Sphere Mapping Quadrics In OpenGL

Like 3 G+1 +3

Sphere Environment Mapping is a quick way to add a reflection to a metallic or reflective object in your scene. Although it is not as accurate as real life or as a Cube Environment Map, it is a whole lot faster! We'll be using the code from lesson eighteen (Quadrics) for the base of this tutorial. Also we're not using any of the same texture maps, we're going to use one sphere map, and one background image.

Before we start... The "red book" defines a Sphere map as a picture of the scene on a metal ball from infinite distance away and infinite focal point. Well that is impossible to do in real life. The best way I have found to create a good sphere map image without using a Fish eye lens is to use Adobe's Photoshop program.

Creating a Sphere Map In Photoshop:

First you will need a picture of the environment you want to map onto the sphere. Open the picture in Adobe Photoshop and select the entire image. Copy the image and create a new PSD (Photoshop Format) the new image should be the same size as the image we just copied. Paste a copy of the image into the new window we've created. The reason we make a copy is so Photoshop can apply its filters. Instead of copying the image you can select mode from the drop down menu and choose RGB mode. All of the filters should then be available.

Next we need to resize the image so that the image dimensions are a power of 2. Remember that in order to use an image as a texture the image needs to be 128x128, 256x256, etc. Under the image menu, select image size, uncheck the constraint proportions checkbox, and resize the image to a valid texture size. If your image is 100X90, it's better to make the image 128x128 than 64x64. Making the image smaller will lose alot of detail.

The last thing we do is select the filter menu, select distort and apply a spherize modifier. You should see that the center of the picture is blown up like a balloon, now in normal sphere maps the outer area will be blackened out, but it doesn't really matter. Save a copy of the image as a .BMP and you're ready to code!

We don't add any new global variables this time but we do modify the texture array to hold 6 textures.

?
```
GLuint  texture[6];                                 // Storage For 6 Textures
```

The next thing I did was modify the LoadGLTextures() function so we can load in 2 bitmaps and create 3 filters. (Like we did in the original texturing tutorials). Basically we loop through twice and create 3 textures each time using a different filtering mode. Almost all of this code is new or modified.

?
```
int LoadGLTextures()                                    // Load Bitmaps And Convert To Textures
{
    int Status=FALSE;                               // Status Indicator

    AUX_RGBImageRec *TextureImage[2];                   // Create Storage Space For The Texture

    memset(TextureImage,0,sizeof(void *)*2);                    // Set The Pointer To NULL

    // Load The Bitmap, Check For Errors, If Bitmap's Not Found Quit
    if ((TextureImage[0]=LoadBMP("Data/BG.bmp")) &&             // Background Texture
        (TextureImage[1]=LoadBMP("Data/Reflect.bmp")))          // Reflection Texture (Spheremap)
    {
        Status=TRUE;                                // Set The Status To TRUE

        glGenTextures(6, &texture[0]);                  // Create Three Textures (For Two Images)

        for (int loop=0; loop<2; loop++)
        {
            // Create Nearest Filtered Texture
            glBindTexture(GL_TEXTURE_2D, texture[loop]);        // Gen Tex 0 And 1
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
            glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[loop]->sizeX,
TextureImage[loop]->sizeY,
                0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage[loop]->data);

            // Create Linear Filtered Texture
            glBindTexture(GL_TEXTURE_2D, texture[loop+2]);      // Gen Tex 2, 3 And 4
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
            glTexImage2D(GL_TEXTURE_2D, 0, 3, TextureImage[loop]->sizeX,
TextureImage[loop]->sizeY,
                0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage[loop]->data);

            // Create MipMapped Texture
            glBindTexture(GL_TEXTURE_2D, texture[loop+4]);      // Gen Tex 4 and 5
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
            glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR_MIPMAP_NEAREST);
            gluBuild2DMipmaps(GL_TEXTURE_2D, 3, TextureImage[loop]->sizeX,
TextureImage[loop]->sizeY,
                GL_RGB, GL_UNSIGNED_BYTE, TextureImage[loop]->data);
        }
        for (loop=0; loop<2; loop++)
        {
            if (TextureImage[loop])                     // If Texture Exists
            {
                    if (TextureImage[loop]->data)            // If Texture Image Exists
                    {
                            free(TextureImage[loop]->data);  // Free The Texture Image Memory
                    }
                    free(TextureImage[loop]);       // Free The Image Structure
            }
        }
    }

    return Status;                              // Return The Status
}
```

We'll modify the cube drawing code a little. Instead of using 1.0 and -1.0 for the normal values, we'll use 0.5 and -0.5. By changing the value of the normal, you can zoom the reflection map in and out. If the normal value is high, the image being reflected will be bigger, and may appear blocky. By reducing the normal value to 0.5 and -0.5 the reflected image is zoomed out a bit so that the image reflecting off the cube isn't all blocky looking. Setting the normal value too low will create undesirable results.

?

```
GLvoid glDrawCube()
{
        glBegin(GL_QUADS);
        // Front Face
        glNormal3f( 0.0f, 0.0f, 0.5f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
        // Back Face
        glNormal3f( 0.0f, 0.0f,-0.5f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
        // Top Face
        glNormal3f( 0.0f, 0.5f, 0.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
        // Bottom Face
        glNormal3f( 0.0f,-0.5f, 0.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
        // Right Face
        glNormal3f( 0.5f, 0.0f, 0.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( 1.0f, -1.0f, -1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f( 1.0f,  1.0f, -1.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f( 1.0f,  1.0f,  1.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f( 1.0f, -1.0f,  1.0f);
        // Left Face
        glNormal3f(-0.5f, 0.0f, 0.0f);
        glTexCoord2f(0.0f, 0.0f); glVertex3f(-1.0f, -1.0f, -1.0f);
        glTexCoord2f(1.0f, 0.0f); glVertex3f(-1.0f, -1.0f,  1.0f);
        glTexCoord2f(1.0f, 1.0f); glVertex3f(-1.0f,  1.0f,  1.0f);
        glTexCoord2f(0.0f, 1.0f); glVertex3f(-1.0f,  1.0f, -1.0f);
    glEnd();
}
```

Now in InitGL we add two new function calls, these two calls set the texture generation mode for S and T to Sphere Mapping. The texture coordinates S, T, R & Q relate in a way to object coordinates x, y, z and w. If you are using a one-dimensional texture (1D) you will use the S coordinate. If your texture is two dimensional, you will use the S & T coordinates.

So what the following code does is tells OpenGL how to automatically generate the S and T coordinates for us based on the sphere-mapping formula. The R and Q coordinates are usually ignored. The Q coordinate can be used for advanced texture mapping extensions, and the R coordinate may become useful once 3D texture mapping has been added to OpenGL, but for now we will ignore the R & Q Coords. The S coordinate runs horizontally across the face of our polygon, the T coordinate runs vertically across the face of our polygon.

?

```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);            // Set The Texture Generation Mode
For S To Sphere Mapping ( NEW )
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);            // Set The Texture Generation Mode
For T To Sphere Mapping ( NEW )
```

We're almost done! All we have to do is set up the rendering, I took out a few of the quadratic objects because they didn't work well with environment mapping. The first thing we need to do is enable texture generation. Then we select the reflective texture (sphere map) and draw our object. After all of the objects you want sphere-mapped have been drawn, you will want to disable texture generation, otherwise everything will be sphere mapped. We disable sphere-mapping before we draw the background scene (we don't want the background sphere mapped). You will notice that the bind texture commands may look fairly complex. All we're doing is selecting the filter to use when drawing our sphere map or the background image.

?

```
int DrawGLScene(GLvoid)                             // Here's Where We Do All The Drawing
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);         // Clear The Screen And The Depth
Buffer
    glLoadIdentity();                           // Reset The View

    glTranslatef(0.0f,0.0f,z);

    glEnable(GL_TEXTURE_GEN_S);                     // Enable Texture Coord Generation For S ( NEW
)
    glEnable(GL_TEXTURE_GEN_T);                     // Enable Texture Coord Generation For T ( NEW
)

    glBindTexture(GL_TEXTURE_2D, texture[filter+(filter+1)]);       // This Will Select A Sphere
Map
    glPushMatrix();
    glRotatef(xrot,1.0f,0.0f,0.0f);
    glRotatef(yrot,0.0f,1.0f,0.0f);
    switch(object)
    {
    case 0:
        glDrawCube();
        break;
    case 1:
        glTranslatef(0.0f,0.0f,-1.5f);                  // Center The Cylinder
        gluCylinder(quadratic,1.0f,1.0f,3.0f,32,32);            // A Cylinder With A Radius Of 0.5
And A Height Of 2
        break;
    case 2:
        gluSphere(quadratic,1.3f,32,32);                // Sphere With A Radius Of 1 And 16
Longitude/Latitude Segments
        break;
    case 3:
        glTranslatef(0.0f,0.0f,-1.5f);                  // Center The Cone
        gluCylinder(quadratic,1.0f,0.0f,3.0f,32,32);            // Cone With A Bottom Radius Of .5
And Height Of 2
        break;
    };

    glPopMatrix();
    glDisable(GL_TEXTURE_GEN_S);                        // Disable Texture Coord Generation ( NEW
)
    glDisable(GL_TEXTURE_GEN_T);                        // Disable Texture Coord Generation ( NEW
)

    glBindTexture(GL_TEXTURE_2D, texture[filter*2]);            // This Will Select The BG Texture
( NEW )
    glPushMatrix();
        glTranslatef(0.0f, 0.0f, -24.0f);
        glBegin(GL_QUADS);
            glNormal3f( 0.0f, 0.0f, 1.0f);
            glTexCoord2f(0.0f, 0.0f); glVertex3f(-13.3f, -10.0f,  10.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f( 13.3f, -10.0f,  10.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f( 13.3f,  10.0f,  10.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(-13.3f,  10.0f,  10.0f);
        glEnd();

    glPopMatrix();

    xrot+=xspeed;
    yrot+=yspeed;
    return TRUE;                                // Keep Going
}
```

The last thing we have to do is update the spacebar section of code to reflect (No Pun Intended) the changes we made to the Quadratic objects being rendered. (We removed the discs)

?

```
if (keys[' '] && !sp)
{
    sp=TRUE;
    object++;
    if(object>3)
        object=0;
}
```

and at the very end of the program we need to delete the quadric to prevent memory leaks.

?

```
    // Shutdown
    gluDeleteQuadric(quadratic);
    KillGLWindow();                             // Kill The Window
    return (msg.wParam);                            // Exit The Program
}

gluDeleteQuadric(quadratic);
```

We're done! Now you can do some really impressive things with Environment mapping like making an almost accurate reflection of a room! I was planning on showing how to do Cube Environment Mapping in this tutorial too but my current video card does not support cube mapping. Maybe in a month or so after I buy a GeForce 2 :) Also I taught myself environment mapping (mostly because I couldnt find too much information on it) so if anything in this tutorial is inaccurate, Email Me or let NeHe know.

Thanks, and Good Luck!

Visit my site: /data/lessons/http://www.tiptup.com/

GB Schmick (**TipTup**)

Jeff Molofee (**NeHe**)

* DOWNLOAD Visual C++ Code For This Lesson.

* DOWNLOAD Delphi Code For This Lesson. (Conversion by Alexandre Hirzel)
* DOWNLOAD Code Warrior 5.3 Code For This Lesson. (Conversion by Scott Lupton)
* DOWNLOAD Delphi Code For This Lesson. (Conversion by Michal Tucek)
* DOWNLOAD Dev C++ Code For This Lesson. (Conversion by Dan)
* DOWNLOAD Euphoria Code For This Lesson. (Conversion by Evan Marshall)
* DOWNLOAD Game GLUT Code For This Lesson. (Conversion by Anonymous)
* DOWNLOAD Java Code For This Lesson. (Conversion by Chris Veenboer)
* DOWNLOAD LCC Win32 Code For This Lesson. (Conversion by Robert Wishlaw)
* DOWNLOAD Linux/SDL Code For This Lesson. (Conversion by Arkadi Shishlov)

* DOWNLOAD LWJGL Code For This Lesson. (Conversion by Mark Bernard)
* DOWNLOAD Mac OS X/Cocoa Code For This Lesson. (Conversion by Bryan Blackburn)
* DOWNLOAD Mac OS X/Cocoa Code For This Lesson. (Conversion by Bryan Blackburn)
* DOWNLOAD Visual Studio .NET Code For This Lesson. (Conversion by Grant James)

< Lesson 22Lesson 24 >

© 1997-2014 Gamedev . All rights reserved.
NeHe™ and NeHe Productions™ are trademarks of GameDev.net, LLC
OpenGL® is a registered trademark of Silicon Graphics Inc.