**Rendering to a Texture: A Good Way to Get a Texture for Use in a Shader Second Render Pass**

---

**Background = “Binding” to the Context**

When you want to use that same Texture Object, just bind it again. All of the characteristics will then be active, just as if you had specified them again.

---

**Code for the Render-to-Texture Process**

In InitGraphics(), generate a Framebuffer object, a ColorBuffer handle, and a DepthBuffer handle:

```gl
GLuint Framebuffer;
GLuint ColorBuffer;
GLuint DepthBuffer;
```

Bind the off-screen framebuffer to be the current output display:

```gl
glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, ColorBuffer, 0);
```

Setup the size you want the texture rendering to be (this can be larger than the display window or even the display monitor):

```gl
int sizeS = 2048;
int sizeT = 2048;
```

---

**The Overall Render-to-Texture Process**

1. Generate a handle for a Framebuffer Object.
2. Generate handles for one Color Texture Object and for one Depth Texture Object.
3. Bind the Color Texture Object to the Context (glBindTexture). Act as if you are downloading texels to it but set that array to NULL. Just give the texture size. Assign texture parameters to it. Attach it to the Framebuffer Object (glFramebufferTexture2D) as the GL_COLOR_ATTACHMENT.
4. Bind the Depth Texture Object to the Context (glBindTexture). Act as if you are downloading texels to it but set that array to NULL. Just give the texture size. Assign texture parameters to it. Attach it to the Framebuffer Object (glFramebufferTexture2D) as the GL_DEPTH_ATTACHMENT.
5. Bind the Framebuffer Object to the Context (glBindFramebuffer), thus re-binding the display monitor.
6. Render as normal. Be sure the size of the viewport matches the size of the textures you created.
7. Unbind the Framebuffer Object from the Context. (glBindFramebuffer(0),) thus un-binding the display monitor.
Code for the Render-to-Texture Process

Now, render as you normally would. Be sure to set the viewport to match the size of the color and depth buffers:
```c
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glEnable( GL_DEPTH_TEST );
glShadeModel( GL_SMOOTH );
glViewport( 0, 0, sizeS, sizeT );
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
gluPerspective( 90., 1., 0.1, 1000. );
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
gluLookAt( 0., 0., 3., 0., 0., 0., 0., 1., 0. );
glRotatef( RotY, 0., 1., 0. );
glRotatef( RotX, 1., 0., 0. );
glScalef( Scale, Scale, Scale );
glColor3f( 1., 1., 1. );
glutWireTetrahedron( 1. );
```

Tell OpenGL to go back to rendering to the display monitor:
```c
glBindFramebuffer( GL_FRAMEBUFFER, 0 );
```

Tell OpenGL to render the scene as normal, mapping the Texture onto a quadrilateral:
```
clear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
```
```
Texture2D 6 1024 1024
```
```
RenderToTexture 6
```
```
Background 0.0.0.
```
```
Clear
```
```
LookAt 0.0.3 0.0.0 0.1.0.
```
```
Vertex cube.vrt
```
```
Fragment cube.frag
```
```
Program Cube
```
```
uAT = 0.25; uBT = 0.25;
```
```
uAL = 0.1; uBL = 0.1;
uAR = 0.1; uBR = 0.1;
```
```
Teapot
```
```
RenderToTexture 6
```
```
Background 0.0.0.
```
```
Clear
```
```
Vertex image.vrt
```
```
Fragment image.frag
```
```
Program Filter
```
```
medf = ( uAL + uAR ) / 2.0;
```
```
medb = ( uBL + uBR ) / 2.0;
```
```
Topaz
```
```
RenderToTexture 6
```
```
Background 0.0.0.
```
```
Clear
```
```
Vertex texture.vrt
```
```
Fragment texture.frag
```
```
Program Texture
```
```
glColor3f( 1.0, 1.0, 1.0 );
```
```
Multipass Algorithm to Render and then Image Process

Original  Sharpened  Edge Detected

No Noise

Noise

Multipass Algorithm to Implement Conway's Game of Life

Ping-pong between two different textures. One texture is being read from (the previous state) and the other is being written into (the next state).
OpenGL GLIB

## Perspective 70

# setup the 2 textures:

```glsl
Texture2D 5  paint0.bmp
Texture2D 6  512 512
```

# execute the first iteration:

```glsl
RenderToTexture 6
Background 0. 0. 0.
Clear
Vertex        life.vert
Fragment    life.frag
Program   GameOfLife1    uTexture 5
TextureMatrix
Translate 0. 0. -3.08
QuadXY .2 2.
```

# render it so we can see it:

```glsl
RenderToTexture
Background 0. .2 0.
Clear
Vertex         texture.vert
Fragment    texture.frag
Program Texture1    uTexture 6
ModelViewMatrix
Translate 0. 0. -3.08
QuadXY .2 2.
SwapBuffers
```

# execute the second iteration:

```glsl
RenderToTexture 5
Background 0. 0. 0.
Clear
Vertex         life.vert
Fragment    life.frag
Program GameOfLife2   uTexture 6
QuadXY .2 2.
```

# render it so we can see it:

```glsl
RenderToTexture
Background .2 0. 0.
Clear
Vertex         texture.vert
Fragment    texture.frag
Program Texture6     uTexture 5
QuadXY .2 2.
```

# repeat:

```glsl
animate
```

---

### life.glib

```glsl
int isize = textureSize(uTexture, 0);
vec2 st = vST;
ivec2 is = ivec2( st.s*float(isize.s-1), st.t*float(isize.t-1) ); // 0 -> dimension-1
ivec2 is0p = ivec2( 1, 0 );
ivec2 is0p = ivec2( 0, 1 );
ivec2 ispp = ivec2( 1, 1 );
ivec2 ispm = ivec2( 1, -1 );
vec3 i00 = texelFetch(uTexture, is, 0).rgb; // index using integer indices
vec3 im10 = texelFetch(uTexture, is-is0p, 0).rgb;
vec3 i0m1 = texelFetch(uTexture, is-is0p, 0).rgb;
vec3 ip10 = texelFetch(uTexture, is+is0p, 0).rgb;
vec3 i0p1 = texelFetch(uTexture, is+is0p, 0).rgb;
vec3 im1m1 = texelFetch(uTexture, is-ispp, 0).rgb;
vec3 ip1p1 = texelFetch(uTexture, is+ispp, 0).rgb;
vec3 im1p1 = texelFetch(uTexture, is-ispm, 0).rgb;
vec3 ip1m1 = texelFetch(uTexture, is+ispm, 0).rgb;

int sum = 0;
if( im10.b > TB  &&  im10.r < TR ) sum++;
if( i0m1.b > TB  &&  i0m1.r < TR ) sum++;
if( ip10.b > TB  &&  ip10.r < TR ) sum++;
if( i0p1.b > TB  &&  i0p1.r < TR ) sum++;
if( im1m1.b > TB  &&  im1m1.r < TR ) sum++;
if( ip1p1.b > TB  &&  ip1p1.r < TR ) sum++;
if( im1p1.b > TB  &&  im1p1.r < TR ) sum++;
if( ip1m1.b > TB  &&  ip1m1.r < TR ) sum++;

vec3 newcolor = i00;
if( sum == T3 )
{
    newcolor = ALIVE;
}
else if( sum <= T1  ||  sum >= T4 )
{
    newcolor = DEAD;
}
gl_FragColor = vec4( newcolor, 1. );
```

Tell OpenGL to go back to rendering to the display monitor:

```glsl
glBindFramebuffer( GL_FRAMEBUFFER, 0 );
```

Read the pixels back and do something with them (such as writing an image file):

```glsl
unsigned char *image = new unsigned char [ 3*sizeS*sizeT ];
glPixelStorei(GL_PACK_ALIGNMENT, 1);
glReadPixels(0, 0, sizeS, sizeT, GL_RGB, GL_UNSIGNED_BYTE, image);
```

Tell OpenGL to go back to rendering to the display monitor:

```glsl
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

---

**Code for Reading Pixels Back**

Read the pixels back and do something with them (such as writing an image file):

```glsl
unsigned char *image = new unsigned char [ 3*sizeS*sizeT ];
```

Tell OpenGL to go back to rendering to the display monitor:

```glsl
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```

---

**Hardcopy and Display**

1. Create Hardcopy File
2. Display Hardcopy File

Render-to-Framebuffer is great for creating arbitrary-resolution hardcopy.

---

**Tell OpenGL to go back to rendering to the display monitor:**

```glsl
glBindFramebuffer(GL_FRAMEBUFFER, 0);
```