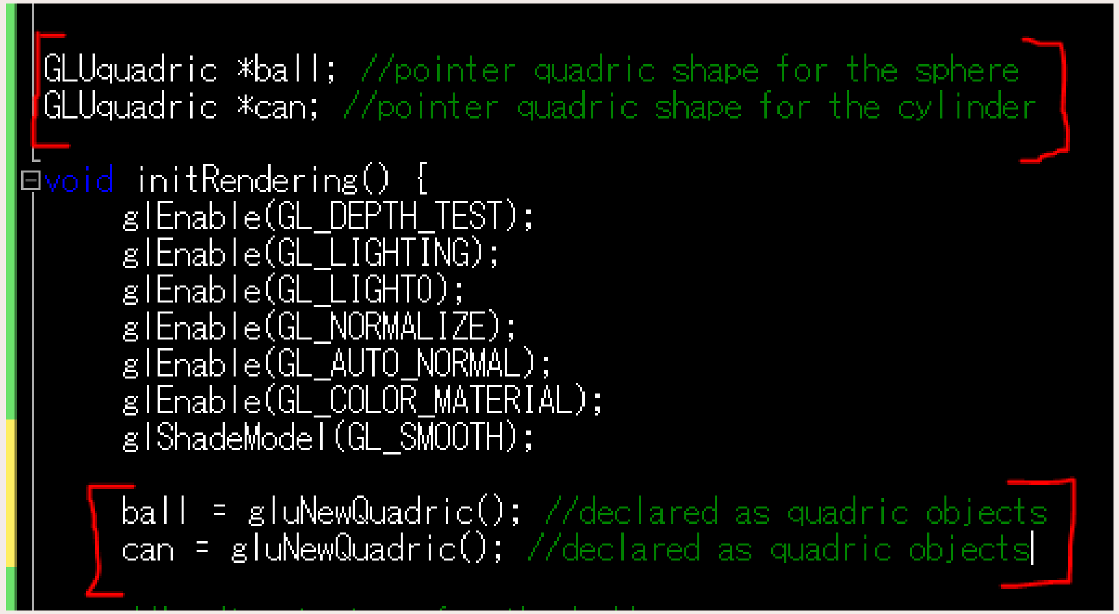
Quadric Shapes and Multi-Texturing

Mark Daniel Dacer

Quadric creation

- To creates quadric shapes with GLU, just begin with the creation of a quadric object with gluNewQuadric :
- gluNewQuadric()



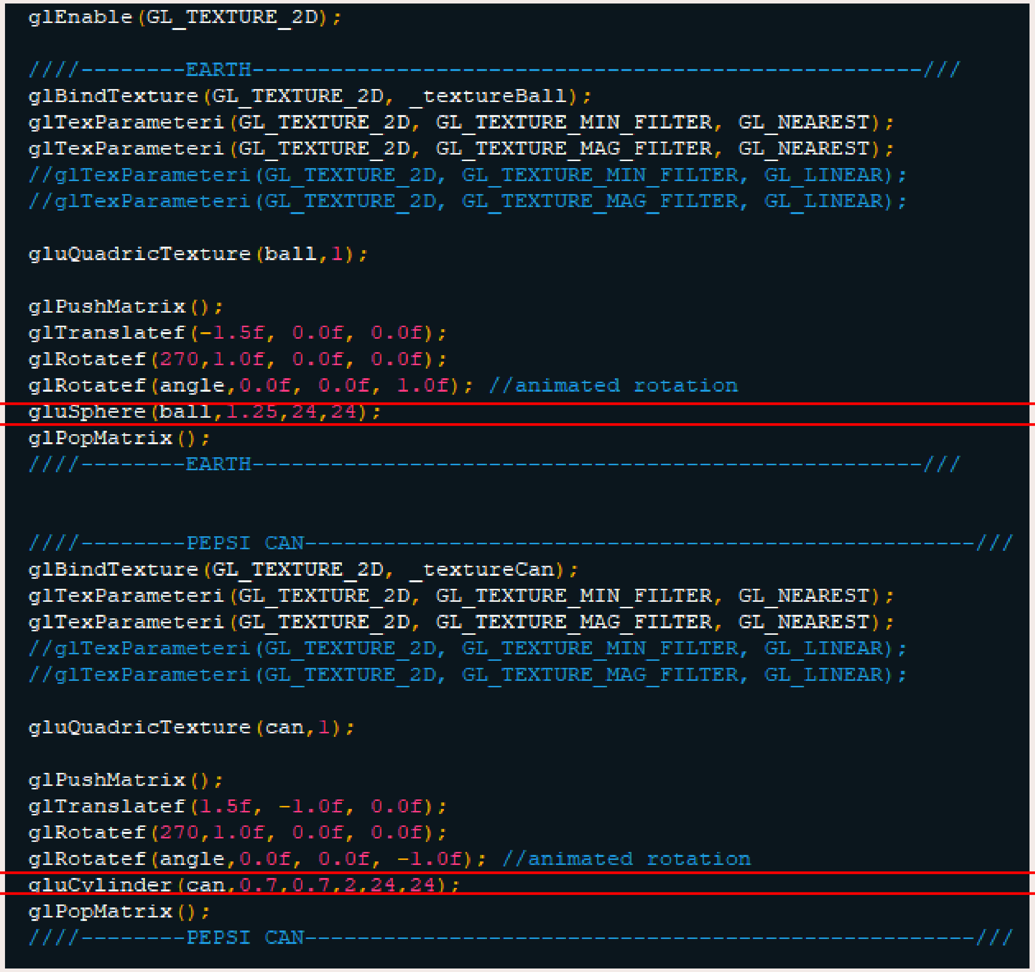
```
GLUquadric *ball; //pointer quadric shape for the sphere
GLUquadric *can; //pointer quadric shape for the cylinder

void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_COLOR_MATERIAL);
    glShadeModel(GL_SMOOTH);

    ball = gluNewQuadric(); //declared as quadric objects
    can = gluNewQuadric(); //declared as quadric objects
```

Quadric rendering properties

- We can defines few rendering properties. If you don't defines these properties, the default value will be used.
  - gluQuadricTexture(quadric, value)   //Default: false
  - gluQuadricDrawStyle(quadric, value)   //Default: GLU_FILL
    - https://learn.microsoft.com/en-us/windows/win32/opengl/gluquadricdrawstyle
  - gluQuadricNormals(quadric, value)   //Default: GLU_SMOOTH
    - https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/gluQuadricNormals.xml
  - gluQuadricOrientation(quadric, value)   //Default: GLU_OUTSIDE
    - https://learn.microsoft.com/en-us/windows/win32/opengl/gluquadricorientation

Rendering quadric shape

- Now we have creates and defines all the properties that we need. We can draw all quadric shapes that we would with the quadric object.
- To draw a sphere :
  - glu.gluSphere(quadric, radius, slices, rings)
- To draw a cylinder (or a cone if a radius is equal to 0) :
  - glu.gluCylinder(quadric, bottomRadius, topRadius, height, slices, rings)
- To draw a CD (or a disk if internalRadius is equal to 0) :
  - glu.gluDisk(quadric, internalRadius, externalRadius, slices, rings)
- To draw a partial CD (or a piece of a disk) :
  - glu.gluPartialDisk(quadric, internalRadius, externalRadius, slices, rings, startAngle, angle)



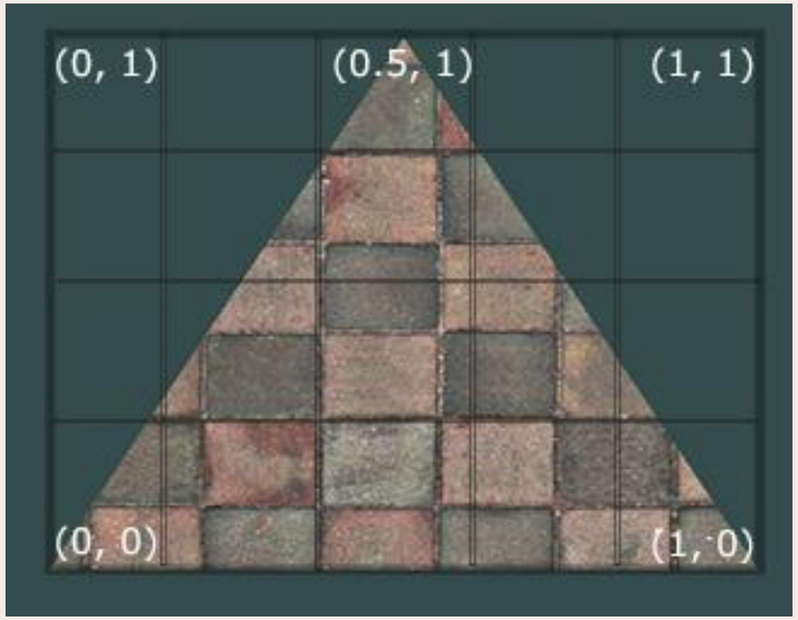
```
glEnable(GL_TEXTURE_2D);

////--------EARTH------------------------------------------------------///
glBindTexture(GL_TEXTURE_2D, _textureBall);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

gluQuadricTexture(ball,1);

glPushMatrix();
glTranslatef(-1.5f, 0.0f, 0.0f);
glRotatef(270,1.0f, 0.0f, 0.0f);
glRotatef(angle,0.0f, 0.0f, 1.0f); //animated rotation
gluSphere(ball,1.25,24,24);
glPopMatrix();
////--------EARTH------------------------------------------------------///


////--------PEPSI CAN----------------------------------------------------///
glBindTexture(GL_TEXTURE_2D, _textureCan);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

gluQuadricTexture(can,1);

glPushMatrix();
glTranslatef(1.5f, -1.0f, 0.0f);
glRotatef(270,1.0f, 0.0f, 0.0f);
glRotatef(angle,0.0f, 0.0f, -1.0f); //animated rotation
gluCylinder(can,0.7,0.7,2,24,24);
glPopMatrix();
////--------PEPSI CAN----------------------------------------------------///
```

Multi-Texturing

Textures

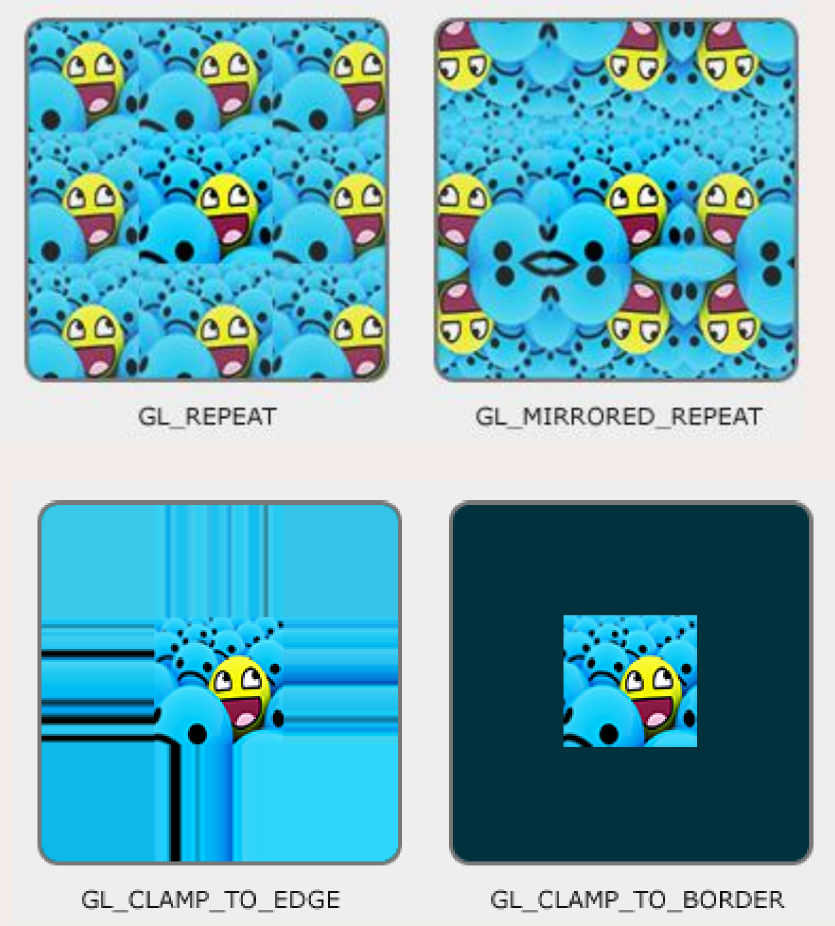
- What artists and programmers generally prefer is to use a texture. A texture is a 2D image (even 1D and 3D textures exist) used to add detail to an object; think of a texture as a piece of paper with a nice brick image (for example) on it neatly folded over your 3D house so it looks like your house has a stone exterior. Because we can insert a lot of detail in a single image, we can give the illusion the object is extremely detailed without having to specify extra vertices.

- Texture coordinates range from 0 to 1 in the x and y axis (remember that we use 2D texture images). Retrieving the texture color using texture coordinates is called sampling. Texture coordinates start at (0,0) for the lower left corner of a texture image to (1,1) for the upper right corner of a texture image. The following image shows how we map texture coordinates to the triangle:



Texture Wrapping

- Texture coordinates usually range from (0,0) to (1,1) but what happens if we specify coordinates outside this range? The default behavior of OpenGL is to repeat the texture images (we basically ignore the integer part of the floating point texture coordinate), but there are more options OpenGL offers:
  - GL_REPEAT: The default behavior for textures. Repeats the texture image.
  - GL_MIRRORED_REPEAT: Same as GL_REPEAT but mirrors the image with each repeat.
  - GL_CLAMP_TO_EDGE: Clamps the coordinates between 0 and 1. The result is that higher coordinates become clamped to the edge, resulting in a stretched edge pattern.
  - GL_CLAMP_TO_BORDER: Coordinates outside the range are now given a user-specified border color.
  - Each of the options have a different visual output when using texture coordinates outside the default range. Let's see what these look like on a sample texture image (original image by Hólger Rezende):

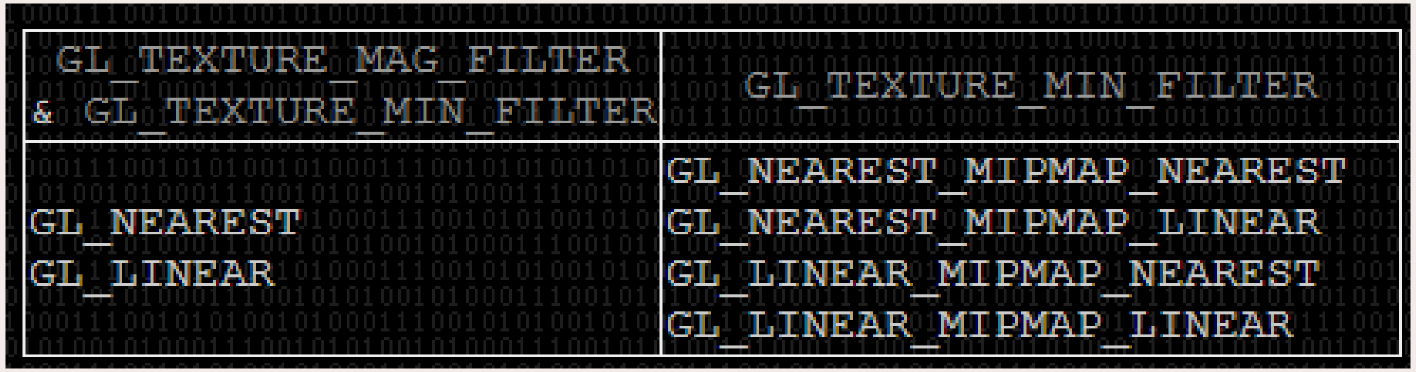


Problems of texture wrapping parameters

- If Clamp to edge & border wont work add this to your code:
  - #define GL_CLAMP_TO_EDGE 0x812F
  - #define GL_CLAMP_TO_BORDER 0x812D

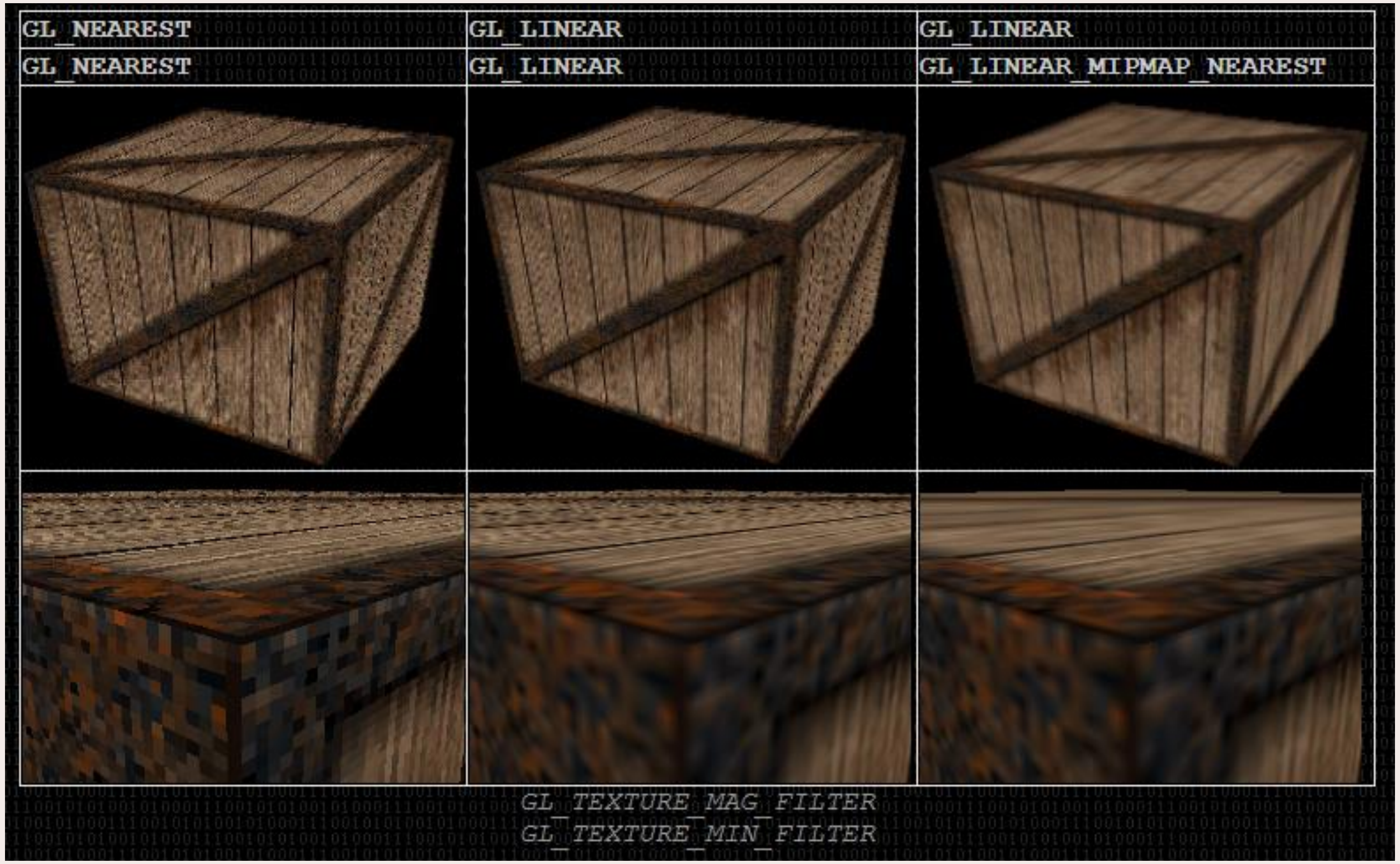
Texture Filtering

- Filters control how the texture is enlarged and shrinked. They are defined with :
- gl.glTexParameteri(target, property, value)
- target is GL_TEXTURE_2D
- property can be GL_TEXTURE_MAG_FILTER (texture magnified, enlarged) or GL_TEXTURE_MIN_FILTER (texture minimized, shrinked).
- value can be one of those constants :



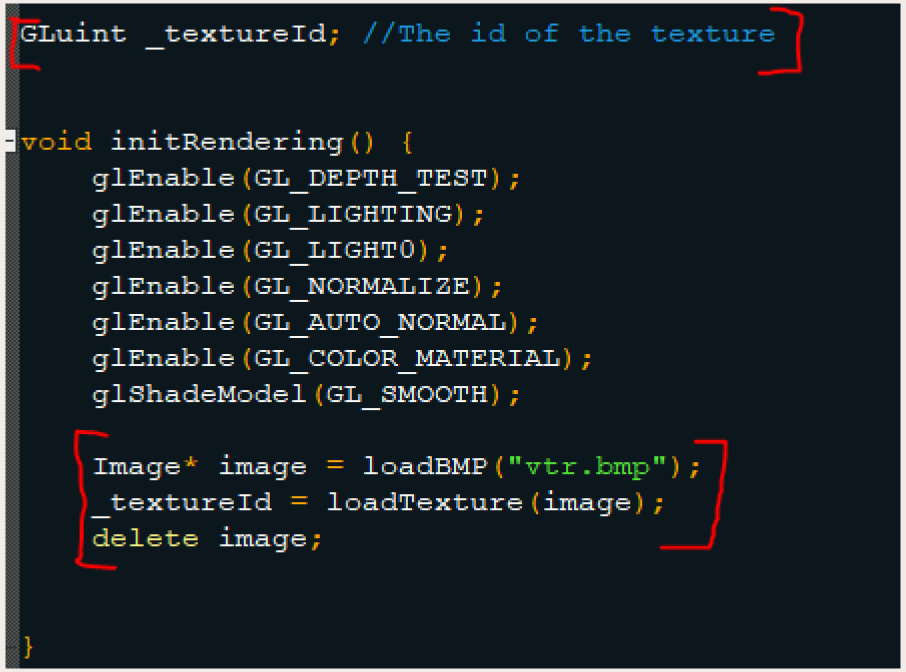
GL_TEXTURE_MAG_FILTER & GL_TEXTURE_MIN_FILTER	GL_TEXTURE_MIN_FILTER
GL_NEAREST GL_LINEAR	GL_NEAREST_MIPMAP_NEAREST GL_NEAREST_MIPMAP_LINEAR GL_LINEAR_MIPMAP_NEAREST GL_LINEAR_MIPMAP_LINEAR

- Here is the result for 3 different magnification/minification filter associations :



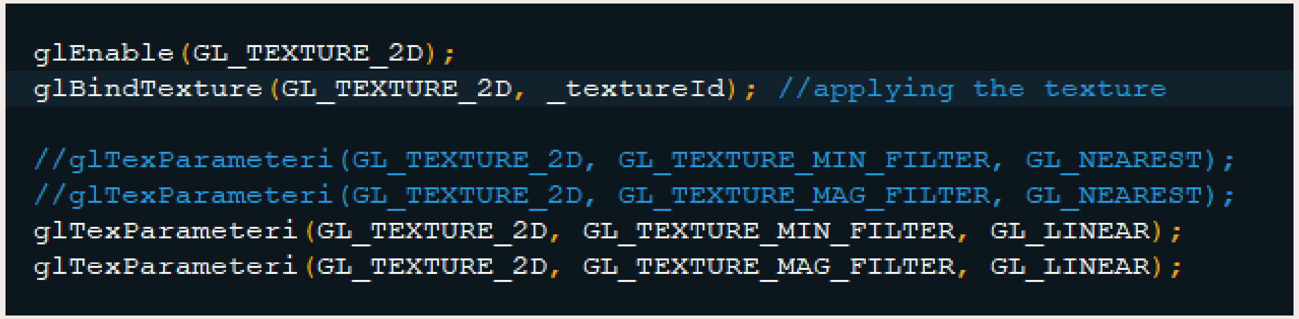
Applying texture to a primitive

1. Declare the texture id and load the image file



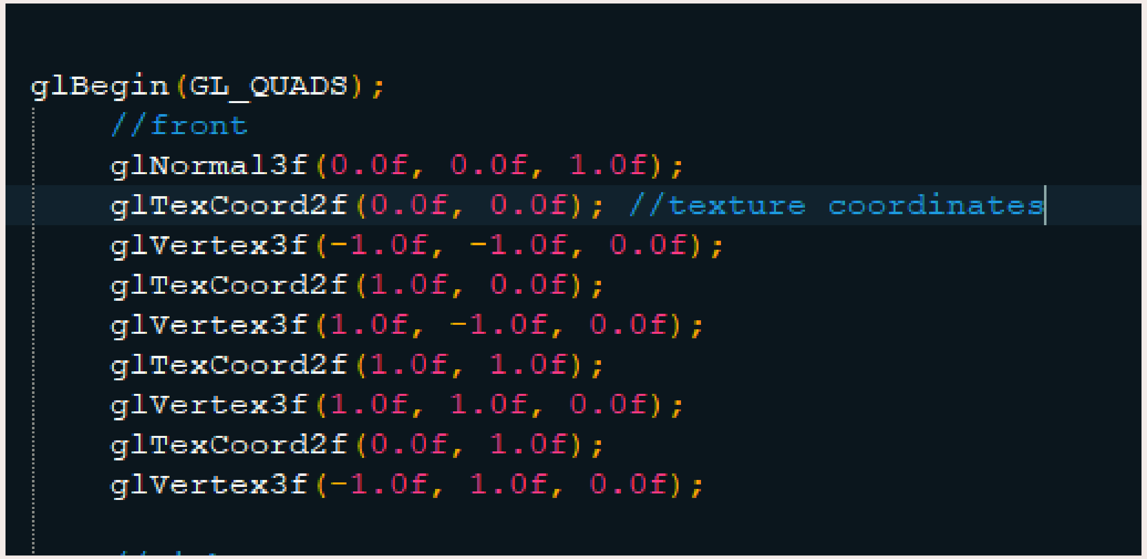
```
GLuint _textureId; //The id of the texture


void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_COLOR_MATERIAL);
    glShadeModel(GL_SMOOTH);

    Image* image = loadBMP("vtr.bmp");
    _textureId = loadTexture(image);
    delete image;


}
```

2. Enable 2D textures, bind the texture and apply texture filtering



```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, _textureId); //applying the texture

//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

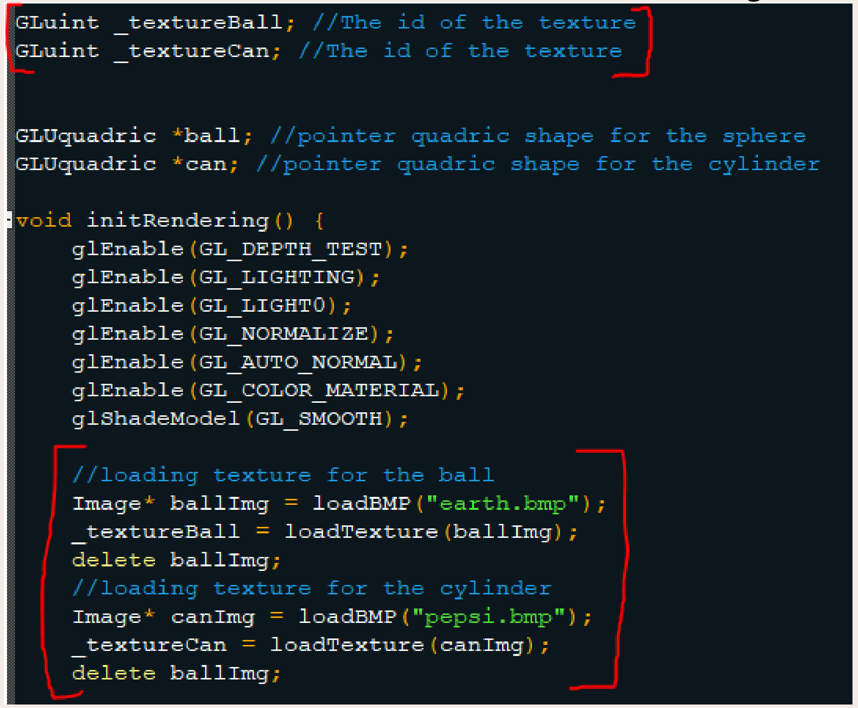
3. Apply Each texture coordinate to each vertex



```
glBegin(GL_QUADS);
    //front
    glNormal3f(0.0f, 0.0f, 1.0f);
    glTexCoord2f(0.0f, 0.0f); //texture coordinates
    glVertex3f(-1.0f, -1.0f, 0.0f);
    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(1.0f, -1.0f, 0.0f);
    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(1.0f, 1.0f, 0.0f);
    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(-1.0f, 1.0f, 0.0f);
```

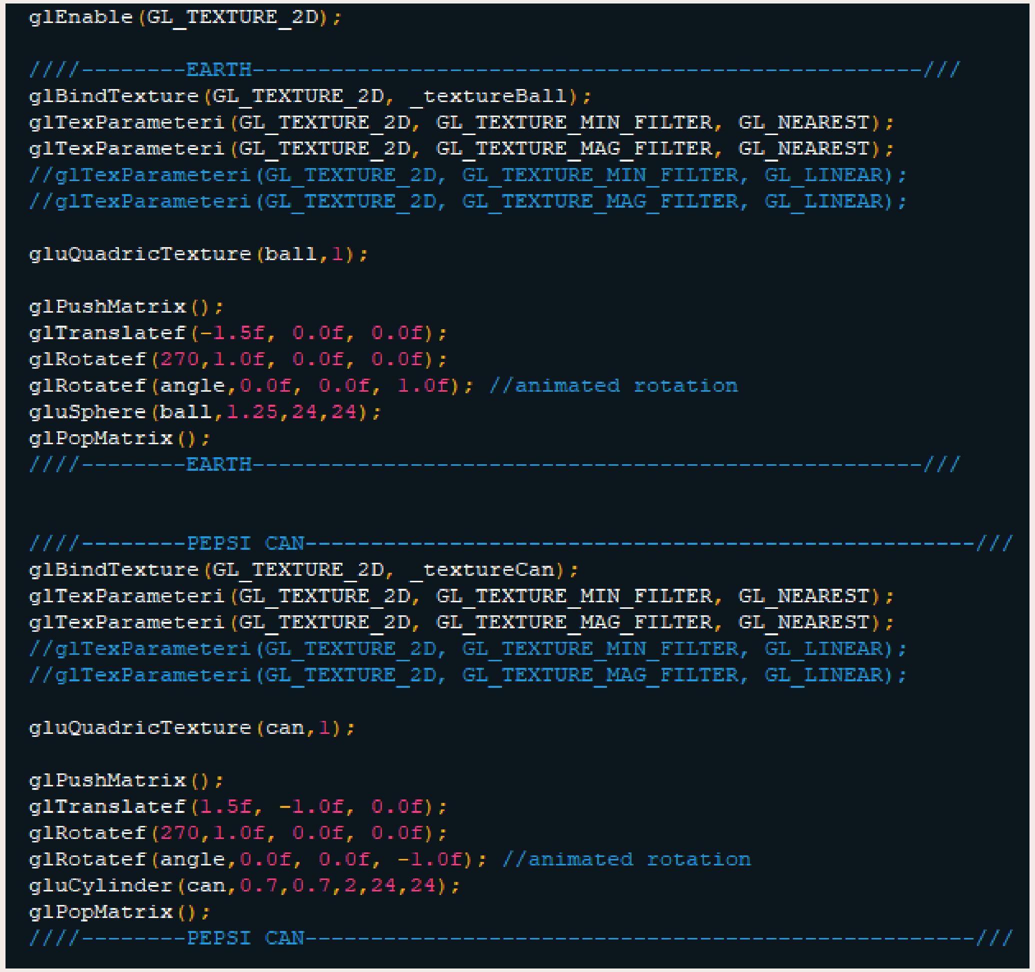
Applying texture to Quadric Shapes

1. Declare the texture id and load the image file



```
GLuint _textureBall; //The id of the texture
GLuint _textureCan; //The id of the texture


GLUquadric *ball; //pointer quadric shape for the sphere
GLUquadric *can; //pointer quadric shape for the cylinder

void initRendering() {
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_NORMALIZE);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_COLOR_MATERIAL);
    glShadeModel(GL_SMOOTH);

    //loading texture for the ball
    Image* ballImg = loadBMP("earth.bmp");
    _textureBall = loadTexture(ballImg);
    delete ballImg;
    //loading texture for the cylinder
    Image* canImg = loadBMP("pepsi.bmp");
    _textureCan = loadTexture(canImg);
    delete ballImg;
```

1. Apply the textures to each quadric shapes



```
glEnable(GL_TEXTURE_2D);

////--------EARTH---------------------------------------------------///
glBindTexture(GL_TEXTURE_2D, _textureBall);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

gluQuadricTexture(ball,1);

glPushMatrix();
glTranslatef(-1.5f, 0.0f, 0.0f);
glRotatef(270,1.0f, 0.0f, 0.0f);
glRotatef(angle,0.0f, 0.0f, 1.0f); //animated rotation
gluSphere(ball,1.25,24,24);
glPopMatrix();
////--------EARTH---------------------------------------------------///


////--------PEPSI CAN-------------------------------------------------///
glBindTexture(GL_TEXTURE_2D, _textureCan);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
//glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

gluQuadricTexture(can,1);

glPushMatrix();
glTranslatef(1.5f, -1.0f, 0.0f);
glRotatef(270,1.0f, 0.0f, 0.0f);
glRotatef(angle,0.0f, 0.0f, -1.0f); //animated rotation
gluCylinder(can,0.7,0.7,2,24,24);
glPopMatrix();
////--------PEPSI CAN-------------------------------------------------///
```

More Learning Links

- Textures - https://learnopengl.com/Getting-started/Textures
- Quadratic Shapes - http://jerome.jouvie.free.fr/opengl-tutorials/Tutorial7.php