Using Fragment Shaders to Manipulate Images

Mike Bailey
mjb@cs.oregonstate.edu

The Basic Idea: Wrap an Image Around a Piece of Geometry

In software, this is a very slow process. In hardware, this is very fast. The development of texture-mapping hardware was one of the most significant events in the history of computer graphics. This is really what finally enabled game development on a realistic scale.

The Basic Ideas

The mapping between the geometry of the 3D object and the S and T of the texture image works like this:

```
(199.68, 171.52) in texels
```

Thus, you do not need to be aware of the texture’s resolution when you are specifying coordinates that point into it. Think of S and T as a measure of what fraction of the way you are into the texture.

Using a Texture: Assigning an (s,t) to each vertex

Enable texture mapping:
```
gEnable( GL_TEXTURE_2D );
```

Draw your polygons, specifying s and t at each vertex:
```
gBegin( GL_TRIANGLES );
gTexCoord2f( s0, t0 );
gNormal3f( nx0, ny0, nz0 );
gVertex3f( x0, y0, z0 );
gTexCoord2f( s1, t1 );
gNormal3f( nx1, ny1, nz1 );
gVertex3f( x1, y1, z1 );
...
gEnd( );
```

(If this geometry is static, i.e., will never change, it is a good idea to put this all into a display list.)

Disable texture mapping:
```
gDisable( GL_TEXTURE_2D );
```

To prevent confusion, the texture image pixels are not called pixels. A pixel is an RGB dot in the final screen image. An RGB dot in the texture image is called a texture element, or texel.

Similarly, to avoid terminology confusion, a texture image’s width and height dimensions are not called X and Y. They are called S and T.

A texture image is not indexed by its actual resolution coordinates. Instead, it is indexed by a coordinate system that is resolution-independent. The left side is always S=0, the right side is S=1, the bottom is T=0, and the top is T=1.

Thus, you do not need to be aware of the texture’s resolution when you are specifying coordinates that point into it. Think of S and T as a measure of what fraction of the way you are into the texture.

Similarly, texture image pixels are not called pixels. A pixel is an RGB dot in the final screen image. An RGB dot in the texture image is called a texture element, or texel.

Similarly, to avoid terminology confusion, a texture image’s width and height dimensions are not called X and Y. They are called S and T.

A texture image is not indexed by its actual resolution coordinates. Instead, it is indexed by a coordinate system that is resolution-independent. The left side is always S=0, the right side is S=1, the bottom is T=0, and the top is T=1.

Thus, you do not need to be aware of the texture’s resolution when you are specifying coordinates that point into it. Think of S and T as a measure of what fraction of the way you are into the texture.
Texture Image Basics in Shaders

Index the image using the usual texture indexing:

\[ (0 \leq s, t \leq 1) \]

When you get back an RGB from the texture, remember that, if the texture's numbers are colors:

\[ (0 \leq r, g, b \leq 1) \]

If the texture contains data, then the numbers can be anything.

Also, if you need to know the texel resolution of this texture, do this:

\[
\text{ivec2 ires = textureSize( uImageUnit, 0 );}
\]
\[
\text{float ResS = float( ires.s );}
\]
\[
\text{float ResT = float( ires.t );}
\]

Thus, to get from the current texel's \((s, t)\) to a neighboring texel's \((s, t)\), add:

\[
\pm (1./\text{ResS}, 1./\text{ResT})
\]

---

A Good Example of Manipulating RGB Numbers – the Image Negative

Image RGB values are just numbers – they can be manipulated any way you'd like!

---

Image Negative

---

Image Negative

---

Image Negative

---

Image Negative

---

Image Distortion

---

Image Distortion

---
Image Un-masking:  
Interpolation can still happen when $t < 0.$ or $t > 1.$  
\[ Q = (1 - t)Q_0 + tQ_1 \]
\[ R_{\text{out}} = \text{mix}(R_{\text{out, want}}, R_{\text{in}}, t) \]

$\begin{align*}
0.0 & \quad t = 0. & \quad \text{All dino, no sphere} \\
1.0 & \quad t = 1. & \quad \text{All sphere, no dino} \\
2.0 & \quad t = 2. & \quad \text{More sphere, negative dino}
\end{align*}$

Brightness  
\[ l_{\text{out}} = \text{vec3}(0, 0, 0); \]

Contrast  
\[ l_{\text{out}} = \text{vec3}(0.5, 0.5, 0.5); \]

HDTV Luminance Standard  
\[ \text{Luminance} = 0.2125^*\text{Red} + 0.7154^*\text{Green} + 0.0721^*\text{Blue} \]
Difference

\[ Q = (1 - t)Q_0 + tQ_1 \]

\[ R_{\text{blend}} = \text{mix}(R_{\text{destination}}, R_{\text{blend}}, t) \]

ChromaKey

Replace the fragment if:
- \( R < t \)
- \( G < t \)
- \( B > 1 - t \)

Blue/Green Screen Usage is ChromaKey

Loyal Studios

https://www.youtube.com/watch?v=Ldh6FKavxeI

Blur Convolution:

\[
\begin{bmatrix}
1 & 2 & 1 \\
4 & 8 & 4 \\
4 & 8 & 4 \\
1 & 2 & 1 \\
\end{bmatrix}
\]

Sharpening

Blur Convolution:

\[
B = \frac{1}{16} \begin{bmatrix}
1 & 2 & 4 & 2 & 1 \\
2 & 4 & 8 & 4 & 2 \\
4 & 8 & 16 & 8 & 4 \\
2 & 4 & 8 & 4 & 2 \\
1 & 2 & 4 & 2 & 1 \\
\end{bmatrix}
\]

\[
Q = (1 - t)Q_0 + tQ_1
\]

\[
R_{\text{final}} = \text{mix}(R_{\text{destination}} \cdot R_{\text{blend}}, t)
\]
Sharpening

\[ t = 0, \quad t = 1, \quad t = 2. \]

Embossing

\[
\begin{align*}
&\text{vec2 sstp} = \text{vec2}(1./\text{ResS}, 0.); \\
&\text{vec2 sstp} = \text{vec2}(1./\text{ResS}, 1./\text{ResT}); \\
&\text{vec3 cb} = \text{texture}(\text{uImageUnit}, \text{vST}); \\
&\text{vec3 cbstp} = \text{texture}(\text{uImageUnit}, \text{vST} + \text{stpp}); \\
&\text{vec3 dths} = \text{cb} - \text{cbstp}; \\
&\text{float max} = \text{dths}.r; \\
&\text{if} (\text{abs(dths.g)} > \text{abs(max)}) \\
&\text{max} = \text{dths.g}; \\
&\text{if} (\text{abs(dths.b)} > \text{abs(max)}) \\
&\text{max} = \text{dths.b}; \\
&\text{float gray} = \text{clamp}(\text{max} + .5, 0., 1.); \\
&\text{vec4 grayVersion} = \text{vec4}(\text{gray}, \text{gray}, \text{gray}, 1.); \\
&\text{vec4 colorVersion} = \text{vec4}(\text{gray}\cdot\text{c00}, 1.); \\
&\text{gl_FragColor} = \text{mix}(\text{grayVersion}, \text{colorVersion}, t);
\end{align*}
\]

Edge Detection

Horizontal and Vertical Sobel Convolutions:

\[
H = \begin{bmatrix}
-1 & -2 & -1 \\
0 & 0 & 0 \\
1 & 2 & 1 
\end{bmatrix} \quad V = \begin{bmatrix}
-1 & 0 & 1 \\
-2 & 0 & 2 \\
-1 & 0 & 1 
\end{bmatrix}
\]

\[
S = \sqrt{H^2 + V^2} \quad \Theta = \text{atan2}(V, H)
\]

Edge Detection

Toon Rendering

Hand-drawn cartoons have a unique style, typically characterized by:

1. Dark outlines between the important elements in the scene.
2. A reduced collection of available colors (i.e., no smooth shading).

http://drawdoo.com/draw/draw-winnie-the-pooh/
float mag = sqrt( h*h + v*v );
if( mag > uMagTol )
{
    gl_FragColor= vec4( 0., 0., 0., 1. );
}
else
{
    rgb.rgb *= uQuantize; // scale up
    rgb.rgb += vec3( .5, .5, .5 ); // round
    ivec3 irgb = ivec3( rgb.rgb ); // cast to all integers
    rgb.rgb = vec3( irgb ); // cast back to floats
    rgb /= uQuantize; // scale down
    gl_FragColor= vec4( rgb, 1. );
}

Quantizing example using the number 3.14159:

<table>
<thead>
<tr>
<th>uQuantize</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.</td>
<td>3.1</td>
</tr>
<tr>
<td>100.</td>
<td>3.14</td>
</tr>
<tr>
<td>1000.</td>
<td>3.141</td>
</tr>
</tbody>
</table>

These are just examples – uQuantize does not need to be a power of 10!
Julia Set

\[ Z_{i+1} = Z_i^2 + C \]

How fast does it converge, if ever?

Doing the Mandelbrot Math in Single vs. Double Precision

32-bit single precision floating point

64-bit double precision floating point

We Can Do Image Processing on Dynamic Scenes with a Two-pass Approach

Pass #1

Lighting Shader

\[ \text{Render a 3D dynamic scene} \]

Texture

\[ \text{Render a quadrilateral} \]

Pass #2

Framebuffer

Sharpen Image Shader

Original

Sharpened