT library for stereo operation. If your card/driver combination don't support stereo this will fail.

```
glutInit(&argc,argv);
if (!stereo)
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
else
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH | GLUT_STEREO);
```

In stereo mode this defines two buffers namely GL_BACK_LEFT and GL_BACK_RIGHT. The appropriate buffer is selected before operations that would affect it are performed, this is using the routine glDrawBuffer(). So for example to clear the two buffers:

```
glDrawBuffer(GL_BACK_LEFT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
if (stereo) {
   glDrawBuffer(GL_BACK_RIGHT);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}
```

Note that some cards (eg: Powerstorm 4D51T) are optimised to clear both left and right buffers if GL_BACK is cleared, this can be significantly faster. In these cases one clears the buffers as follows.

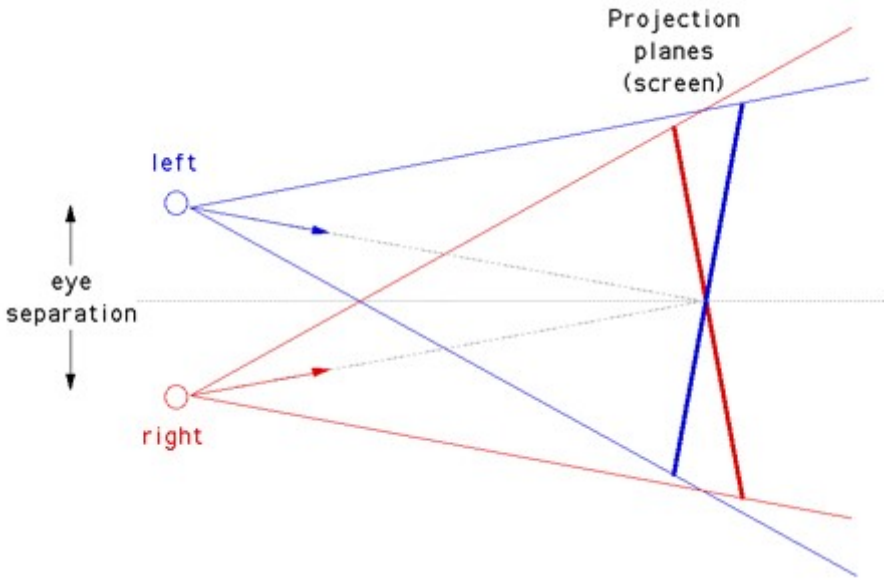
```
glDrawBuffer(GL_BACK);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Projection

All that's left now is to render the geometry into the appropriate buffer. There are many ways this can be organised depending on the way the particular application is written, in this example see the Display() handler. Essentially the idea is to select the appropriate buffer and render the scene with the appropriate projection.

Toe-in Method

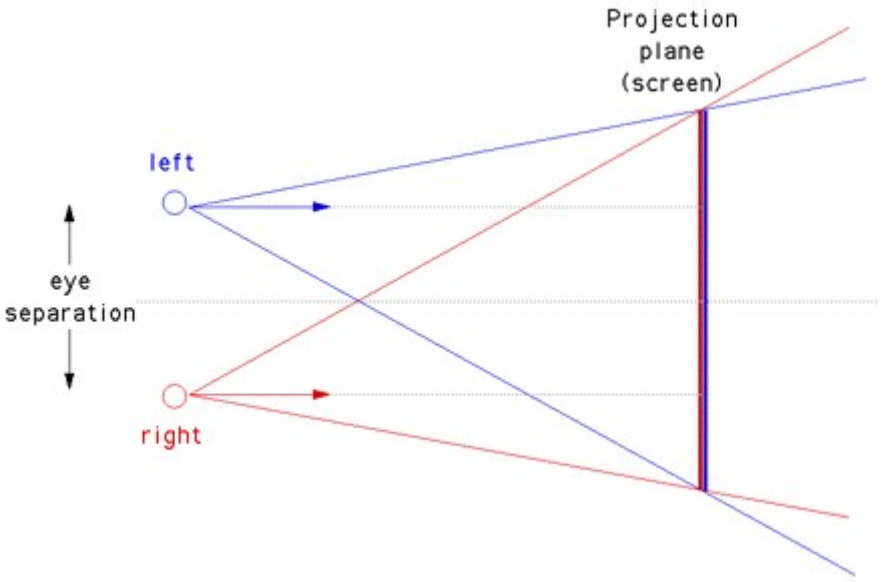
A common approach is the so called "toe-in" method where the camera for the left and right eye is pointed towards a single focal point and gluPerspective() is used.




```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(camera.aperture,screenwidth/(double)screenheight,0.1,10000.0);

if (stereo) {

   CROSSPROD(camera.vd,camera.vu,right);
   Normalise(&right);
   right.x *= camera.eyesep / 2.0;
   right.y *= camera.eyesep / 2.0;
   right.z *= camera.eyesep / 2.0;

   glMatrixMode(GL_MODELVIEW);
   glDrawBuffer(GL_BACK_RIGHT);
   glLoadIdentity();
   gluLookAt(camera.vp.x + right.x,
             camera.vp.y + right.y,
             camera.vp.z + right.z,
             focus.x,focus.y,focus.z,
             camera.vu.x,camera.vu.y,camera.vu.z);
   MakeLighting();
   MakeGeometry();

   glMatrixMode(GL_MODELVIEW);
   glDrawBuffer(GL_BACK_LEFT);
   glLoadIdentity();
   gluLookAt(camera.vp.x - right.x,
             camera.vp.y - right.y,
             camera.vp.z - right.z,
             focus.x,focus.y,focus.z,
             camera.vu.x,camera.vu.y,camera.vu.z);
   MakeLighting();
   MakeGeometry();
} else {
   glMatrixMode(GL_MODELVIEW);
   glDrawBuffer(GL_BACK_LEFT);
   glLoadIdentity();
   gluLookAt(camera.vp.x,
             camera.vp.y,
             camera.vp.z,
             focus.x,focus.y,focus.z,
             camera.vu.x,camera.vu.y,camera.vu.z);
   MakeLighting();
```

```
   MakeGeometry();
}

/* glFlush(); This isn't necessary for double buffers */
glutSwapBuffers();
```

Correct method



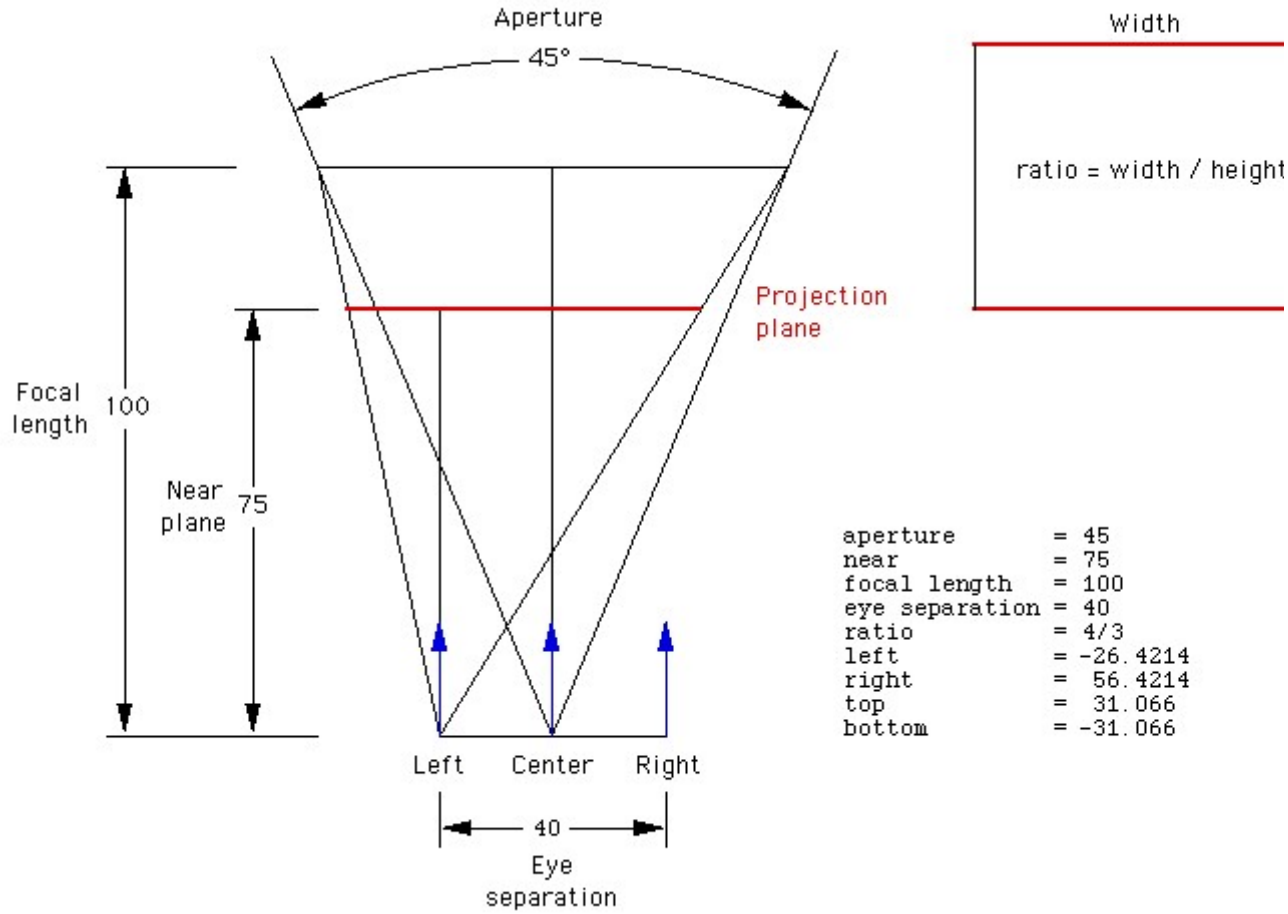
The Toe-in method while giving workable stereo pairs is not correct, it also introduces vertical parallax which is most noticeable for objects in the outer field of view. The correct method is to use what is sometimes known as the ″parallel axis asymmetric frustum perspective projection″. In this case the view vectors for each camera remain parallel and a glFrustum() is used to describe the perspective projection.

```
/* Misc stuff */
ratio  = camera.screenwidth / (double)camera.screenheight;
radians = DTOR * camera.aperture / 2;
wd2     = near * tan(radians);
ndfl    = near / camera.focallength;

/* Clear the buffers */
glDrawBuffer(GL_BACK_LEFT);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
if (stereo) {
   glDrawBuffer(GL_BACK_RIGHT);
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
}

if (stereo) {

   /* Derive the two eye positions */
   CROSSPROD(camera.vd,camera.vu,r);
   Normalise(&r);
   r.x *= camera.eyesep / 2.0;
   r.y *= camera.eyesep / 2.0;
   r.z *= camera.eyesep / 2.0;

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   left  = - ratio * wd2 - 0.5 * camera.eyesep * ndfl;
   right =   ratio * wd2 - 0.5 * camera.eyesep * ndfl;
   top    =   wd2;
   bottom = - wd2;
   glFrustum(left,right,bottom,top,near,far);
```

```
      glMatrixMode(GL_MODELVIEW);
      glDrawBuffer(GL_BACK_RIGHT);
      glLoadIdentity();
      gluLookAt(camera.vp.x + r.x,camera.vp.y + r.y,camera.vp.z + r.z,
                camera.vp.x + r.x + camera.vd.x,
                camera.vp.y + r.y + camera.vd.y,
                camera.vp.z + r.z + camera.vd.z,
                camera.vu.x,camera.vu.y,camera.vu.z);
      MakeLighting();
      MakeGeometry();

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      left  = - ratio * wd2 + 0.5 * camera.eyesep * ndfl;
      right =   ratio * wd2 + 0.5 * camera.eyesep * ndfl;
      top    =   wd2;
      bottom = - wd2;
      glFrustum(left,right,bottom,top,near,far);

      glMatrixMode(GL_MODELVIEW);
      glDrawBuffer(GL_BACK_LEFT);
      glLoadIdentity();
      gluLookAt(camera.vp.x - r.x,camera.vp.y - r.y,camera.vp.z - r.z,
                camera.vp.x - r.x + camera.vd.x,
                camera.vp.y - r.y + camera.vd.y,
                camera.vp.z - r.z + camera.vd.z,
                camera.vu.x,camera.vu.y,camera.vu.z);
      MakeLighting();
      MakeGeometry();

   } else {

      glMatrixMode(GL_PROJECTION);
      glLoadIdentity();
      left  = - ratio * wd2;
      right =   ratio * wd2;
      top    =   wd2;
      bottom = - wd2;
      glFrustum(left,right,bottom,top,near,far);

      glMatrixMode(GL_MODELVIEW);
      glDrawBuffer(GL_BACK_LEFT);
      glLoadIdentity();
      gluLookAt(camera.vp.x,camera.vp.y,camera.vp.z,
                camera.vp.x + camera.vd.x,
                camera.vp.y + camera.vd.y,
                camera.vp.z + camera.vd.z,
                camera.vu.x,camera.vu.y,camera.vu.z);
      MakeLighting();
      MakeGeometry();
   }

   /* glFlush(); This isn't necessary for double buffers */
   glutSwapBuffers();
```

Note that sometimes it is appropriate to use the left eye position when not in stereo mode in which case the above code can be simplified. It seems more elegant and consistent when moving between mono and stereo if the point between the eyes is used when in mono.

On the off chance that you want to write the code differently and would like to test the correctness of the glFrustum() parameters, here's an explicit example.



Passive stereo

Updated in May 2002: sample code to deal with passive stereo, that is, drawing the left eye to the left half of a dual display OpenGL card and the right eye to the right half. pulsar2.c and pulsar2.h

Macintosh OS-X example

Source code and Makefile illustrating stereo under Mac OS-X using "blue line" syncing, contributed by Jamie Cate.

Demonstration stereo application for Mac OS-X from the Apple development site based upon the method and code described above: <a href="GLUTStereo/’>GLUTStereo. (Also uses blue line syncing)