Stereographics

Mike Bailey
mjb@cs.oregonstate.edu
Oregon State University

Stereovision is not new –
It's been in common use since the 1950s,
and sporadically even before that

In everyday living, part of our perception of depth comes from the slight difference in how our two eyes see the world around us. This is known as binocular vision. We care about this, and are discussing it, because stereo computer graphics can be a great help in de-cluttering a complex 3D visualization scene.

Binocular Vision

In the world of computer graphics, the two eye views can be reconstructed using standard projection mathematics. The simplest approach is the Cyclops Model. In this model, the left and right eye views are obtained by rotating the scene plus and minus what a Cyclops at the origin would see. Looking at everything from the top:

The left eye view is obtained by rotating the scene an angle +Ø about the Y axis. The right eye view is obtained by rotating the scene an angle -Ø about the Y axis. In practice, a good value of Ø is 1-4˚.

The Vertical Parallax Problem

This seems too simple, and in fact, it is. This works OK if you are doing orthographic projections, but if you use perspective, you will achieve a nasty phenomenon called vertical parallax, as illustrated below:

The fact that the perspective shortening causes points A and B to have different vertical positions in the left and right eye views makes it very difficult for the eyes to converge the two images. For perspective projections, we need a better way.

Why not just keep using orthographic projections? Mathematically this is fine, but in practice, the two depth cues, stereo and no-perspective, fight each other. This will bring on an optical illusion. A good example of this is a simple cube, drawn below using an orthographic projection:

Because of the use of stereographics, the binocular cues will say that the Near face is closer to the viewer than is the Far face. However, our visual experience says that the only way a far object can appear the same size as a near object is if it is, in fact, larger. Thus, your visual system will perceive the Far face as being larger than the Near face, when in fact they are the same size.
Diversion #1 – Specifying the Viewing Frustum
The OpenGL glFrustum call can be used in place of gluPerspective:

```c
void FrustumZ( float left, float right, float bottom, float top, float znear, float zfar, float zproj );
```

Rather than having to specify the left, right, bottom, and top limits at the near clipping plane (which is what glFrustum expects), let’s set up a way to specify those limits at a particular distance in front of us. (This is defined using similar triangles.)

```
float znear, zfar;
FrustumZ( left, right, bottom, top, znear, zfar, zproj );
```

So, if you wanted to view a car from 30 feet away, you could say:

```
FrustumZ( -10., 10., -10., 10., .1, 100., 30. );
```

Diversion #2 – Where does a 3D Point Map in a 2D Window?
Take an arbitrary 3D point in the viewing volume. Place a plane parallel to the near and far clipping planes at its Z value (i.e., depth in the frustum). The location of the point on that plane shows proportionally where the 3D point will be perspective-mapped from left to right in the 2D window.

```
T0p = Z0p * tan( ø/2 )
B0p = -Z0p * tan( ø/2 )
R0p = Z0p * tan( ø/2 )
L0p = -Z0p * tan( ø/2 )
```

Two Side-by-side Perspective Viewing Volumes
The best stereographics work is done with perspective projections. To avoid the vertical parallax problem, we keep both the left and right eyes looking straight ahead so that, in the vertical parallax example shown before, points A and B will project with exactly the same amount of shortening.

```
Prime = X0p
```

Two Side-by-side Perspective Viewing Volumes
We do this by defining a distance in front of the eye, Z0p, to the plane of zero parallax, where a 3D point projects to the same window location for each eye. To the viewer, the plane of zero parallax will be the glass screen and objects in front of it will appear to live in the air in front of the glass screen and objects behind this plane will appear to live inside the monitor.

```
1. Set the distance from the eyes to the plane of zero parallax based on the location of the geometry and the look you are trying to achieve.
2. Looking from the Cyclops eye at the origin, determine the left, right, bottom, and top boundaries of the viewing window on the plane of zero parallax as would be used in a call to glFrustum(). These can be determined by knowing Z0p and the field-of-view angle ø:
```

Two Side-by-side Perspective Viewing Volumes
The left eye sees the box towards the far right side of its display.

The right eye sees the box towards the far left side of its display.

The left eye view is obtained by translating the eye by -E in the X direction, which is actually accomplished by translating the scene by E instead. Similarly, the right eye view is obtained by translating the scene by E in the X direction. We now have a horizontal parallax situation, where the same point projects to a different horizontal position in the left and right eye views. Note that this is a situation not a problem. The difference in the left and right eye views requires at least some horizontal parallax to work. You can convince yourself of this by alternately opening and closing your left and right eyes. We just need a good way to control the horizontal parallax.
An Example

Parallel viewing stereo

Cross-eye viewing stereo

Acquiring Stereo Video

Quad-Buffered OpenGL

Remember double buffering, where you draw into the back buffer and display from the front buffer? OpenGL actually has two back buffers and two front buffers, one for each eye. So, draw the left eye view into GL_BACK_LEFT and the right eye view into GL_BACK_RIGHT. First you need to tell GLUT that you are doing stereo graphics. In InitGraphics():

```c
glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH | GLUT_STEREO );
```

Then go ahead and create the window as normal. After creating the window, you can also expand it to be the full screen with:

```c
glutFullScreen( );
```

In Display(), you need to clear both buffers:

```c
glDrawBuffer( GL_BACK_LEFT );
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```c
glDrawBuffer( GL_BACK_RIGHT );
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

```c
for( int eye = 0; eye <= 1; eye++ ) {
    glMatrixMode( GL_PROJECTION );
    glLoadIdentity();
    if( eye == 0 )  // left eye view
    {
        Stereopersp(fovy, 1.0, znear, zfar, z0p, -eyesep);
    }
    else          // right eye view
    {
        Stereopersp(fovy, 1.0, znear, zfar, z0p, eyesep);
    }
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    << draw the 3D scene >>
    glutSwapBuffers( );
}
```

Separating the Left and Right-eye Views — Shutterglasses

In Display(), you also need to draw into both back buffers:

```c
for( int eye = 0; eye <= 1; eye++ ) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if( eye == 0 )  // left eye view
    {
        glDrawBuffer( GL_BACK_LEFT );
        Stereopersp(fovy, 1.0, znear, zfar, z0p, -eyesep);
    }
    else          // right eye view
    {
        glDrawBuffer( GL_BACK_RIGHT );
        Stereopersp(fovy, 1.0, znear, zfar, z0p, eyesep);
    }
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    <-- draw the 3D scene >>
    glutSwapBuffers( );
}
```

Quad-Buffered OpenGL

In Display(), you need to clear both buffers:

```c
gDrawBuffer( GL_BACK_LEFT );
gClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
gDrawBuffer( GL_BACK_RIGHT );
gClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```

Separating the Left and Right-eye Views — Shutterglasses

In Display(), you also need to draw into both back buffers:

```c
for( int eye = 0; eye <= 1; eye++ ) {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if( eye == 0 )  // left eye view
    {
        glDrawBuffer( GL_BACK_LEFT );
        Stereopersp(fovy, 1.0, znear, zfar, z0p, -eyesep);
    }
    else          // right eye view
    {
        glDrawBuffer( GL_BACK_RIGHT );
        Stereopersp(fovy, 1.0, znear, zfar, z0p, eyesep);
    }
    glMatrixMode( GL_MODELVIEW );
    glLoadIdentity( );
    <-- draw the 3D scene >>
    glutSwapBuffers( );
}
```
Separating the Left and Right-eye Views – Head-mounted Goggles

Separating the Left and Right-eye Views – the Stereo Mirror

Separating the Left and Right-eye Views – Dual Projectors ("GeoWall")

Separating the Left and Right-eye Views – Stereo Movie Projectors

Separating the Left and Right-eye Views – Stereo Movie Projectors

Two filters statically provide the polarization

One filter dynamically provides the polarization

For movies and sporting events
Separating the Left and Right-eye Views – Left-Right 3DTV

Separating the Left and Right-eye Views – Top-Bottom 3DTV

Separating the Left and Right-eye Views – Interlaced 3DTV

Separating the Left and Right-eye Views – Red-Cyan Anaglyphs

Separating the Left and Right-eye Views – ColorCode™

This is what was used at the 2009 Super Bowl halftime!

Encoding Stereo in a Single Image – ChromaDepth™
Stereographics Rules of Thumb

• Stereographics is especially good for de-cluttering wireframe displays.
• Use perspective, not orthographic, projections to avoid the optical illusion.
• Use an eye separation, \( E \), of approximately:
  \[ E = Z_{0p} \cdot \tan(1\degree) \]
• Use the far clipping plane well. The stereo effects are enhanced when the scene is not complicated by a lot of tiny detail that is far away. The interactive response is improved too.
• Because you are drawing the scene twice, using display lists is especially important.
• It is fun to set \( Z_{0p} = Z_{far} \) so that the image appears to be hanging out in the air in front of the monitor. However, in real life we rarely see anything hanging out in the air that has its sides clipped for no apparent reason, as the scene is likely to have. Perceptually, it is often better to set \( Z_{0p} = \) Zero so that the entire scene looks like it is inside the monitor and that you are viewing it through a rectangular hole cut through the glass. This situation is common in everyday life, so we are used to seeing things that way.
• Intensity depth cueing (\text{glFog}) nicely enhances the stereo illusion.
• If you are using texture mapping, be sure to use \text{GL_LINEAR}, not \text{GL_NEAREST}, for the texture filtering.

From what you now know, real stereo images have to be generated from the original data – they cannot be as effectively retro-generated from a mono image, at least not without a lot of work.

Beware the multiple plane effect!
Watch out for a lot of monoscopic movies being “re-released” in stereo!