# Using Fragment Shaders to Manipulate Images

**Mike Bailey**

mjb@cs.oregonstate.edu

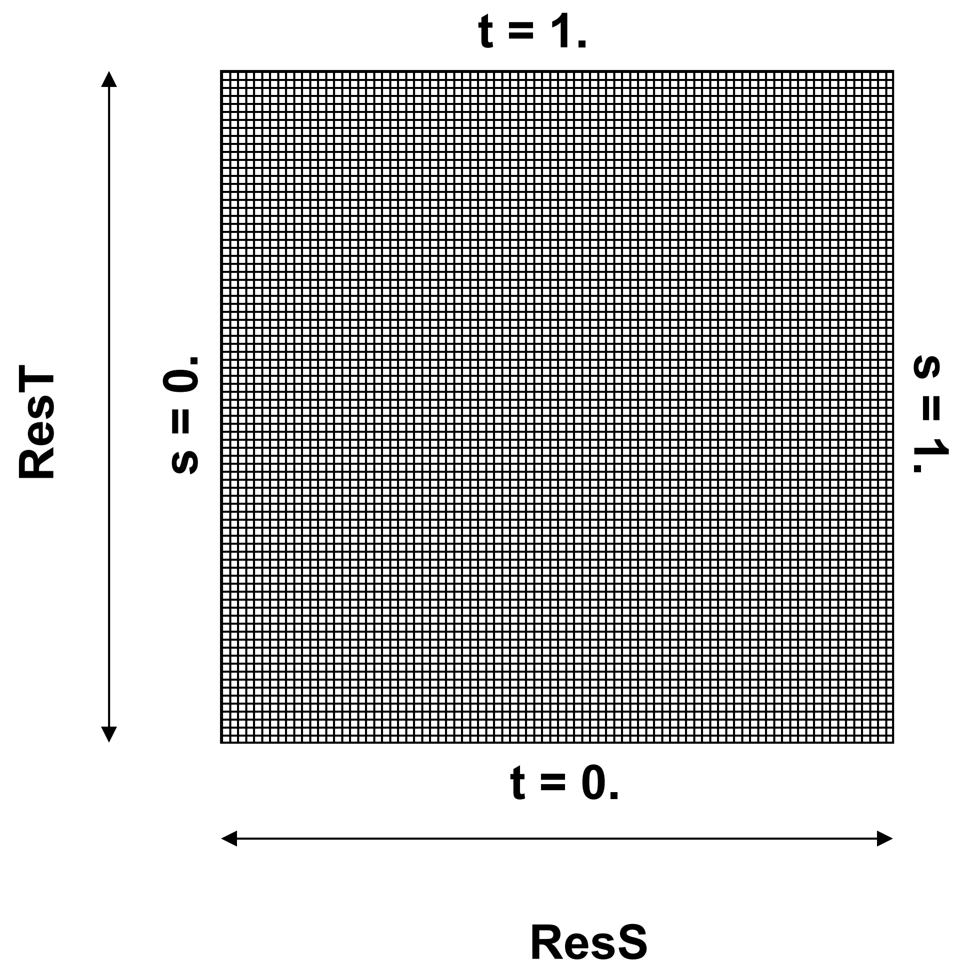
**Oregon State University**

# Image Basics

**Treat the image as a texture. Index it using usual texture indexing ($0. \le s,t \le 1.$)**

**If you need it, the resolution of this texture can be found by saying:**

```
ivec2 ires = textureSize( ImageUnit, 0 );
float ResS = float( ires.s );
float ResT = float( ires.t );
```

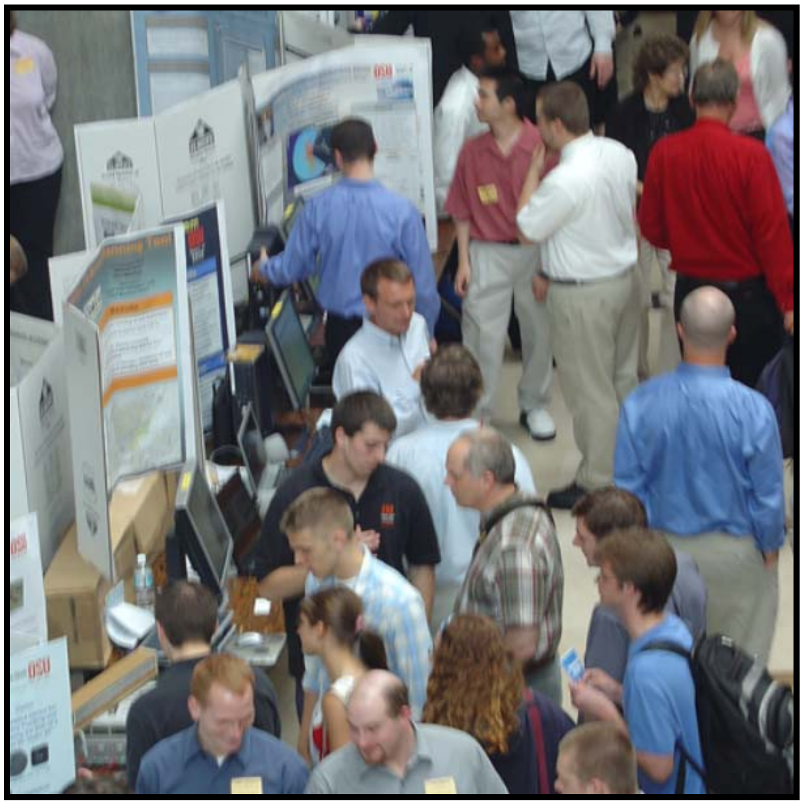



**To get from the current texel to a neighboring texel, add**

$$\pm\ (1./ResS\ ,\ 1./ResT)$$

**to the current (S,T)**

# Image Negative



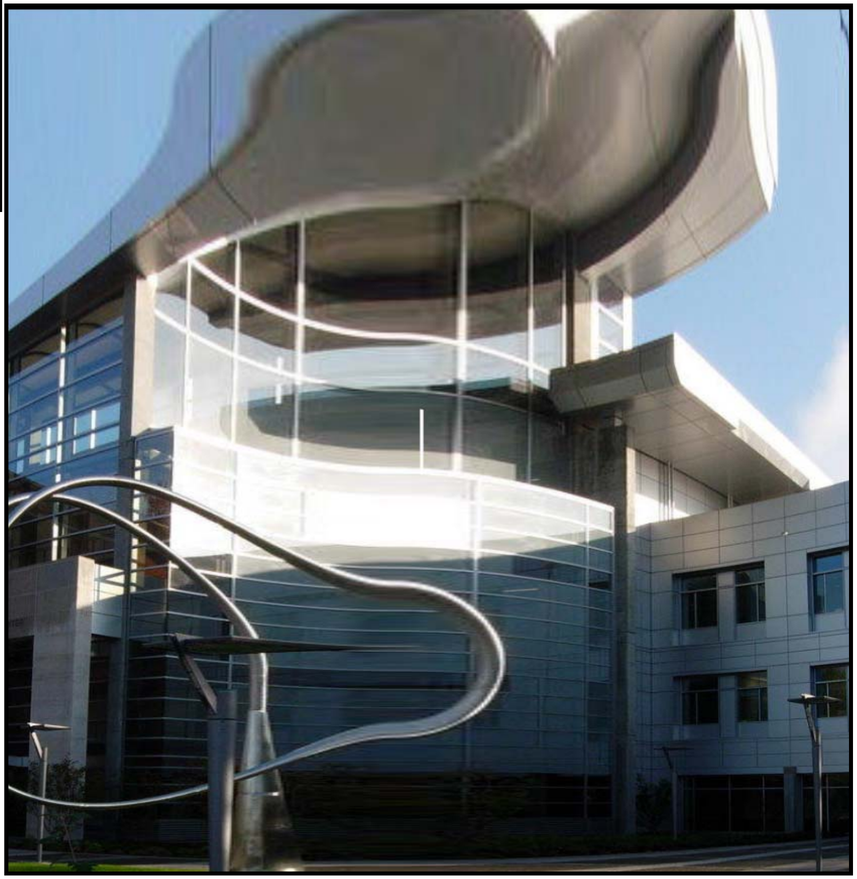
**( R, G, B )**

**( 1.-R, 1.-G, 1.-B )**

# Image Distortion

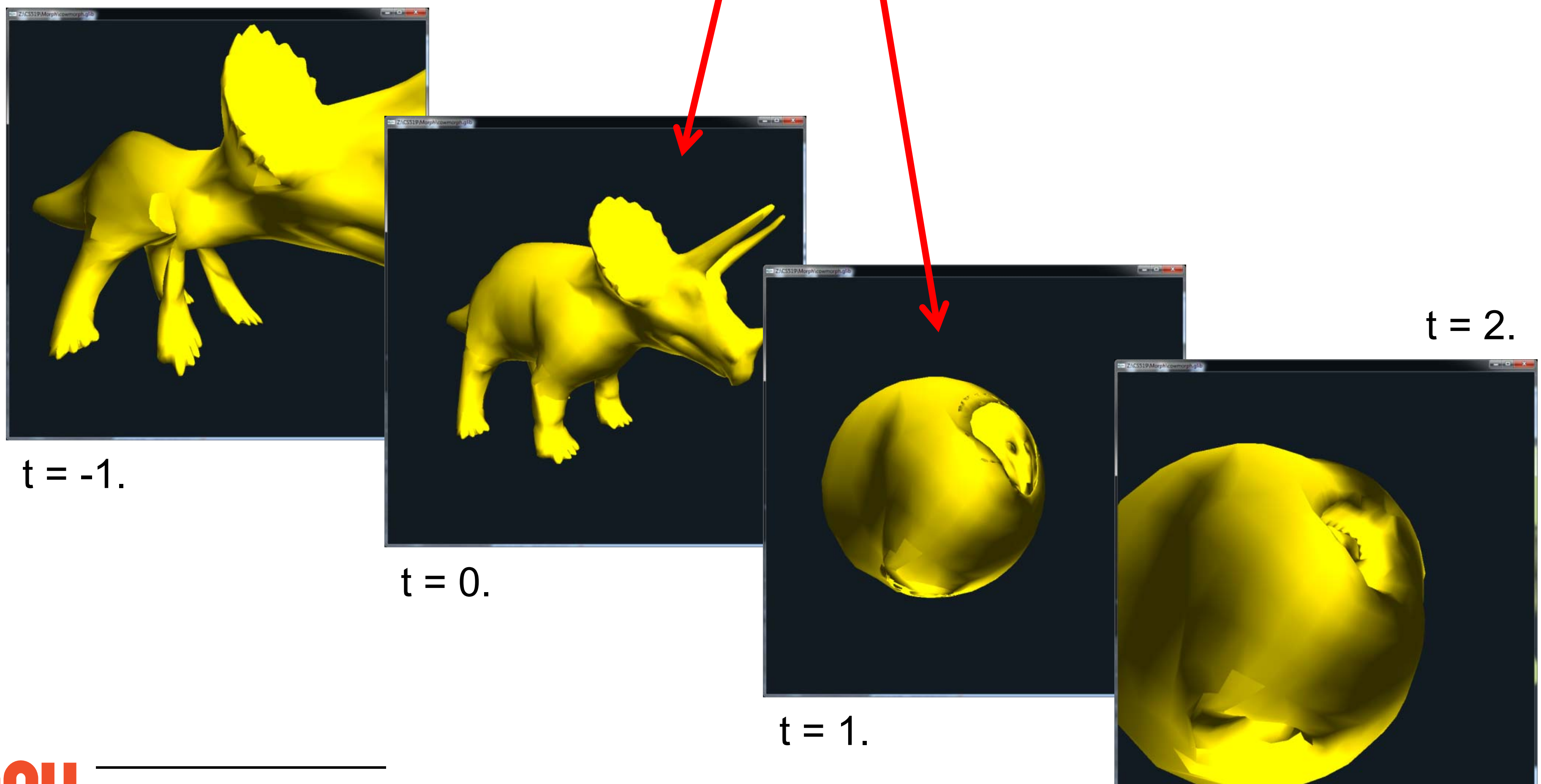
```
uniform float          uS0, uT0;
uniform float          uPower;
uniform sampler2D      uTexUnit;
in vec2                vST;

void
main( )
{
   vec2 delta = vST - vec2(uS0,uT0);
   st = vec2(uS0,uT0) + sign(delta) * pow( abs(delta), uPower );
   vec3 rgb = texture2D( uTexUnit, vST ).rgb;
   gl_FragColor= vec4( rgb, 1. );
}
```

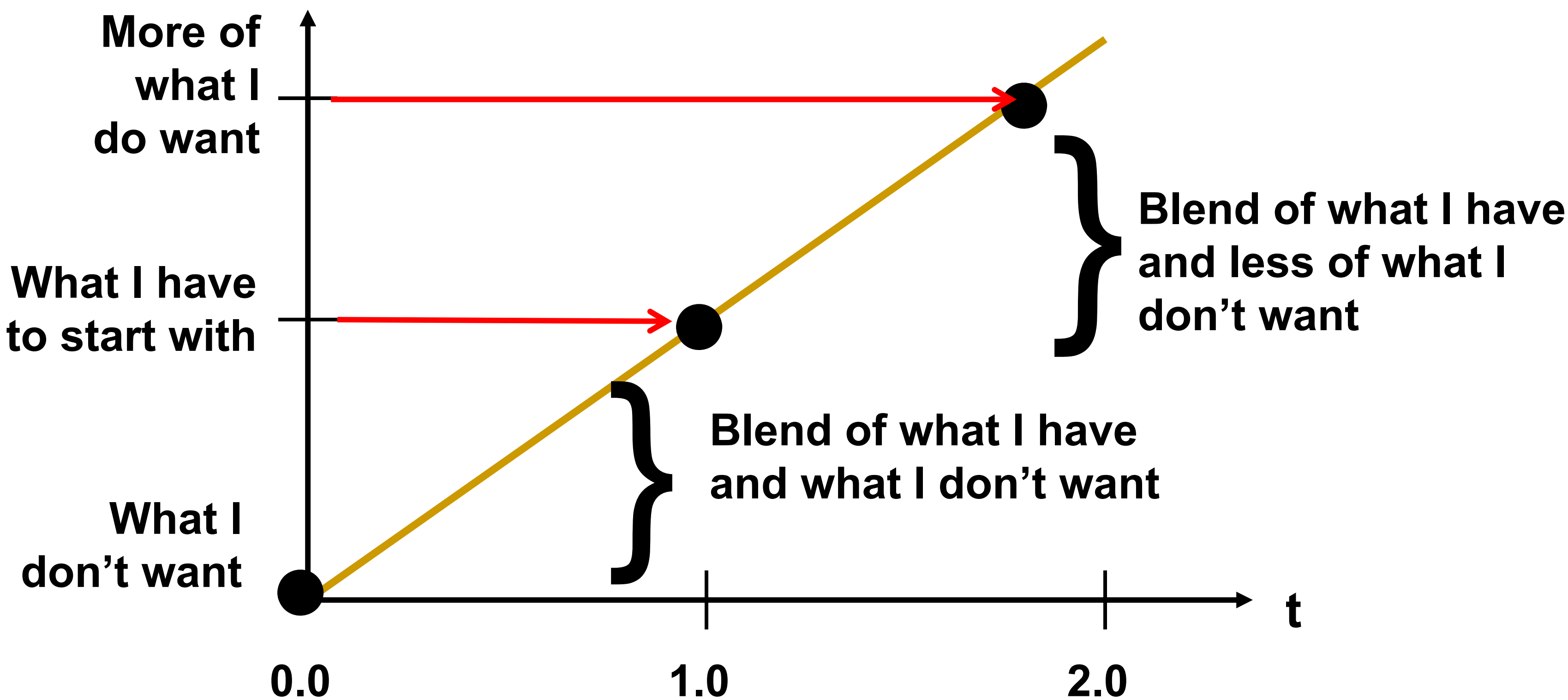


# Image Un-masking:
# Interpolation can still happen when t < 0. or t > 1.

$$Q = (1-t)Q_0 + tQ_1$$



# Image Un-Masking:
# Abusing the Linear Blending Equation for a Good Purpose

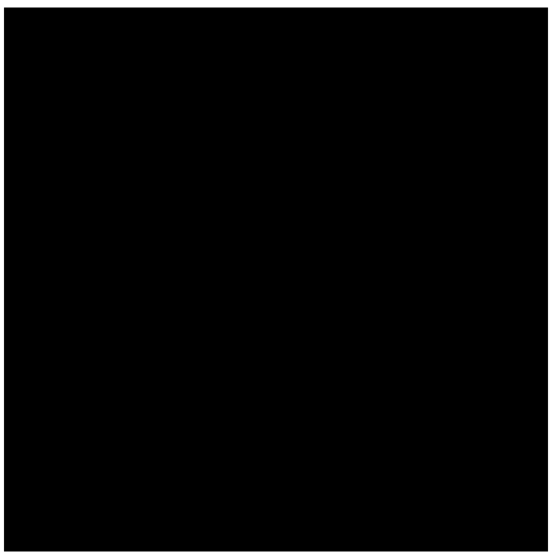



$$Q = (1-t)Q_0 + tQ_1$$

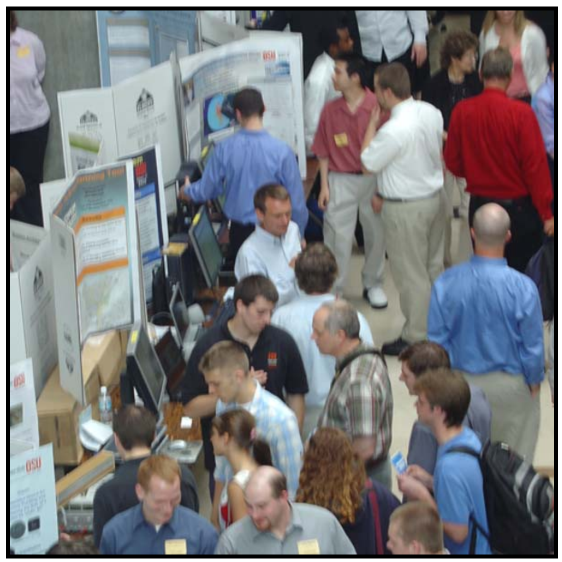
$$I_{out} = (1 - t)*I_{dontwant} + t*I_{in}$$

# Brightness

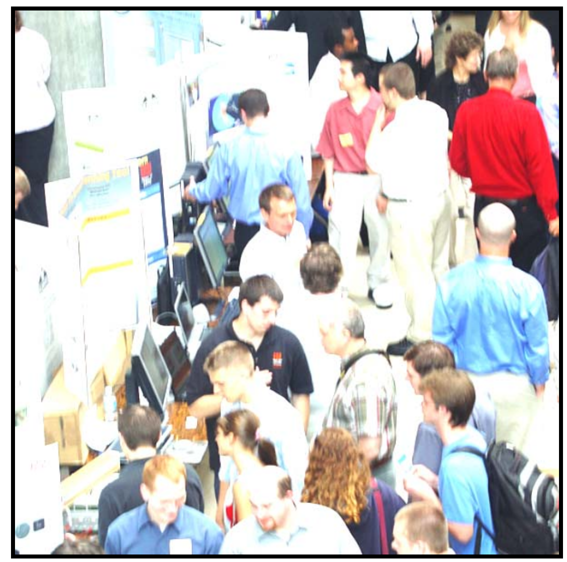
$$I_{dontwant} = vec3(\ 0.,\ 0.,\ 0.\ );$$



T = 0.



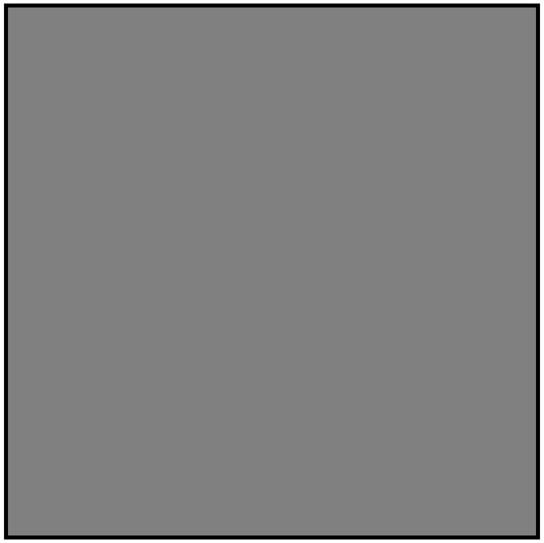
T = 1.



T = 2.

# Contrast

$$I_{dontwant} = \text{vec3}(\ 0.5,\ 0.5,\ 0.5\ );$$



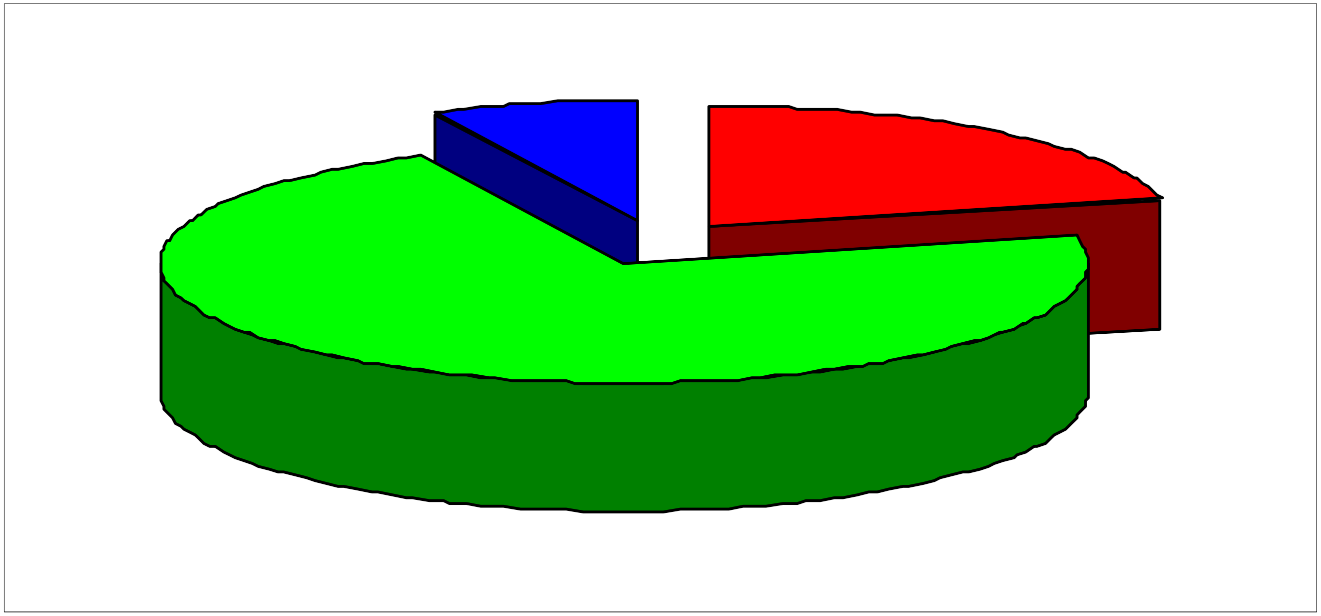
T = 0.

T = 1.

T = 2.

# HDTV Luminance Standard

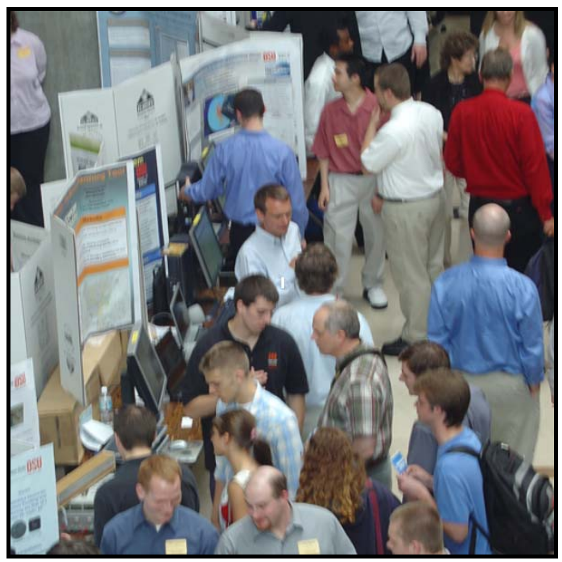
**Luminance = 0.2125*Red + 0.7154*Green + 0.0721*Blue**



# Saturation

$I_{dontwant}$ = vec3( luminance, luminance, luminance );

T = 0.



T = 1.

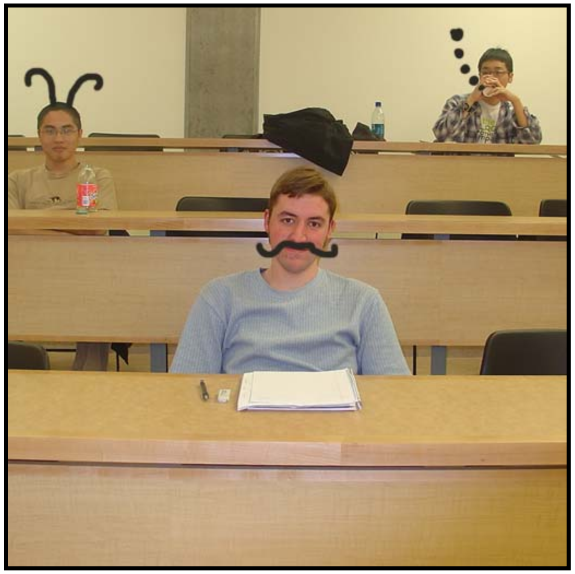
T = 3.

# Difference

$$I_{dontwant} = I_{before}$$

$$I_{in} = I_{after}$$

T = 0.



T = 1.

T = 2.

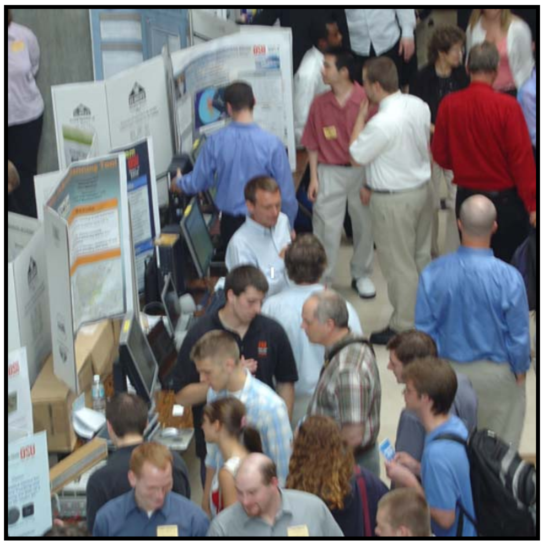
# ChromaKey

Replace fragment if:

R < T

G < T

B > 1.-T



T = 0.

T = 0.5

T = 1.

# Blur

## Blur Convolution:

$$B = \frac{1.}{16.}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

# Sharpening

**Blur Convolution:**

$$B = \frac{1.}{16.}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$I_{dontwant} = I_{blur}$$

# Sharpening

```
vec2 stp0 = vec2(1./ResS,  0.      );
vec2 st0p = vec2(0.      ,  1./ResT);
vec2 stpp = vec2(1./ResS,  1./ResT);
vec2 stpm = vec2(1./ResS, -1./ResT);
vec3 i00 =   texture2D( uImageUnit, vST ).rgb;
vec3 im1m1 = texture2D( uImageUnit, vST-stpp ).rgb;
vec3 ip1p1 = texture2D( uImageUnit, vST+stpp ).rgb;
vec3 im1p1 = texture2D( uImageUnit, vST-stpm ).rgb;
vec3 ip1m1 = texture2D( uImageUnit, vST+stpm ).rgb;
vec3 im10 =  texture2D( uImageUnit, vST-stp0 ).rgb;
vec3 ip10 =  texture2D( uImageUnit, vST+stp0 ).rgb;
vec3 i0m1 =  texture2D( uImageUnit, vST-st0p ).rgb;
vec3 i0p1 =  texture2D( uImageUnit, vST+st0p ).rgb;
vec3 target = vec3(0.,0.,0.);
target += 1.*(im1m1+ip1m1+ip1p1+im1p1);
target += 2.*(im10+ip10+i0m1+i0p1);
target += 4.*(i00);
target /= 16.;
gl_FragColor= vec4( mix( target, irgb, T ), 1. );
```

# Sharpening

T = 0.

T = 1.

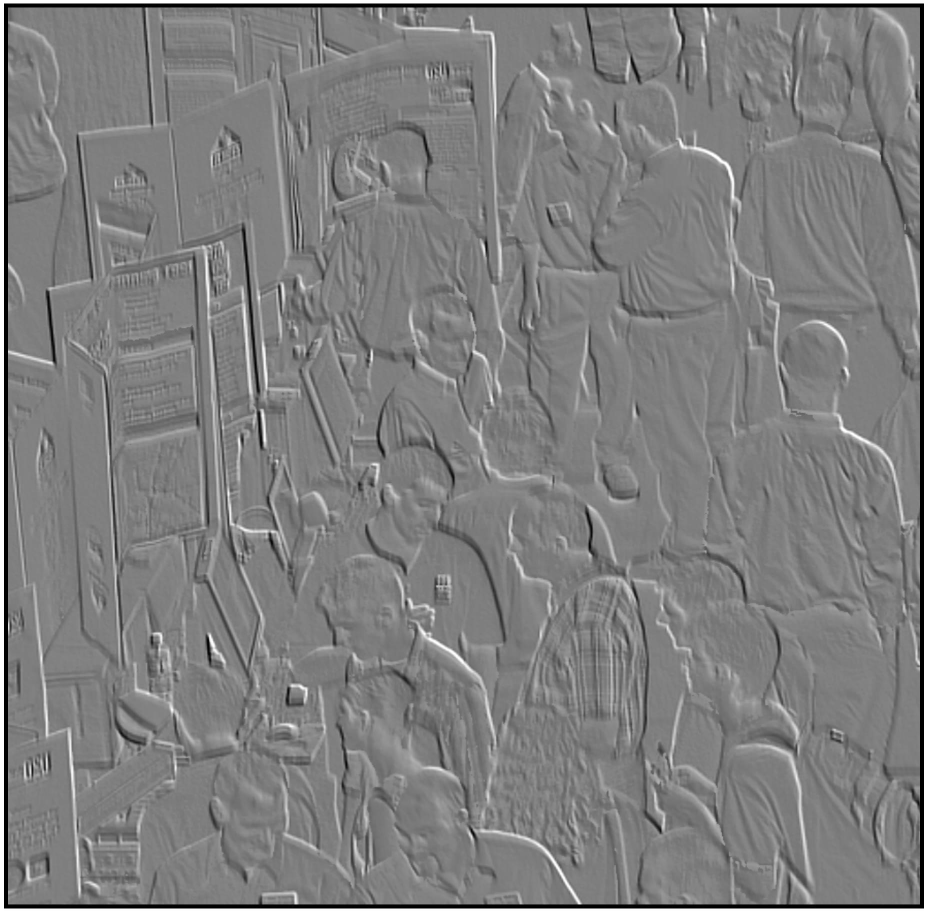
T = 2.

# Embossing

```
vec2 stp0 = vec2( 1./ResS,  0. );
vec2 stpp = vec2( 1./ResS,  1./ResT);
vec3 c00    = texture2D(    uImageUnit, vST ).rgb;
vec3 cp1p1  = texture2D( uImageUnit, vST + stpp ).rgb;

vec3 diffs = c00 - cp1p1;
float max = diffs.r;
if( abs(diffs.g) > abs(max) )
                max = diffs.g;
if( abs(diffs.b) > abs(max) )
                max = diffs.b;

float gray = clamp( max + .5, 0., 1. );
vec4 grayVersion  = vec4( gray, gray, gray, 1. );
vec4 colorVersion = vec4( gray*c00, 1. );
gl_FragColor= mix( grayVersion, colorVersion, T );
```



# Edge Detection

**Horizontal and Vertical Sobel Convolutions:**

$$H = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad V = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$S = \sqrt{H^2 + V^2} \qquad \Theta = \mathrm{atan2}(\ \mathrm{V}, \mathrm{H}\ )$$

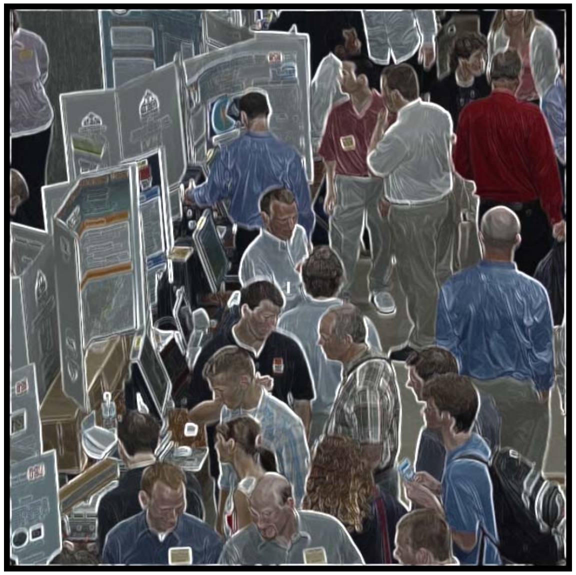
# Edge Detection

```
const vec3 LUMCOEFFS = vec3( 0.2125,0.7154,0.0721 );
. . .
vec2 stp0 = vec2(1./ResS,  0. );
vec2 st0p = vec2(0.     ,  1./ResT);
vec2 stpp = vec2(1./ResS,  1./ResT);
vec2 stpm = vec2(1./ResS, -1./ResT);
float i00 =   dot( texture2D( uImageUnit, vST ).rgb     , LUMCOEFFS );
float im1m1 = dot( texture2D( uImageUnit, vST-stpp ).rgb, LUMCOEFFS );
float ip1p1 = dot( texture2D( uImageUnit, vST+stpp ).rgb, LUMCOEFFS );
float im1p1 = dot( texture2D( uImageUnit, vST-stpm ).rgb, LUMCOEFFS );
float ip1m1 = dot( texture2D( uImageUnit, vST+stpm ).rgb, LUMCOEFFS );
float im10 =  dot( texture2D( uImageUnit, vST-stp0 ).rgb, LUMCOEFFS );
float ip10 =  dot( texture2D( uImageUnit, vST+stp0 ).rgb, LUMCOEFFS );
float i0m1 =  dot( texture2D( uImageUnit, vST-st0p ).rgb, LUMCOEFFS );
float i0p1 =  dot( texture2D( uImageUnit, vST+st0p ).rgb, LUMCOEFFS) );
float h = -1.*im1p1 - 2.*i0p1 - 1.*ip1p1  +  1.*im1m1 + 2.*i0m1 + 1.*ip1m1;
float v = -1.*im1m1 - 2.*im10 - 1.*im1p1  +  1.*ip1m1 + 2.*ip10 + 1.*ip1p1;

float mag = sqrt( h*h + v*v );
vec3 target = vec3( mag,mag,mag );
color = vec4( mix( irgb, target, T ), 1. );
```

# Edge Detection

T = 0.



T = 0.5

T = 1.

# Toon Rendering

```
float mag = sqrt( h*h + v*v );
if( mag > uMagTol )
{
        gl_FragColor= vec4( 0., 0., 0., 1. );
}
else
{
        rgb.rgb *= uQuantize;
        rgb.rgb += vec3( .5, .5, .5 );
        ivec3 irgb = ivec3( rgb.rgb );
        rgb.rgb = vec3( irgb ) / uQuantize;
        gl_FragColor= vec4( rgb, 1. );
}
```

# Toon Rendering

**Original Image**

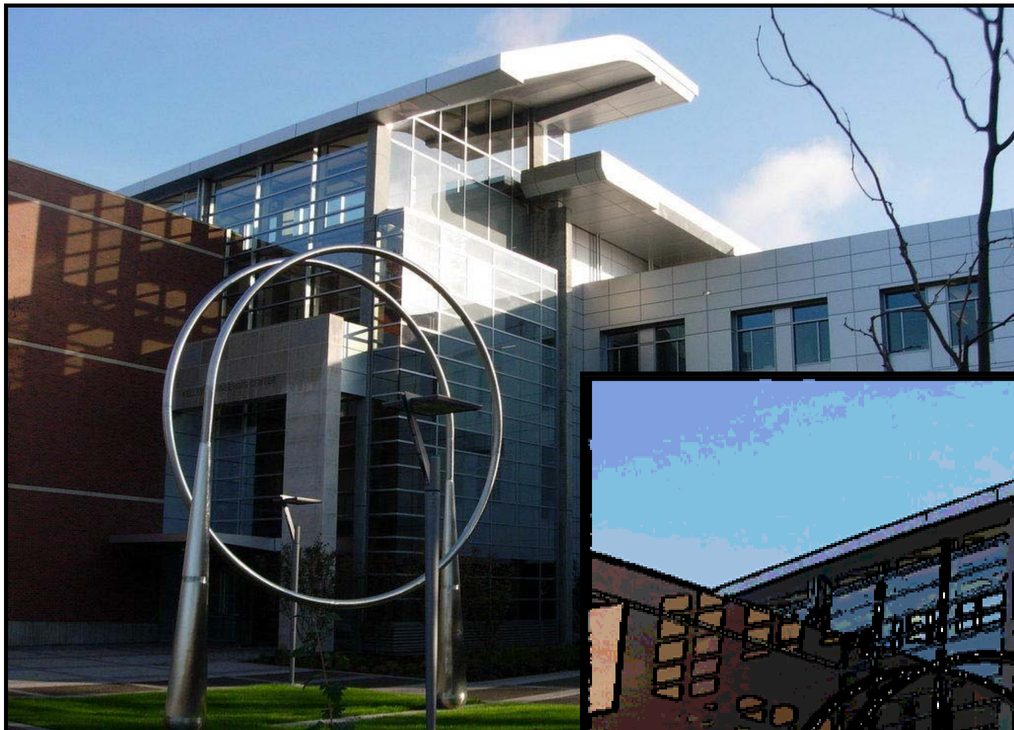
**Colors Quantized**

**Outlines Added**

# Toon Rendering for Non-Photorealistic Effects

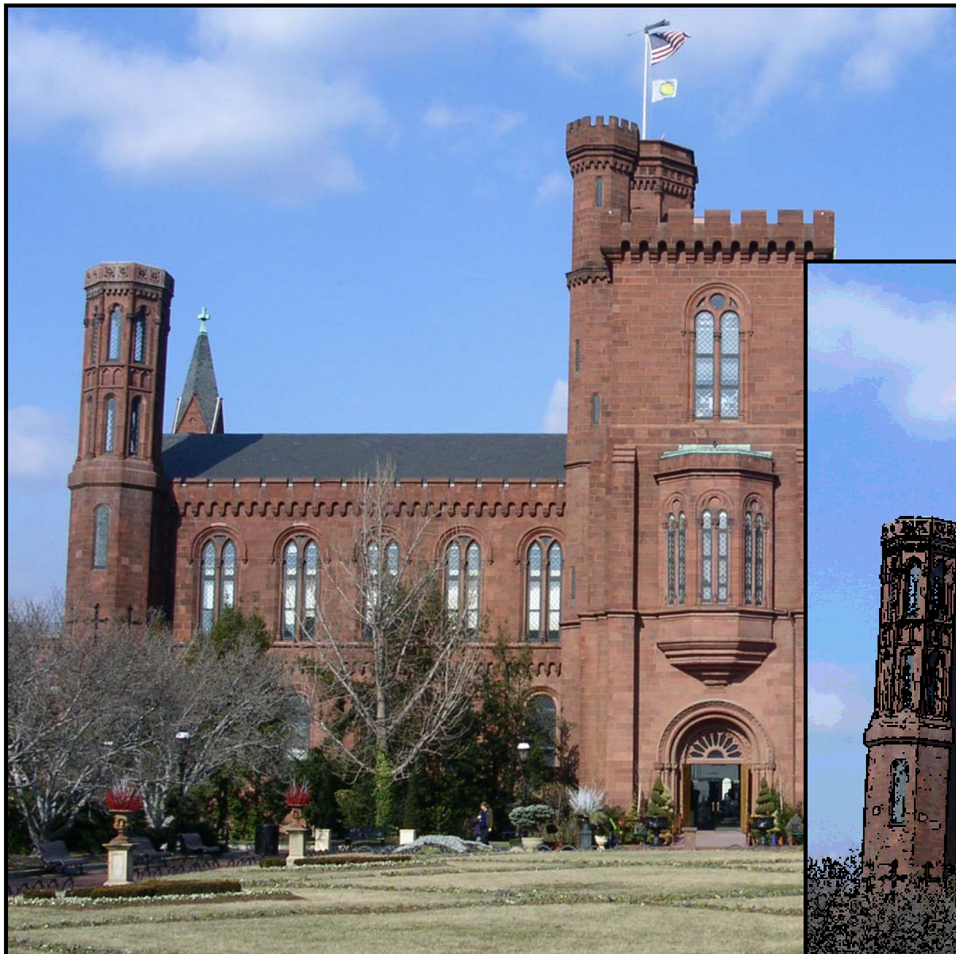


Use the GPU to enhance scientific, engineering, and architectural illustration



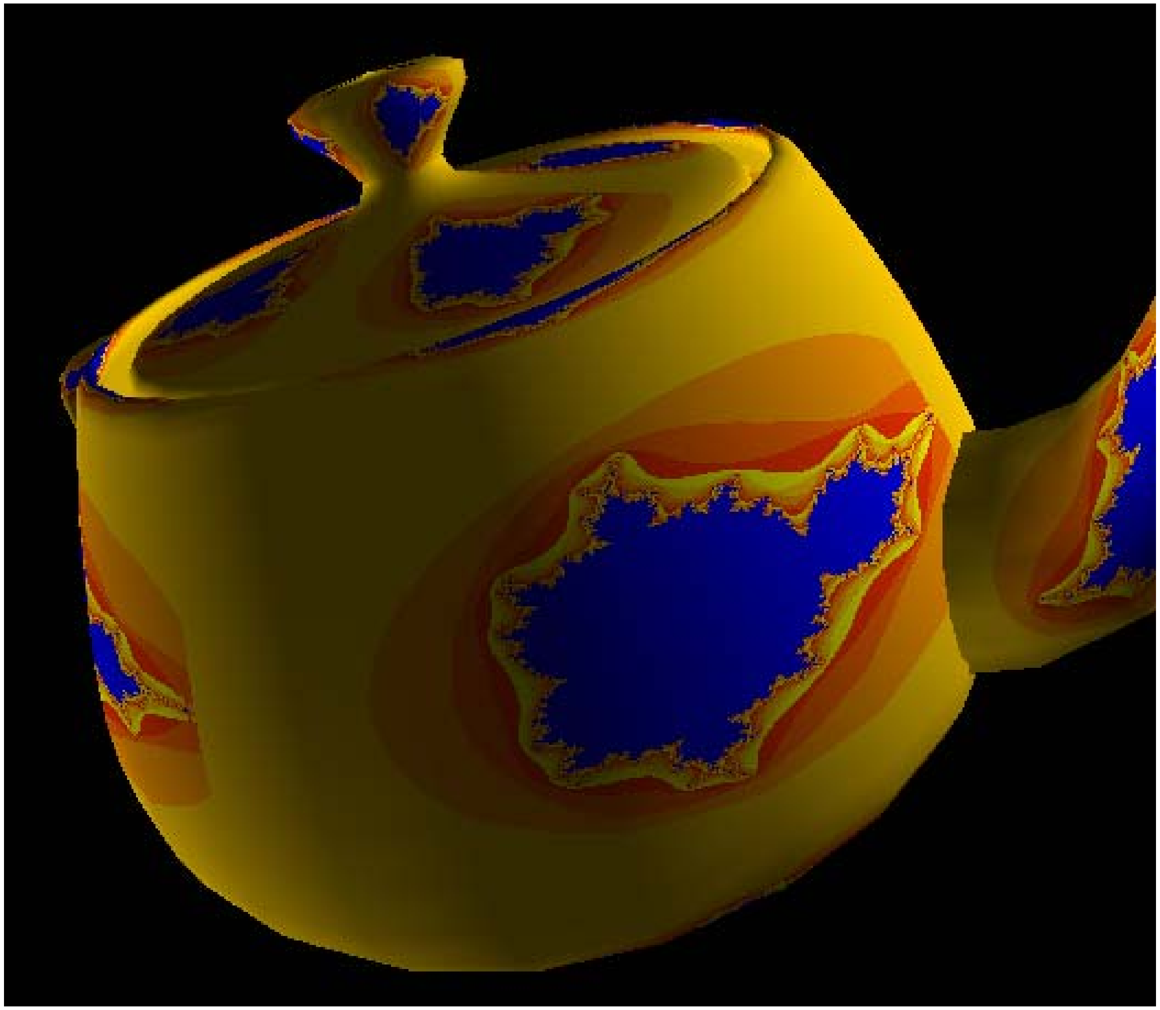
# Toon Rendering for Non-Photorealistic Effects

Use the GPU to enhance scientific, engineering, and architectural illustration



## Mandelbrot Set

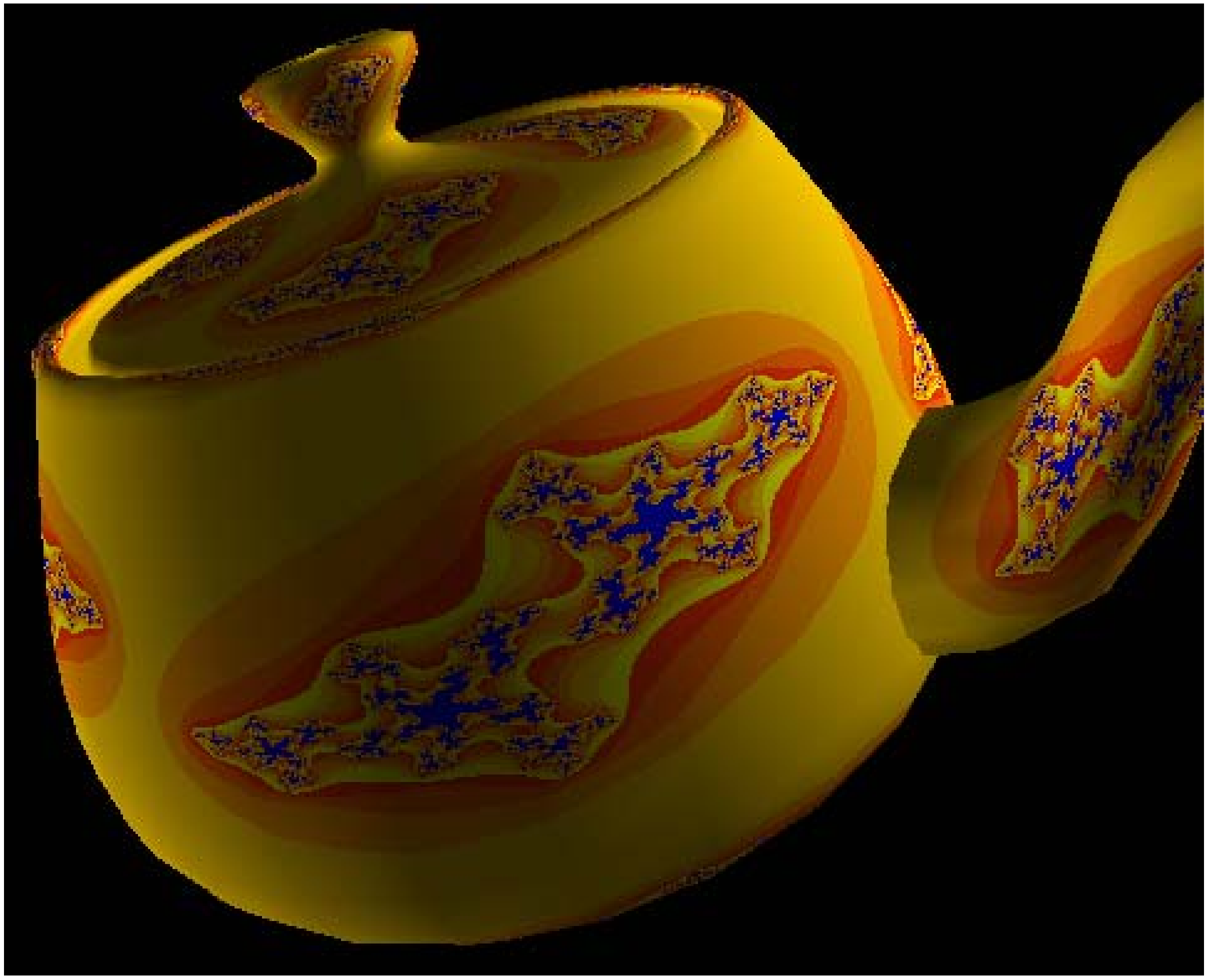
$$Z_{i+1} = Z_i^2 + Z_0$$



How fast does it converge, if ever?

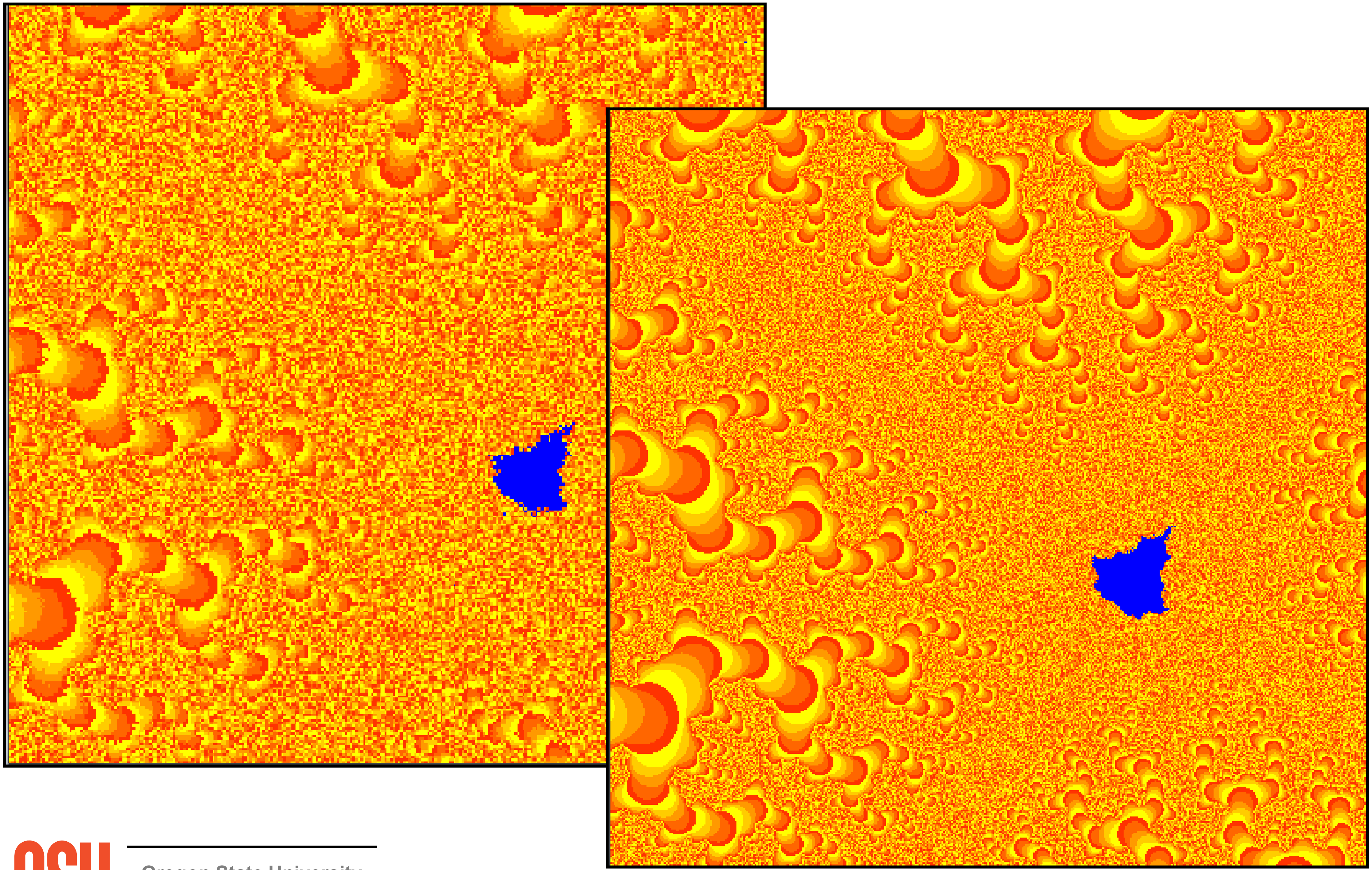
# Julia Set

$$Z_{i+1} = Z_i^2 + C$$



How fast does it converge, if ever?

# Using Double Precision



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