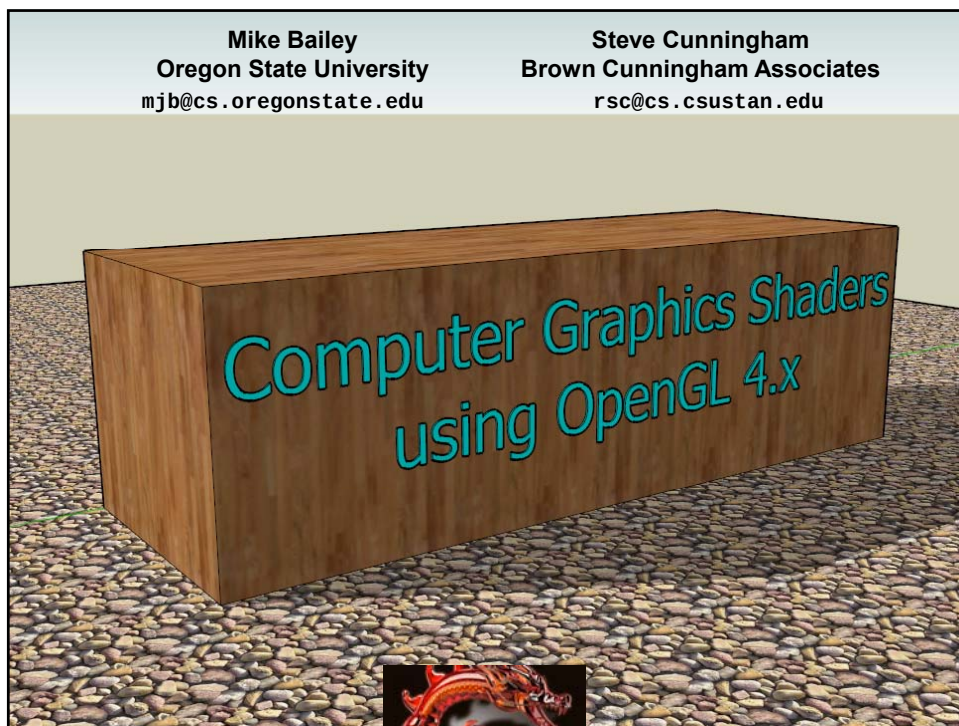




Mike Bailey



- Professor of Computer Science, Oregon State University
- Has worked at Sandia Labs, Purdue University, Megatek, San Diego Supercomputer Center (UC San Diego), and OSU
- Has taught over 4,000 students in his classes
- mjb@cs.oregonstate.edu



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Steve Cunningham



- Retired Professor of Computer Science, California State University Stanislaus
- Has served as chair of both the SIGGRAPH Education Board and the Eurographics Education Board
- Has written 7 books on computer graphics topics
- rsc@cs.csustan.edu



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Schedule



- 0:00 Welcome and Overview
- 0:05 Review of the Graphics Pipeline
- 0:15 Basic Shader Concepts
- 0:30 Transformations
- 0:45 Introduction to GLSL
- 1:00 GLSL Variables
- 1:15 glman
- 1:30 Vertex Shaders
- 1:45 Break
- 2:00 Fragment Shaders
- 2:15 Image Manipulation
- 2:30 Textures
- 2:45 Noise
- 3:00 Geometry and Tessellation Shaders
- 3:30 Questions and Answers / Discussion



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Two Windows Program Executables and Lots of Shader Files



Many of you have them on the *glman* CD

For those who don't, you can get a .zip file of everything by going to:

<http://cs.oregonstate.edu/~mjb/glman>

and following the link that says "SIGGRAPH Asia 2011 Attendees"

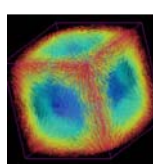
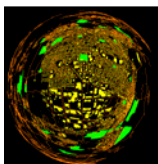
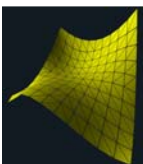
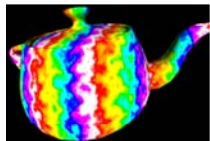
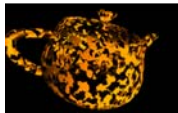
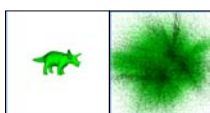
Feel free to unload them now on your laptop (all in the same folder) and follow along with the examples.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Why Do We Care About Graphics Shaders?



1. You can get effects that are difficult or impossible to get any other way
2. You can get innovative data displays
3. You get a much better understanding of the graphics pipeline
4. The fixed-function pipeline was deprecated in OpenGL Desktop starting with OpenGL 3.0
5. The fixed-function pipeline has completely gone away in OpenGL ES 2.0



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Start with Some Terminology



Fragment – a “pixel-to-be”: all of the information about that pixel is available, but the pixel’s color has not yet been determined

Fragment Processor – the part of the graphics pipeline that takes all of the information about a fragment and determines what color to paint there

Fragment Shader – the code you can write to determine the color to paint at a particular fragment

Geometry Shader – the code that you can write to convert or expand one form of geometry into another

GLSL – the OpenGL Shading Language

OpenGL – a multi-vendor, multi-platform, multi-operating system graphics API

Tessellation Shader – the code that you can write to adaptively convert coarse geometry into much finer geometry

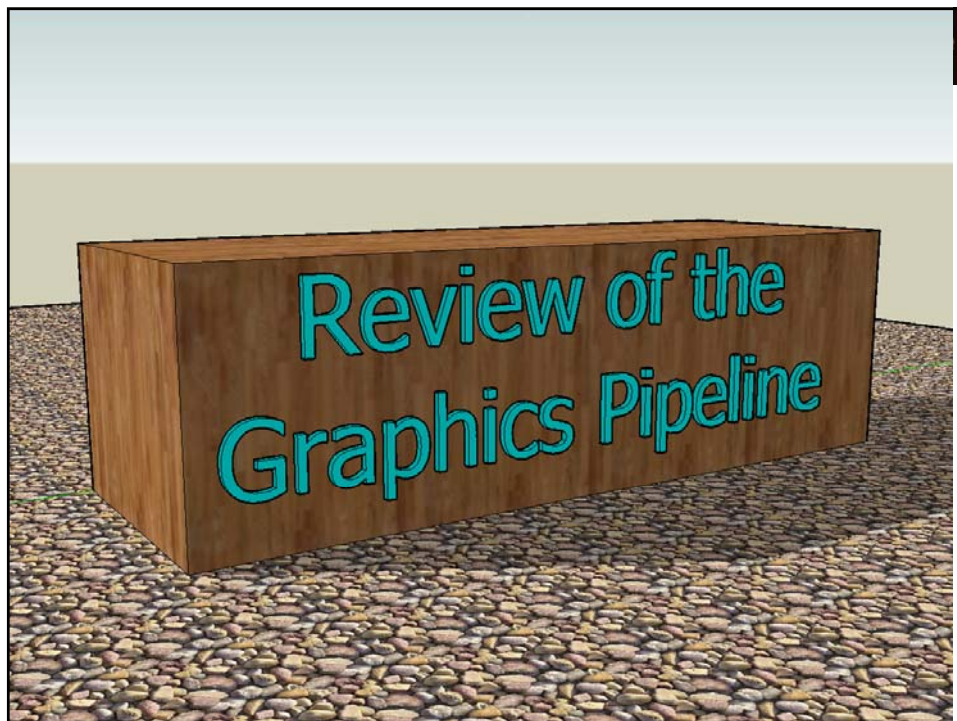
Texture – an image (read or computed) to be attached to a piece of geometry

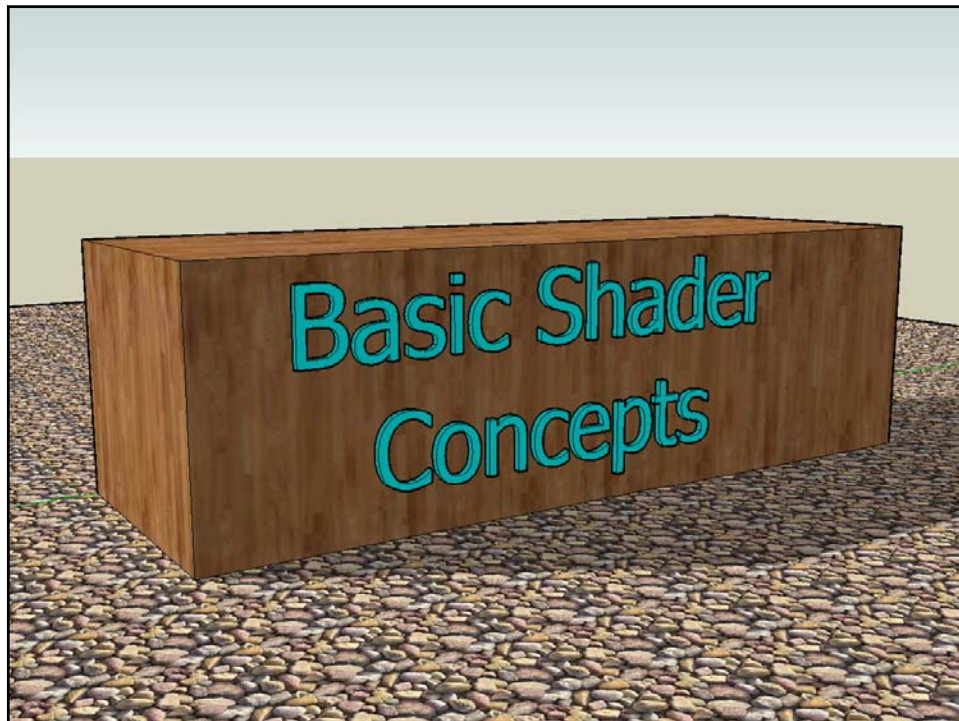
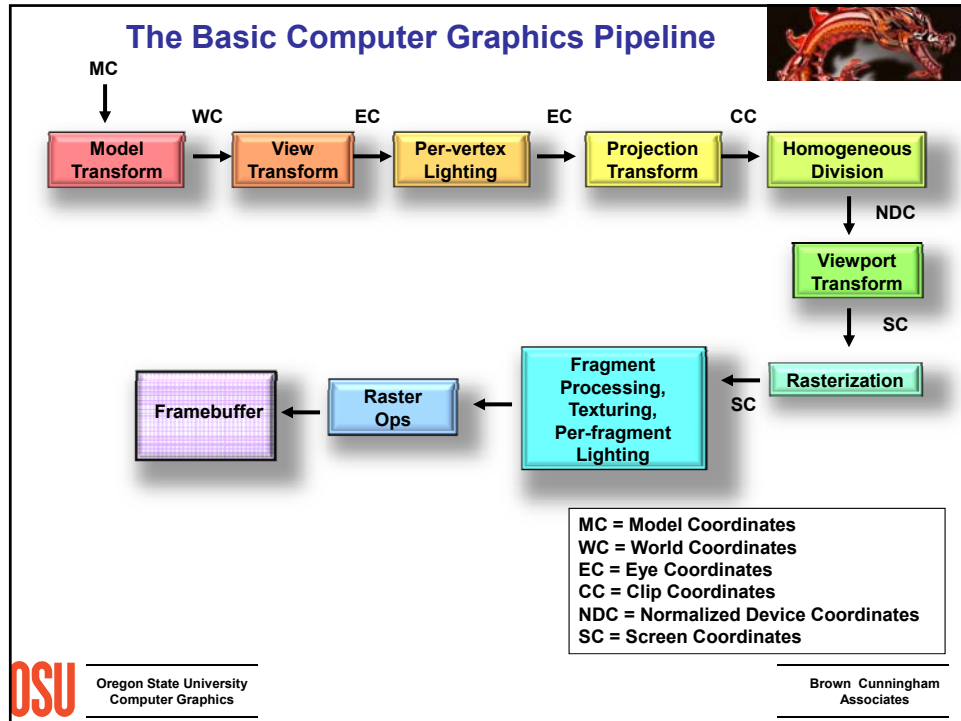
Vertex Processor – the part of the graphics pipeline that handles vertices, from model coordinates to clipped screen space coordinates

Vertex Shader – the code that you can write to perform the transformations of the vertices and set auxiliary values

OS

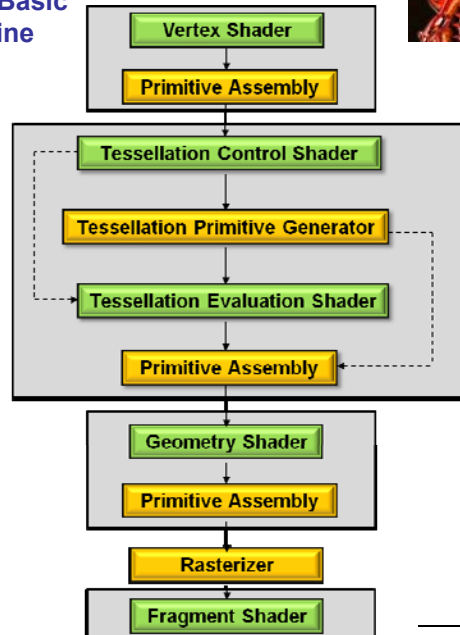
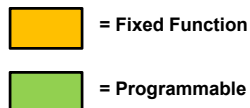
ham





The Shaders' View of the Basic Computer Graphics Pipeline

- In general, you want to have a vertex and fragment shader as a minimum.
- A missing stage is OK. The output from one stage becomes the input of the next stage that is there.
- The last stage before the fragment shader feeds its output variables into the **rasterizer**. The interpolated values then go to the fragment shaders



Oregon State University
Computer Graphics

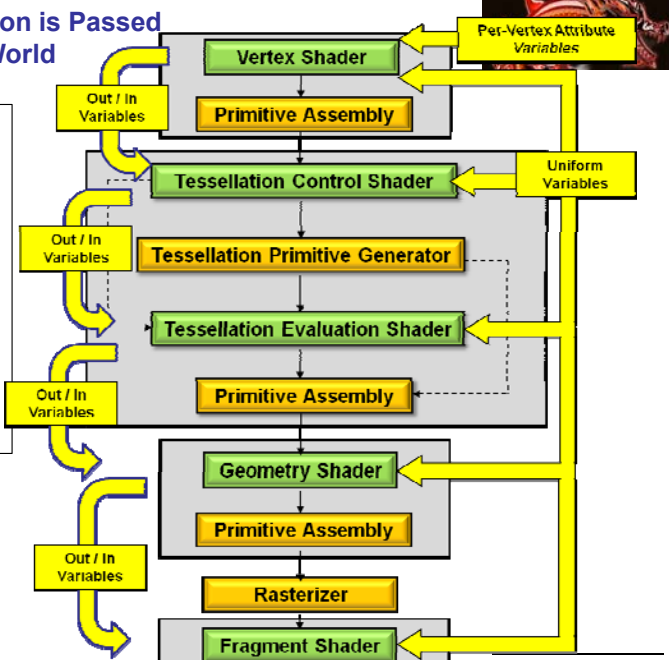
Brown Cunningham
Associates

Here's How Information is Passed Around the Shader World

Attribute Variables are passed per-vertex from the application into the vertex shader.

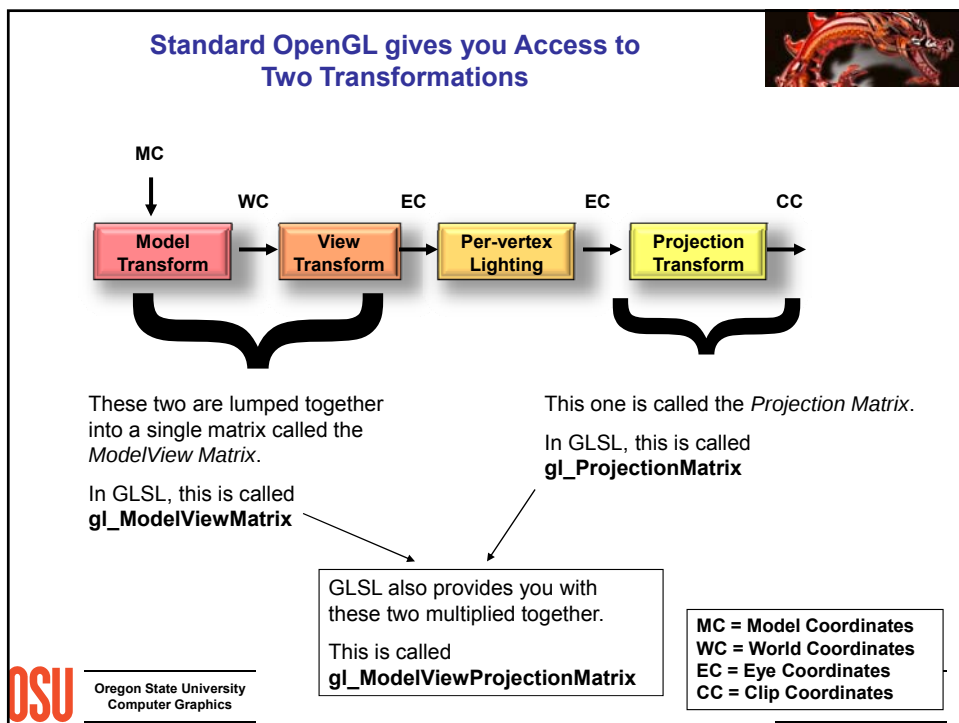
Uniform Variables are passed per-primitive from the application into all shaders. To each shader, they look like read-only global variables.

Out and In Variables are used to pass information from one stage to the next



Oregon State University
Computer Graphics

Brown Cunningham
Associates



Producing Transformed Coordinates and Normals



```
vec4 ModelCoords = gl_Vertex ;
```

```
vec4 EyeCoords = gl_ModelViewMatrix * gl_Vertex ;
```

```
vec4 ClipCoords = gl_ModelViewProjectionMatrix * gl_Vertex ;
```

GLSL also gives you the matrix to transform normal vectors. It performs the same operations on normal vectors as the ModelView matrix does on vertices.

In GLSL, this is called **gl_NormalMatrix**

It is actually the transpose of the inverse of the ModelView matrix. (Trust us on this...)

```
vec3 TransfNorm = gl_NormalMatrix * gl_Normal ;
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

GLSL Deprecation



Variables like **gl_Vertex** and **gl_ModelViewMatrix** have been built-in to the GLSL language.

However, starting with Desktop OpenGL 3.0, they have been deprecated in favor of you defining your own variables and passing them in from the application yourself. The built-ins still work, but be prepared for them to go away some day. Also, OpenGL ES has already completely *eliminated* the built-ins.

What to do?

In these notes, we have chosen to pretend that we have created variables in an application and have passed them in. So, the previous lines of code would be changed to look like:

```
vec4 ModelCoords = gl_Vertex ;  
vec4 ModelCoords = aVertex ;
```

```
vec4 EyeCoords = gl_ModelViewMatrix * gl_Vertex ;  
vec4 EyeCoords = uModelViewMatrix * aVertex ;
```

```
vec4 ClipCoords = gl_ModelViewProjectionMatrix * gl_Vertex ;  
vec4 ClipCoords = uModelViewProjectionMatrix * aVertex ;
```

```
vec3 TransfNorm = gl_NormalMatrix * gl_Normal ;  
vec3 TransfNorm = uNormalMatrix * aNormal ;
```

Why do some of our
variables begin with 'a'?
Why do some begin with 'u'?



Oregon State University
Computer Graphics

Our Own Variable Naming Convention



With 7 different places GLSL variables can be written, we have decided to adopt a naming convention to help us recognize what variables came from what sources:

Beginning letter(s)	Means that the variable ...
a	Is a per-vertex attribute from the application
u	Is a uniform variable from the application
v	Came from the vertex shader
tc	Came from the tessellation control shader
te	Came from the tessellation evaluation shader
g	Came from the geometry shader
f	Came from the fragment shader

This isn't part of "official" OpenGL - it is *our* way of handling the confusion



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Handling the Transition Now



This is how we equivalence our new names to the deprecated (but still working) ones:

```
// uniform variables:

#define uModelViewMatrix      gl_ModelViewMatrix
#define uProjectionMatrix     gl_ProjectionMatrix
#define uModelViewProjectionMatrix gl_ModelViewProjectionMatrix
#define uNormalMatrix         gl_NormalMatrix
#define uModelViewMatrixInverse gl_ModelViewMatrixInverse

// attribute variables:

#define aColor                 gl_Color
#define aNormal                gl_Normal
#define aVertex                gl_Vertex
#define aTexCoord0             gl_MultiTexCoord0
#define aTexCoord1             gl_MultiTexCoord1
#define aTexCoord2             gl_MultiTexCoord2
#define aTexCoord3             gl_MultiTexCoord3
#define aTexCoord4             gl_MultiTexCoord4
#define aTexCoord5             gl_MultiTexCoord5
#define aTexCoord6             gl_MultiTexCoord6
#define aTexCoord7             gl_MultiTexCoord7
```

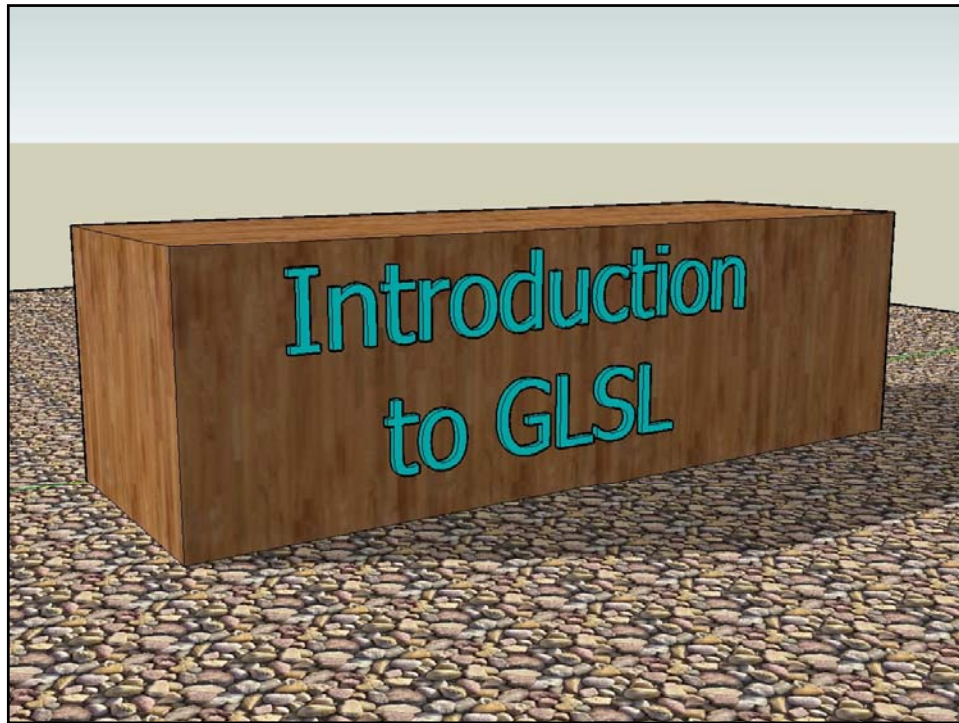
File *gstap.h*

This isn't part of "official" OpenGL - it is *our* way of handling the transition

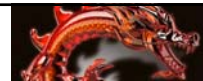


Oregon State University
Computer Graphics

Brown Cunningham
Associates



GLSL Shaders Look Like C With Extensions for Graphics:



- Types include `int`, `ivec2`, `ivec3`, `ivec4`
- Types include `float`, `vec2`, `vec3`, `vec4`
- Types include `mat2`, `mat3`, `mat4`
- Types include `bool`, `bvec2`, `bvec3`, `bvec4`
- Types include `sampler` to access textures
- Vector components are accessed with `[index]` or with the name sets: `.rgba`, `.xyzw`, or `.stpq`
- Vector components can be "swizzled" (`c1.rgba = c2.abgr`)
- `discard` operator used in fragment shaders to discard fragments
- Type qualifiers: `out`, `in`, `const`, `uniform`, `flat`, `noperspective`
- Procedure type qualifiers: `in`, `out`, `inout`



Oregon State University
Computer Graphics

Brown Cunningham
Associates

GLSL Shaders Are Missing Some C-isms:



- No type casts (use constructors instead)
- No automatic promotion
- No pointers
- No strings
- No enums



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Here's What a GLSL Vertex Shader Looks Like



```
out vec4 vColor;
out float vX, vY, vZ;
out float vLightIntensity;

const vec3 LIGHTPOS = vec3( 0., 0., 10. );

void
main()
{
    vec3 TransNorm = normalize( uNormalMatrix * aNormal );
    vec3 ECposition = ( uModelViewMatrix * aVertex ).xyz;
    vLightIntensity = dot( normalize( LIGHTPOS - ECposition ), TransNorm );
    vColor = aColor;
    vec3 MCposition = aVertex.xyz;

    vX = MCposition.x;
    vY = MCposition.y;
    vZ = MCposition.z;

    gl_Position = uModelViewProjectionMatrix * aVertex;
}
```

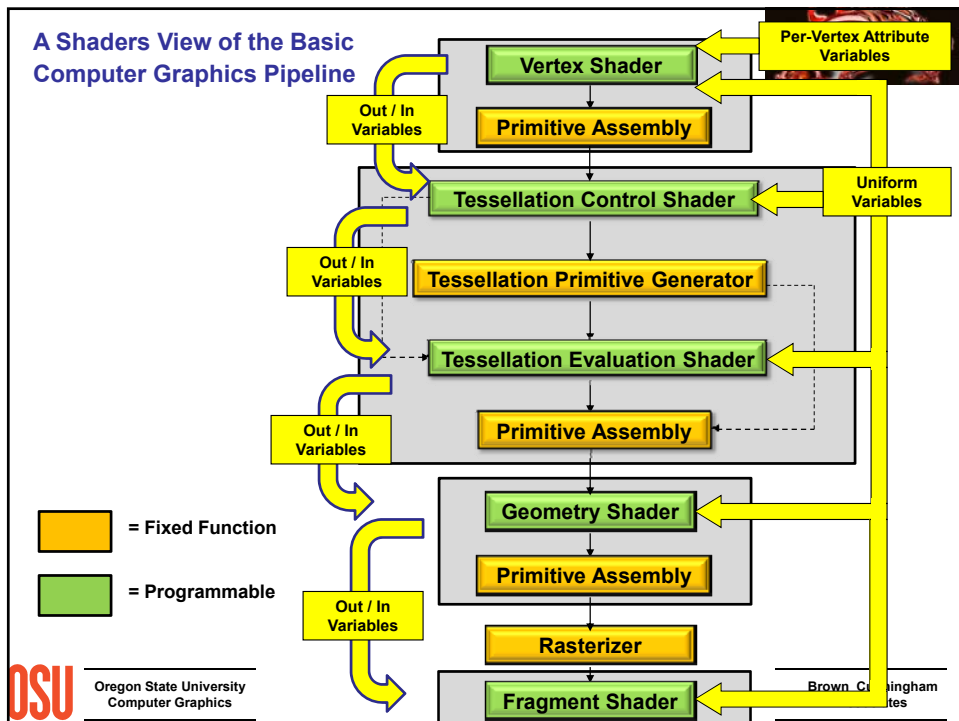
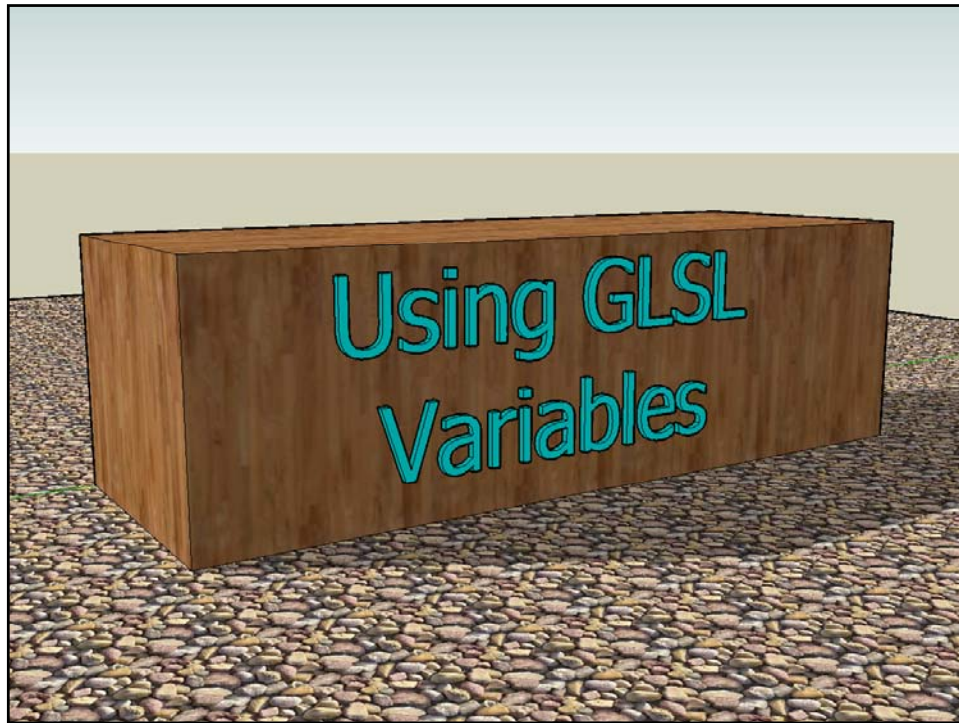


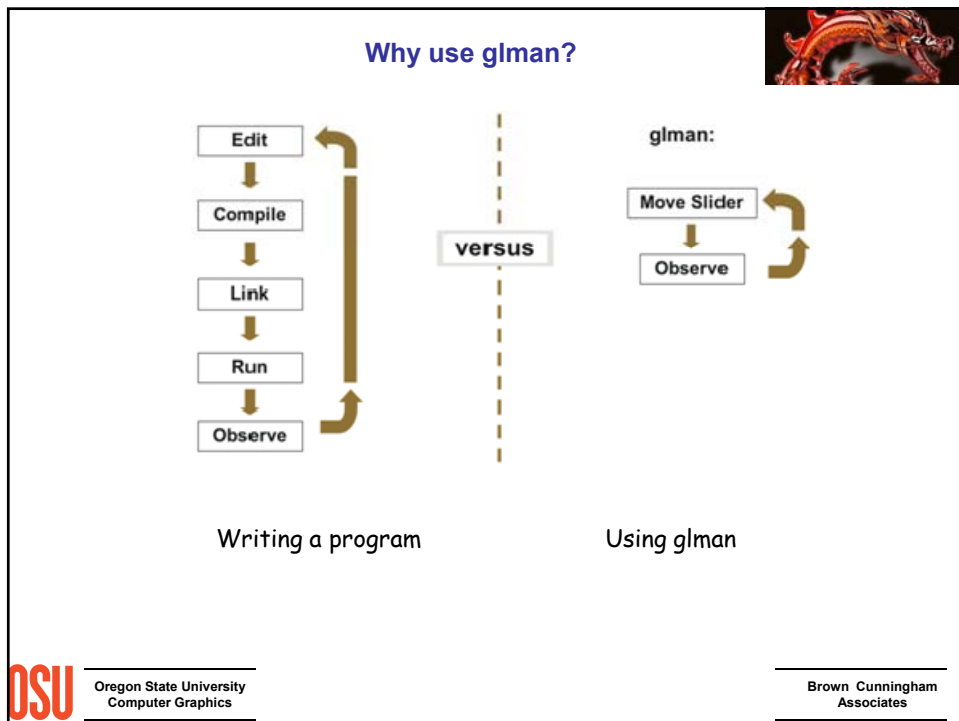
Don't worry about the details right now, just take comfort in the fact that it is C-like and that there appears to be a lot of support routines for you to use



Oregon State University
Computer Graphics

Brown Cunningham
Associates





Load or re-load a .glib file

Edit a specific type of file

Dump an arbitrary-resolution BMP file

Display the speed of the display (fps)

Transformations in the projection matrix

Transformations in the modelview matrix

Allow picking and transformation of individual objects

glman
User
Interface

Oregon State University
Computer Graphics

Brown Cunningham
Associates

glman is looking for up to 6 different files

- A **.glib** file that acts as a scene description script
- A **.vert** file that contains the vertex shader
- A **.frag** file that contains the fragment shader
- Optional **.tcs** and **.tes** files that contain the tessellation control shader and the tessellation evaluation shader.
- An optional **.geom** file that contains the geometry shader

Oregon State University
Computer Graphics

Brown Cunningham
Associates

Sample .glib file and the User Interface it creates

```
##OpenGL GLIB
Perspective 70
LookAt 0 0 3 0 0 0 0 1 0

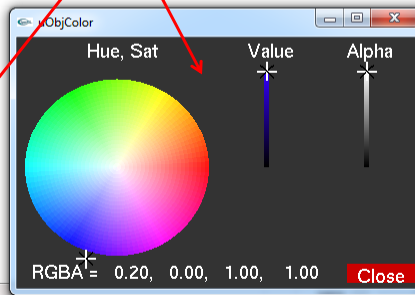
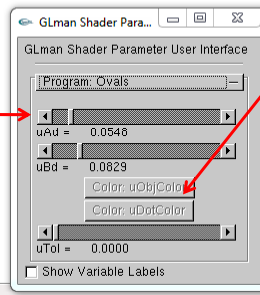
Gstap ←

Vertex ovals.vert
Fragment ovals.frag
Program Ovals

uAd <.01 .05 .5> uBd <.01 .05 .5> \
uObjColor { .2 0. 1. } \
uDotColor { 1. 1. 1. } \
uTol <0. 0. 1.>
```

Equivalence our names to deprecated names

Sphere



Oregon State University
Computer Graphics

Brown Cunningham
Associates

.glib Range Variables

- Scalar variables are just listed as numbers.

Array variables are enclosed in square brackets, as [].

- Range variables are enclosed in angle brackets, as < >. These are scalar variables, and *glman* automatically generates a slider in the Uniform Variable user interface for each range variable so that you can then change this value as *glman* executes. The three values in the brackets are : <min current max>, e.g., <0. 5. 10.>. *glman* will look into your shader program's symbol table to decide if this range variable should be a *float* an *int*, or a *bool*, and will create a slider of the appropriate type.

- Color variables are enclosed in curly brackets, as { }. Color variables may be either RGB or RGBA, as:
{red green blue}

or

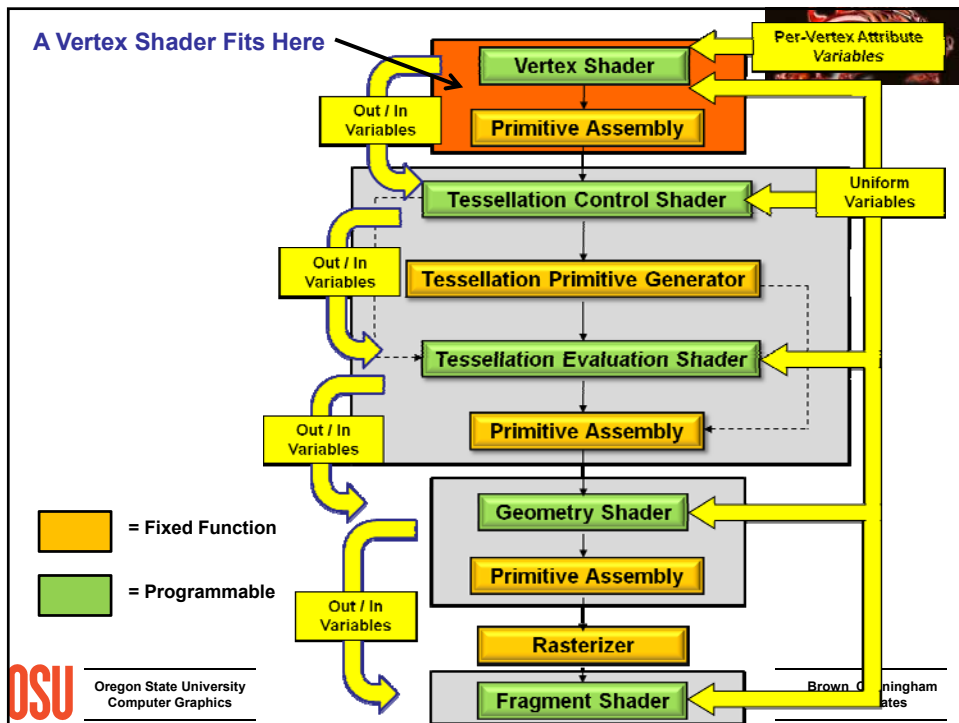
{red green blue alpha}

This will generate a button in the UI panel that, when clicked, brings up a color selector window. The color selector allows you to change the value of this color variable as *glman* executes.

- A Boolean variable is available to select or de-select options in your shader. The *glman* user interface will automatically create a checkbox in the user interface window. In the GLIB file, a Boolean variable has a name and then the word true or the word false inside parenthesis, e.g., "(true)". This is the initial setting of the checkbox.

- Multiple vertex-geometry-fragment-program combinations are allowed in the same GLIB file. If there is more than one combination, then they will appear as separate rollout panels in the user interface. The first program rollout will be open, and all the others will be closed. Open the ones you need when you need them.





What does a Vertex Shader Do?



The basic function of a vertex shader is to take the vertex coordinates and other per-vertex information as supplied by the application, and perform whatever arithmetic is required.

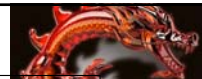
At the same time, the vertex shader can perform various analyses based on those vertex coordinates and other data and prepare variable values for later on in the graphics process.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

A GLSL Vertex Shader Replaces These Operations:



- Vertex transformations
- Normal transformations
- Normal normalization
- Handling of per-vertex lighting
- Handling of per-vertex colors
- Handling of texture coordinates

A GLSL Vertex Shader Does Not Replace These Operations:

- Frustum clipping
- Homogeneous division
- Viewport mapping
- Backface culling
- Polygon mode
- Polygon offset



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Built-in Vertex Shader Variables You Will Use a Lot:



```
vec4 aVertex
vec3 aNormal
vec4 aColor
vec4 aTexCoordi (i=0, 1, 2, ...)
mat4 uModelViewMatrix
mat4 uProjectionMatrix
mat4 uModelViewProjectionMatrix
mat4 uNormalMatrix
```

Note: these are *our* names for these variables. The application would either need to pass them into the shaders under these names, or you would need to #define them to their (deprecated) built-in equivalents.

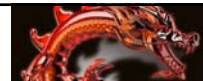
```
// uniform variables:
#define uModelViewMatrix    gl_ModelViewMatrix
#define uProjectionMatrix  gl_ProjectionMatrix
#define uModelViewProjectionMatrix gl_ModelViewProjectionMatrix
#define uNormalMatrix      gl_NormalMatrix

// attribute variables:
#define aColor              gl_Color
#define aNormal            gl_Normal
#define aVertex            gl_Vertex
#define aTexCoord0        gl_MultiTexCoord0
#define aTexCoord1        gl_MultiTexCoord1
#define aTexCoord2        gl_MultiTexCoord2
#define aTexCoord3        gl_MultiTexCoord3
#define aTexCoord4        gl_MultiTexCoord4
#define aTexCoord5        gl_MultiTexCoord5
#define aTexCoord6        gl_MultiTexCoord6
#define aTexCoord7        gl_MultiTexCoord7
```



Oregon State University
Computer Graphics

Sample Vertex Shader: Stripes in Model and Eye Coordinates



```
uniform bool uUseEyeCoords;
out vec4 vColor;
out float vX, vY, vZ;
out float vLightIntensity;

const vec3 LIGHTPOS = vec3( 0., 0., 10. );

void
main()
{
    vec3 transNorm = normalize( uNormalMatrix * aNormal );
    vec3 ECposition = ( uModelViewMatrix * aVertex ).xyz;
    vLightIntensity = dot(normalize( LIGHTPOS - ECposition), transNorm );
    vLightIntensity = abs( vLightIntensity );
    vColor = aColor;
    vec3 MCposition = aVertex.xyz;
    if( uUseEyeCoords )
    {
        vX = ECposition.x;
        vY = ECposition.y;
        vZ = ECposition.z;
    }
    else
    {
        vX = MCposition.x;
        vY = MCposition.y;
        vZ = MCposition.z;
    }
    gl_Position = uModelViewProjectionMatrix * aVertex;
}
```

stripes.glib

This is a good example of why we adopted a consistent naming convention !



Oregon State University
Computer Graphics

The Fragment Shader then sets the color based on the X value.

Associates

Sample Fragment Shader: Stripes in Model and Eye Coordinates



```

in vec4    vColor;
in float   vX, vY, vZ;
in float   vLightIntensity;

out vec4    fFragColor;

uniform float uA;
uniform float uP;
uniform float uTol;

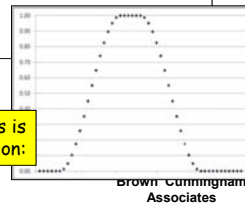
const vec3 WHITE = vec3( 1., 1., 1. );

void
main( )
{
    float f = fract( uA*vX );
    float t = smoothstep( 0.5-uP-uTol, 0.5-uP+uTol, f ) - smoothstep( 0.5+uP-uTol, 0.5+uP+uTol, f );

    vec3 color = mix( WHITE, vColor.rgb, t );
    fFragColor = vec4( vLightIntensity*color, 1. );
}

```

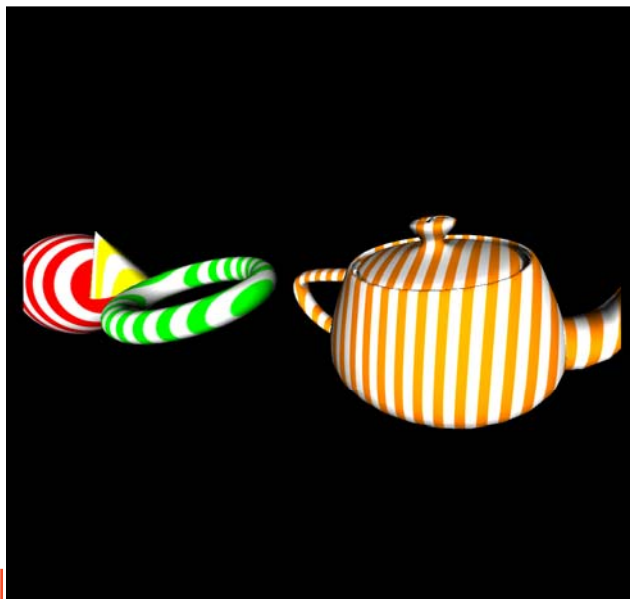
This combination of smoothstep() functions is known as a "smoothpulse" function:



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Sample Vertex Shader: Stripes in Model and Eye Coordinates

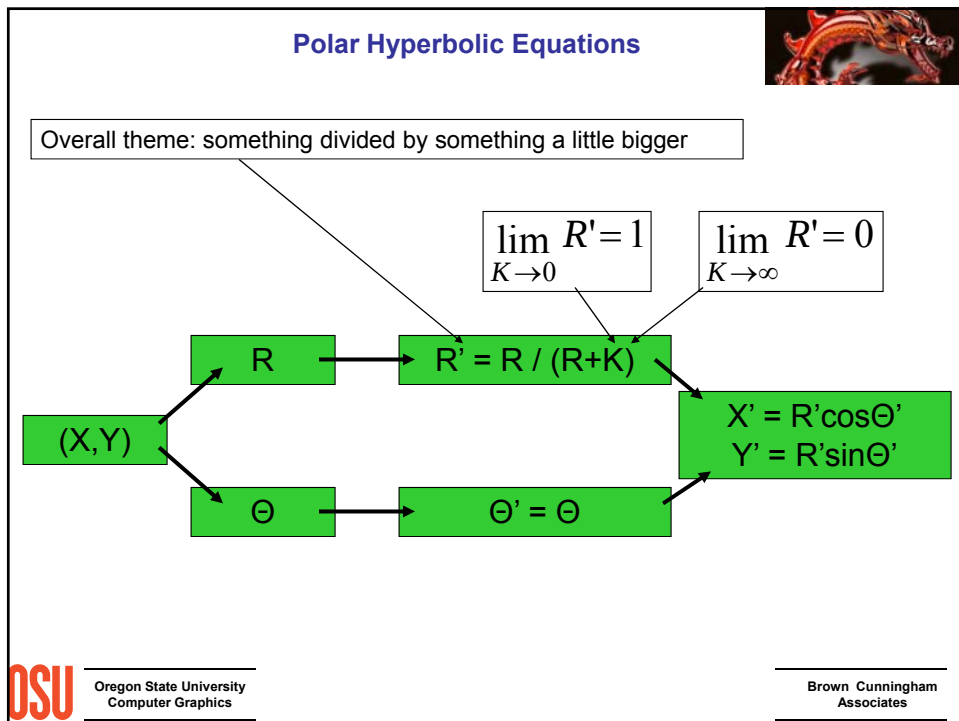
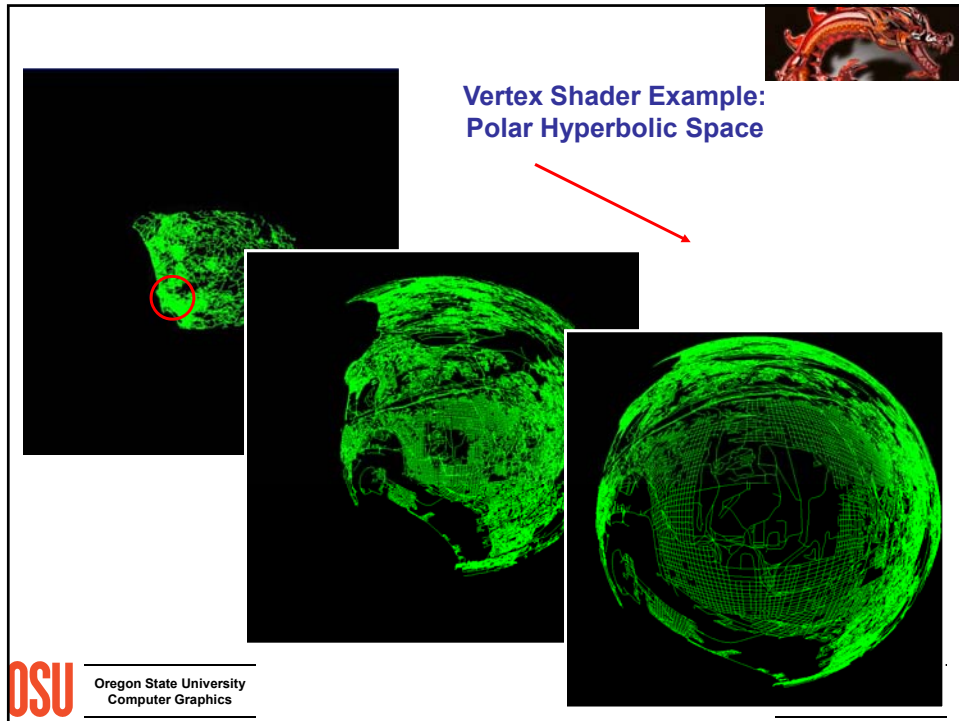


They might
(momentarily) look
the same, but they
don't act the same !



Computer Graphics

Brown Cunningham
Associates



Polar Hyperbolic Equations



$$R = \sqrt{X^2 + Y^2}$$

$$\Theta = \tan^{-1}\left(\frac{Y}{X}\right)$$

$$R' = \frac{R}{R + K}$$

Coordinates moved to outer edge
when $K = 0$

Coordinates moved to center when $K = \infty$

$$X' = R' \cos \Theta = \frac{R}{R + K} \times \frac{X}{R} = \frac{X}{R + K}$$

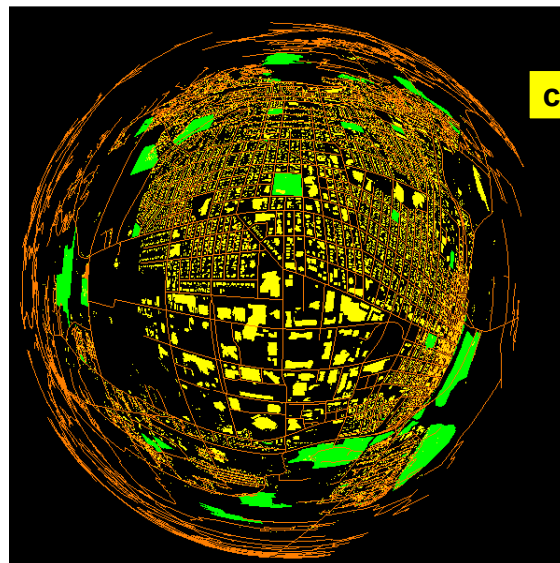
$$Y' = R' \sin \Theta = \frac{R}{R + K} \times \frac{Y}{R} = \frac{Y}{R + K}$$



Oregon State University
Computer Graphics

Brown Cunningham
Associates

A Good Way to Look at Detailed City Streets, Buildings, Parks

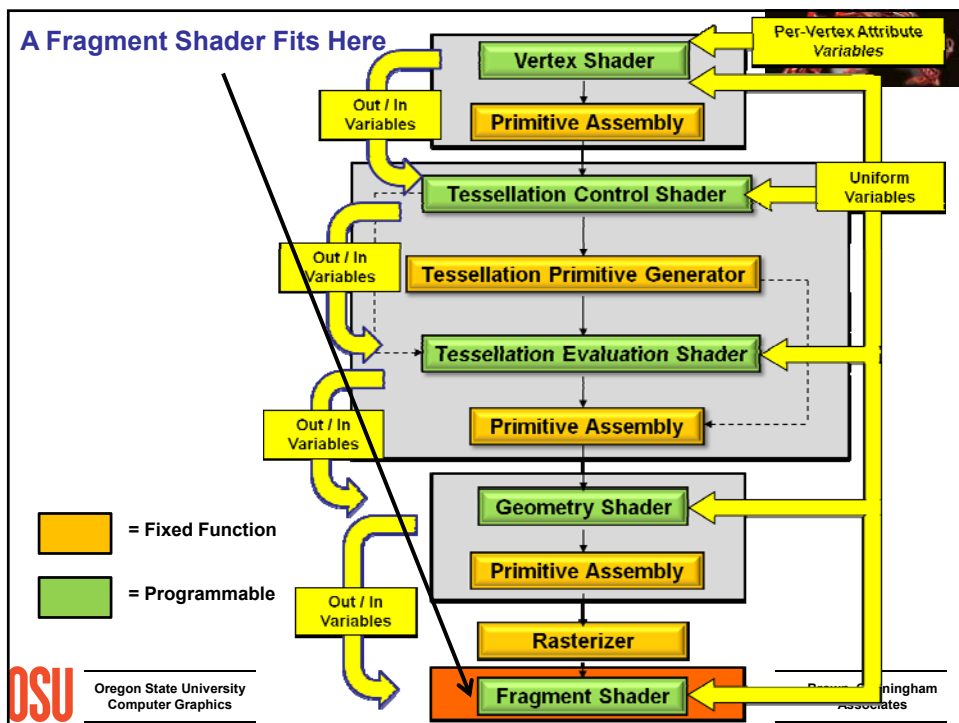
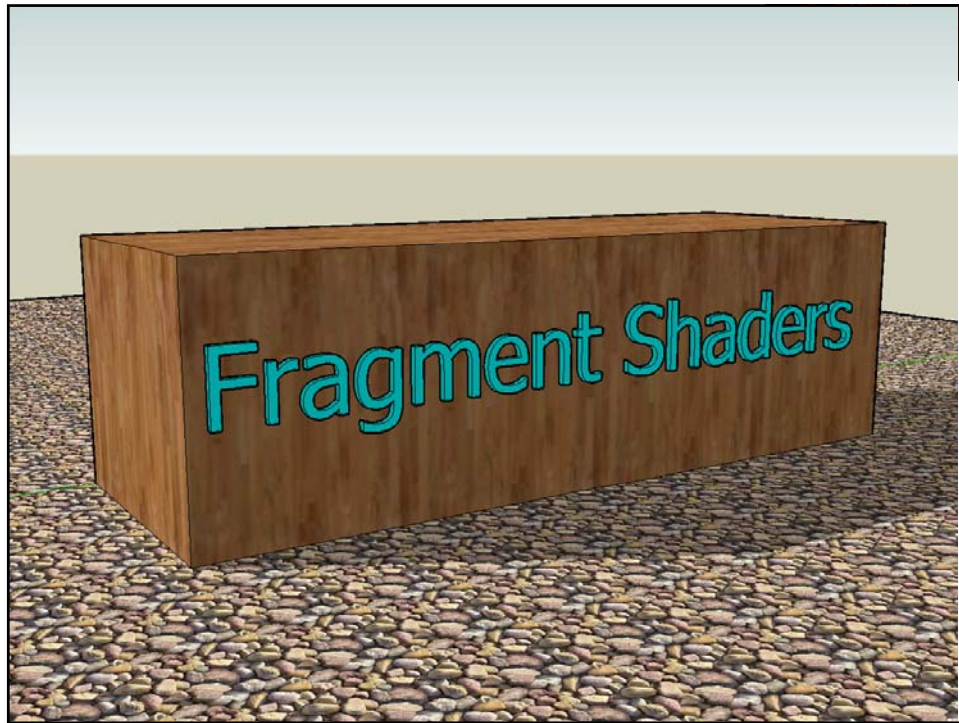


corvallis.glib



Oregon State University
Computer Graphics

Brown Cunningham
Associates



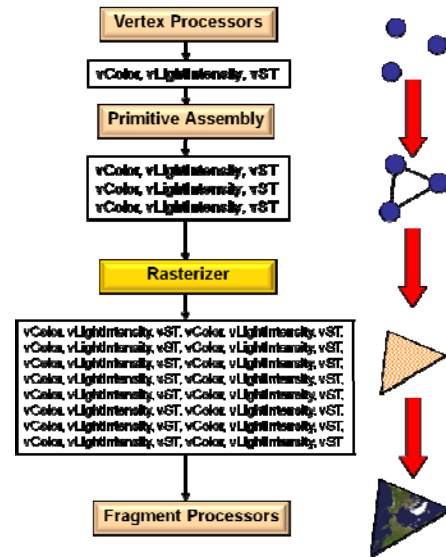
What does a Fragment Shader Do?



The fragment shader starts with the output of the rasterizer. This figure illustrates the rasterizer process, showing how the distinct vertices of a primitive are processed to create the fragments that make up the primitive.

The interpolated quantities, the textures, and the uniform variables are then used to compute the final color of each fragment.

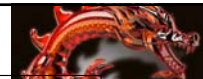
This diagram shows the vertex shaders feeding the rasterizer, but they could also be fed from a geometry or a tessellation shader.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

A GLSL Fragment Shader Replaces These Operations:



- Color computation
- Texturing
- Handling of per-pixel lighting
- Fog
- Blending
- Discarding fragments

A GLSL Fragment Shader Does Not Replace These Operations:

- Stencil test
- Z-buffer test
- Stippling



Oregon State University
Computer Graphics

Brown Cunningham
Associates

The Fragment Shader Variable You Will Use a Lot:



```
out vec4 fFragColor;
```

← You can call this whatever you want. This name is our standard.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Simple Fragment Shader: Setting the Color



```
in float      vLightIntensity;  
  
uniform vec4   uColor;  
  
out vec4       fFragColor;  
  
void main( )  
{  
    fFragColor= vec4( vLightIntensity * uColor.rgb, 1. );  
}
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Fragment Shader: Discarding Fragments



```
uniform vec4 uColor;
uniform float uDensity;
uniform float uFrequency;

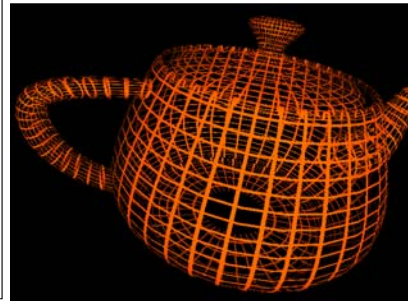
in float vLightIntensity;
in vec2 vST;

out vec4 fFragColor;

void main( )
{
    vec2 stf = vST * uFrequency;

    if( all( fract( stf ) >= uDensity ) )
        discard;

    fFragColor = vec4( vLightIntensity * uColor.rgb, 1. );
}
```



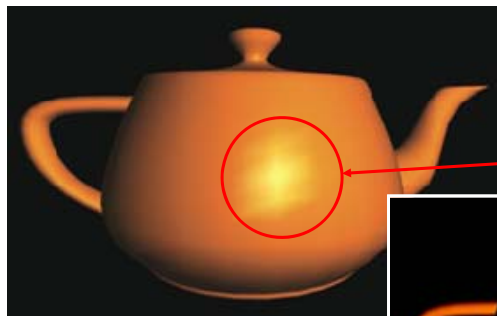
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Per-vertex vs. Per-fragment Lighting



lighting.glib



In per-vertex lighting, the normal at each vertex is turned into a lighted intensity. That intensity is then interpolated throughout the polygon. This gives splotchy polygon artifacts like this.

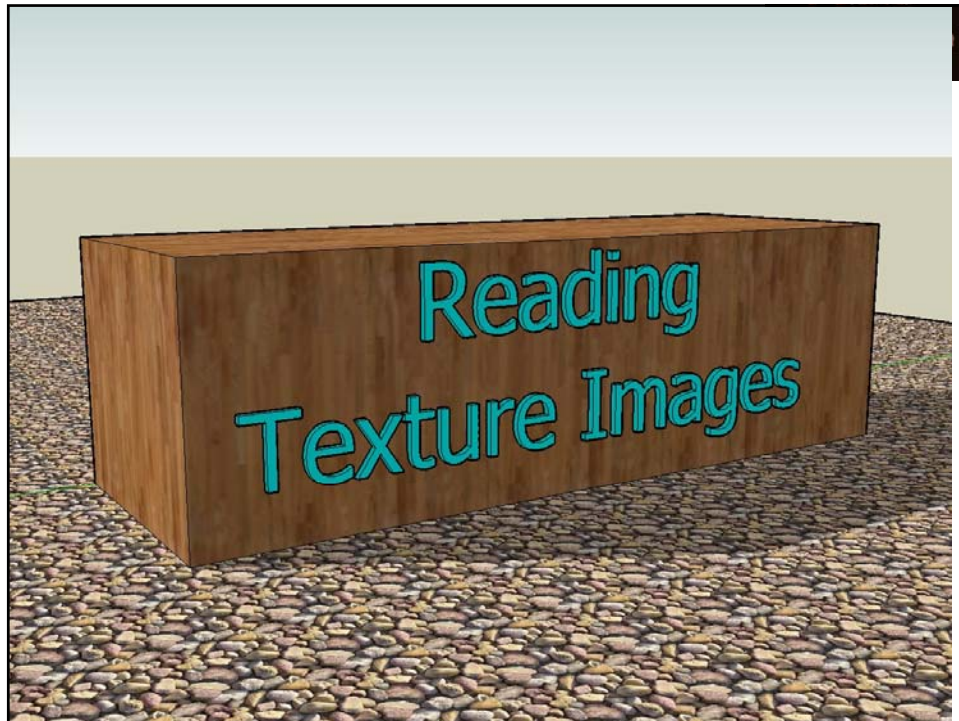


In per-fragment lighting, the normal is interpolated throughout the polygon and turned into a lighted intensity at each fragment. This gives smoother results, like this.



Oregon State University
Computer Graphics

Brown Cunningham
Associates



Some of the Texture-reading Functions



<pre>vec4 texture(sampler1D sampler, float coord) vec4 texture(sampler2D sampler, vec2 coord) vec4 texture(sampler3D sampler, vec3 coord)</pre>	Use the texture coordinate <code>coord</code> to do a texture lookup in the n-D texture currently bound to <code>sampler</code>.
<pre>vec4 texture(samplerCube sampler, vec3 coord)</pre>	Use the texture coordinate <code>coord</code> to do a texture lookup in the cube map texture currently bound to <code>sampler</code>. The direction of <code>coord</code> is used to select in which face to do a two-dimensional texture lookup.

You usually call these routines from a fragment shader (that's why we're covering it here), but in fact you can read textures into any other shader as well.

Texture-reading Example



glib file

```
##OpenGL GLIB

Gstap

Texture2D texture.bmp

Vertex texture.vert
Fragment texture.frag
Program Texture uTexUnit 7

Teapot
```

vert file

```
out vec2 vST;

void
main( )
{
    vST = aTexCoord0.st;
    gl_Position = uModelViewProjectionMatrix * aVertex;
}
```

frag file

```
uniform sampler2D uTexUnit;
in vec2 vST;
out vec4 fFragColor;

void
main( )
{
    vec3 rgb = texture( uTexUnit, vST ).rgb;
    fFragColor = vec4( rgb, 1. );
}
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

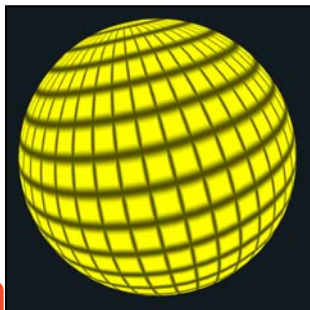
Texture Example



worldtex.bmp



+



=

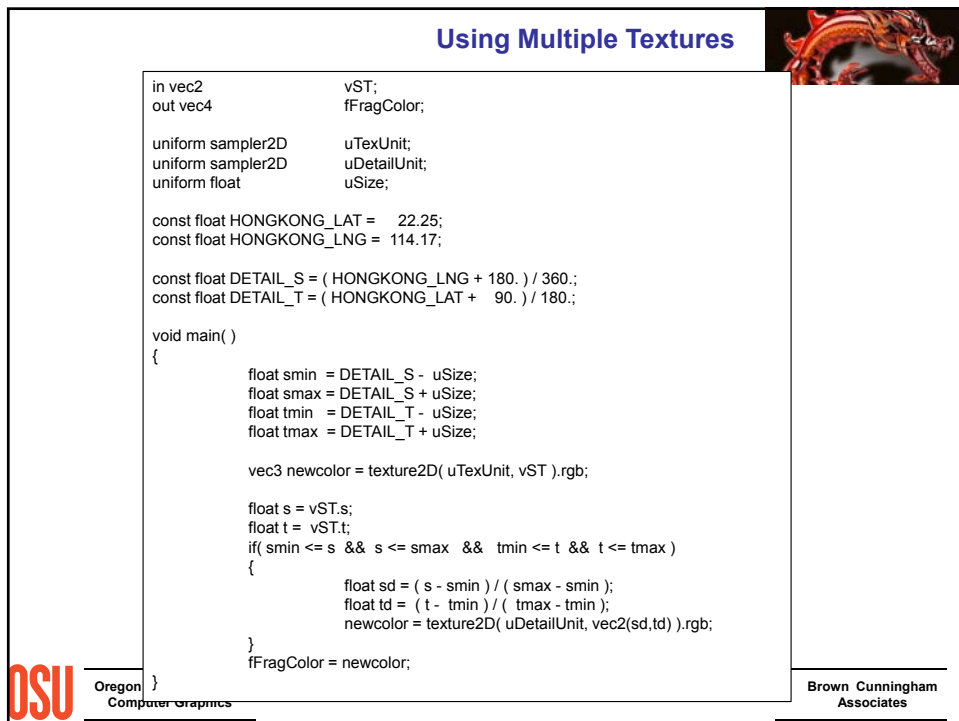
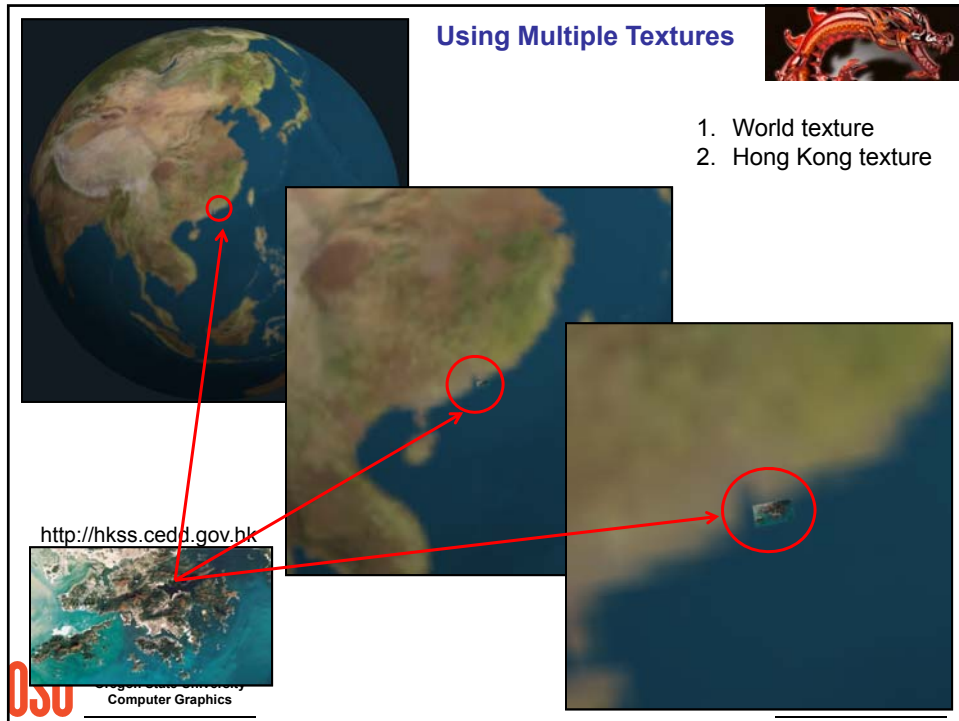


world.glib

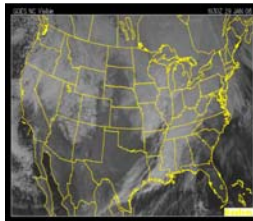


Computer Graphics

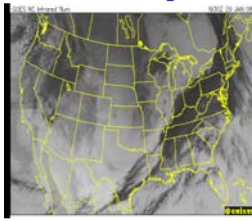
Associates



Using Textures as Data: Where is it Likely to Snow?



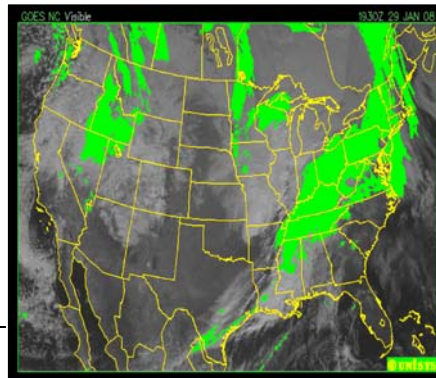
Visible



Infrared



Water vapor



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Using Textures as Data

glib file

```
##OpenGL GLIB

Texture2D 5 goes.visible.bmp
Texture2D 6 goes.infrared.bmp
Texture2D 7 goes.watervapor.bmp

Gstap

Vertex multiband.vert
Fragment multiband.frag
Program MultiBand
    uVisibleUnit 5    uInfraRedUnit 6    uWaterVaporUnit 7
    uVisible <0. 1. 1.> uInfraRed <0. 0. 1.> uWaterVapor <0. 0. 1.>
    uVisibleThreshold <0. 1. 1.>
    uInfraRedThreshold <0. 0. 1.>
    uWaterVaporThreshold <0. 0. 1.>
    uBrightness <0. 1. 3.>

QuadXY
```

vert file

```
out vec2 vST;
void main( )
{
    vST= aTexCoord0.st;
    gl_Position = uModelViewProjectionMatrix * aVertex;
}
```



Brown Cunningham
Associates

Using Textures as Data



frag file, I

```
uniform sampler2D uVisibleUnit;
uniform sampler2D uInfraRedUnit;
uniform sampler2D uWaterVaporUnit;
uniform float uVisible;
uniform float uInfraRed;
uniform float uWaterVapor;
uniform float uVisibleThreshold;
uniform float uInfraRedThreshold;
uniform float uWaterVaporThreshold;
uniform float uBrightness;

in vec2 vST;

out vec4 fFragColor;

void
main( )
{
    vec3 visibleColor = texture( uVisibleUnit, vST ).rgb;
    vec3 infraredColor = texture( uInfraRedUnit, vST ).rgb;
    infraredColor = vec3(1., 1., 1.) - infraredColor;
    vec3 watervaporColor = texture( uWaterVaporUnit, vST ).rgb;

    vec3 rgb;
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Using a Texture as Data



frag file, II

```
if( visibleColor.r - visibleColor.g > .25 && visibleColor.r - visibleColor.b > .25 )
{
    rgb = vec3( 1., 1., 0. ); // state outlines become yellow
}
else
{
    rgb = uVisible*visibleColor + uInfraRed*infraredColor + uWaterVapor*watervaporColor;
    rgb /= 3.;
    vec3 coefs = vec3( 0.296, 0.240, 0.464 );
    float visibleInten = dot(coefs, visibleColor);
    float infraredInten = dot(coefs, infraredColor);
    float watervaporInten = dot(coefs, watervaporColor);
    if( visibleInten > uVisibleThreshold && infraredInten < uInfraRedThreshold && watervaporInten > uWaterVaporThreshold )
    {
        rgb = vec3( 0., 1., 0. );
    }
    else
    {
        rgb *= uBrightness;
        rgb = clamp( rgb, 0., 1. );
    }
}

fFragColor = vec4( rgb, 1. );
}
```

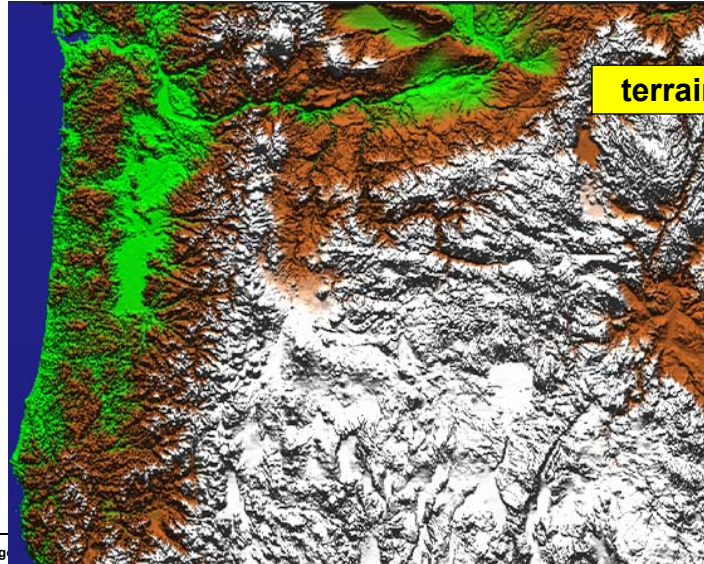


Oregon State University
Computer Graphics

Brown Cunningham
Associates

Bonus Demo

Bump-Mapping a Terrain Surface by Storing Heights in a Texture

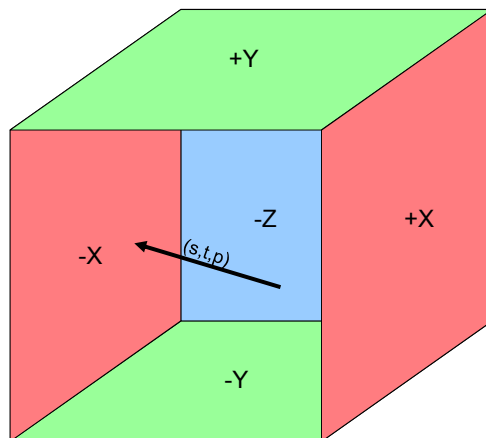


OSU

Oregon State University
Computer Graphics

Brown Cunningham
Associates

Cube Map Texture Lookup: Simulating a Surrounding 3D Environment



- Let L be the texture coordinate of (s, t, p) with the largest magnitude
- L determines which of the 6 2D texture "walls" is being hit by the vector $(-X)$ in this case
- The texture coordinates in that texture are the remaining two texture coordinates divided by L : $(a/L, b/L)$

`vec3 ReflectVector = reflect( vec3 eyeDir, vec3 normal );`

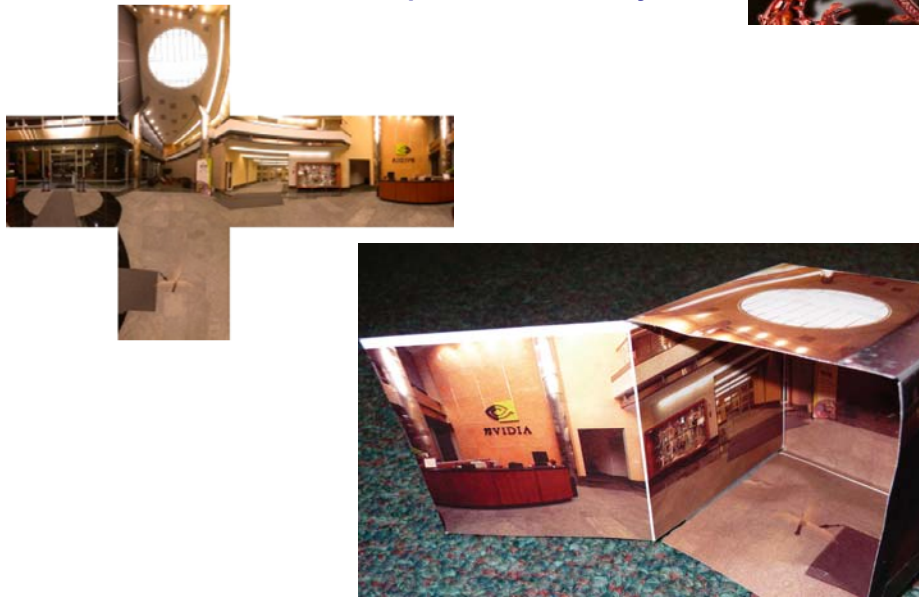
`vec3 RefractVector = refract( vec3 eyeDir, vec3 normal, float Eta );`

OSU

Oregon State University
Computer Graphics

Brown Cunningham
Associates

Cube Map of Nvidia's Lobby



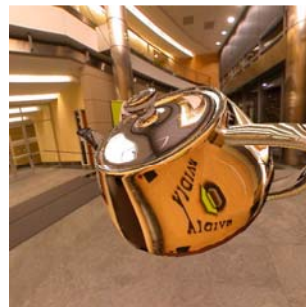
OSU

Oregon State Un
Computer Graphics

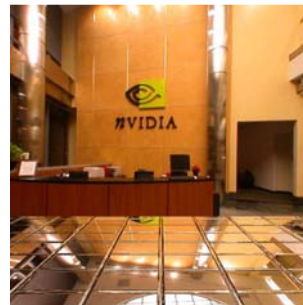
<http://www.codemonsters.de/home/content.php?show=cubemaps>

Cunningham
Associates

Using the Cube Map for Reflection



reflect.glib



OSU

Oregon State Un
Computer Gr

rown Cunningham
Associates

Using the Cube Map for Reflection



```
out vec3 vReflectVector;

void main( )
{
    vec3 ECposition = vec3( uModelViewMatrix * aVertex );
    vec3 eyeDir = ECposition - vec3(0.,0.,0.);           // vector from eye to pt
    vec3 normal = normalize( uNormalMatrix * aNormal );
    vReflectVector = reflect( eyeDir, normal );
    gl_Position = uModelViewProjectionMatrix * aVertex;
}
```

```
in vec3 vReflectVector;
out vec4 fFragColor;
uniform samplerCube uReflectUnit;

void main( )
{
    vec4 newcolor = textureCube( uReflectUnit, vReflectVector );
    fFragColor = newcolor;
}
```



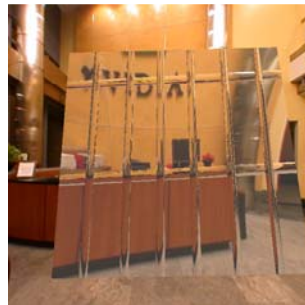
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Using the Cube Map for Refraction



refract.glib



Oregon State
Computer

Brown Cunningham
Associates

Using the Cube Map for Refraction



```

out vec3 vRefractVector;
out vec3 vReflectVector;
uniform float uEta;

void main( )
{
    vec3 ECposition = vec3( uModelViewMatrix * aVertex );
    vec3 eyeDir = normalize( ECposition ) - vec3(0.,0.,0.);           // vector from eye to pt
    vec3 normal = normalize( uNormalMatrix * aNormal );
    vRefractVector = refract( eyeDir, normal, uEta );
    vReflectVector = reflect( eyeDir, normal );
    gl_Position = uModelViewProjectionMatrix * aVertex;
}

```

```

in vec3 vReflectVector;
in vec3 vRefractVector;
out vec4 fFragColor;
uniform float uMix;
uniform samplerCube uReflectUnit;
uniform samplerCube uRefractUnit;
const vec4 WHITE = vec4( 1.,1.,1.,1. );

void main( )
{
    vec4 refractcolor = textureCube( uRefractUnit, vRefractVector );
    vec4 reflectcolor = textureCube( uReflectUnit, vReflectVector );
    refractcolor = mix( refractcolor, WHITE, .3 );
    fFragColor = mix( refractcolor, reflectcolor, uMix );
}

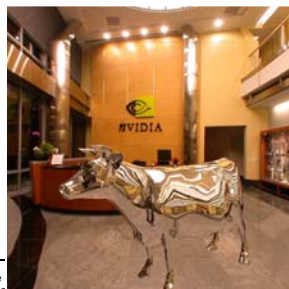
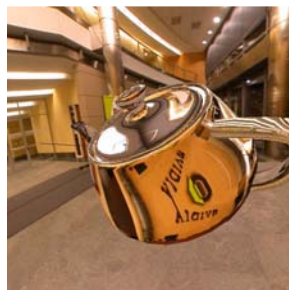
```



Oregon State
Computer G

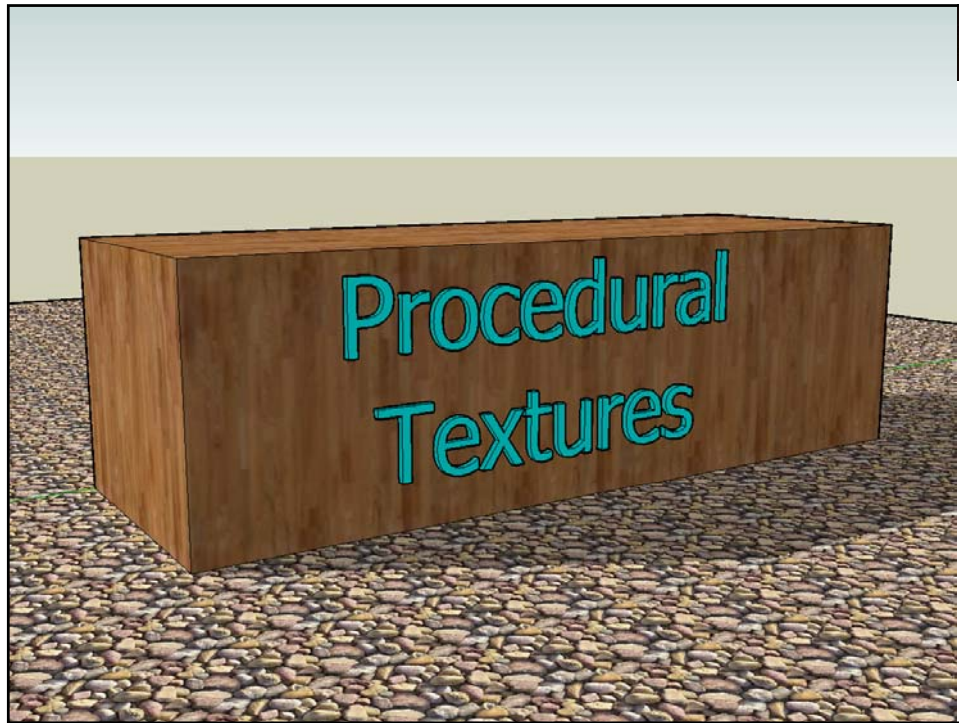
Brown Cunningham
Associates

A Comparison of Reflection and Refraction

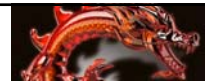


Oregon State
Computer G

Brown Cunningham
Associates



What if you want multi-colored stripes?



rainbow.glib



Tol = 0.

And, what if you want the stripes to smoothly blend into each other?



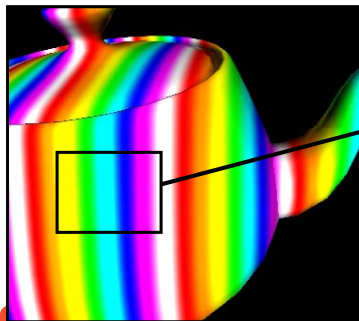
Tol > 0.

What if you want multi-colored stripes?



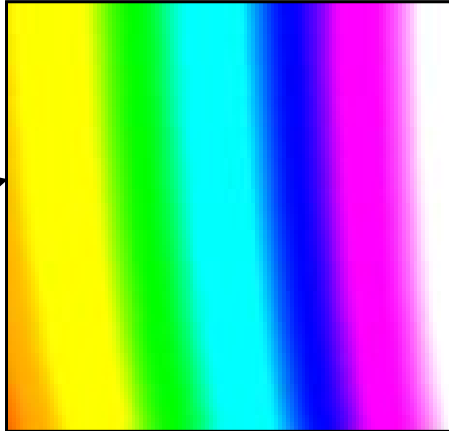
This is a good example of a *Procedural Texture*. It is like a texture that is read from a file, but instead is computed as the display is being created.

Procedural Textures are very popular because (1) you can do some amazing things with them, and (2) they don't "run out of texels" like a fixed-size texture would.



OSU

Oregon State University
Computer Graphics



Here's how to do the Colored Stripes



frag file, I

```
in vec3 vMCposition;
in float vLightIntensity;
out vec4 fFragColor;

uniform float uA;
uniform float uTol;

const vec4 RED      = vec4( 1., 0., 0., 1. );
const vec4 ORANGE   = vec4( 1., .5, 0., 1. );
const vec4 YELLOW    = vec4( 1., 1., 0., 1. );
const vec4 GREEN     = vec4( 0., 1., 0., 1. );
const vec4 CYAN      = vec4( 0., 1., 1., 1. );
const vec4 BLUE       = vec4( 0., 0., 1., 1. );
const vec4 MAGENTA   = vec4( 1., 0., 1., 1. );
const vec4 WHITE      = vec4( 1., 1., 1., 1. );

const float ONE16    = 1./16.;
const float THREE16  = 3./16.;
const float FIVE16   = 5./16.;
const float SEVEN16  = 7./16.;
const float NINE16   = 9./16.;
const float ELEVEN16 = 11./16.;
const float THIRTEEN16 = 13./16.;
const float FIFTEEN16 = 15./16.;
```

OSU

Oregon State University
Computer Graphics

Brown, Cunningham
Associates

frag file, II

```
void
main( )
{
    float X = vMCposition.x;
    float Y = vMCposition.y;
    float f = fract( uA*X );
    float t = smoothstep( ONE16 - uTol, ONE16 + uTol, f );
    fFragColor = vLightIntensity * mix( WHITE, RED, t );

    if( f >= THREE16 - Tol )
    {
        t = smoothstep( THREE16 - uTol, THREE16 + uTol, f );
        fFragColor = vLightIntensity * mix( RED, ORANGE, t );
    }
    if( f >= FIVE16 - Tol )
    {
        t = smoothstep( FIVE16 - uTol, FIVE16 + uTol, f );
        fFragColor = vLightIntensity * mix( ORANGE, YELLOW, t );
    }

    ...
}
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates



Image Negative



image.glib

```
in vec2 vST;
out vec4 fFragColor;
uniform sampler2D ulmageUnit;
uniform float uT;

void main( )
{
    vec2 st = vST;
    vec3 irgb = texture( ulmageUnit, st ).rgb;
    vec3 neg = vec3( 1.,1.,1. ) - irgb;
    fFragColor = vec4( mix( irgb, neg, uT ), 1. );
}
```



Oregon State University
Computer Graphics

am

Image Un-Masking:

Sometimes it's easier to ask for what you *don't* want
than asking for what you *do* want !



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



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Brightness



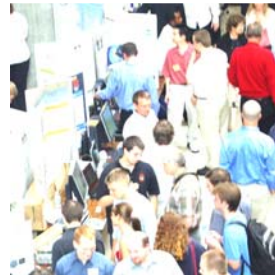
```
ldontwant = vec3( 0., 0., 0. );
```



T = 0.



T = 1.



T = 2.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Contrast



```
ldontwant = vec3( 0.5, 0.5, 0.5 );
```



T = 0.



T = 1.



T = 2.



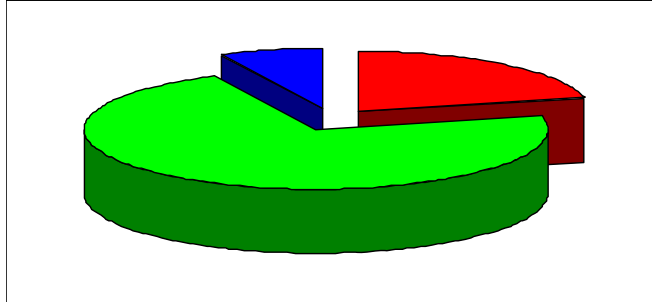
Oregon State University
Computer Graphics

Brown Cunningham
Associates

HDTV Luminance Standard



$$\text{Luminance} = 0.2125 \cdot \text{Red} + 0.7154 \cdot \text{Green} + 0.0721 \cdot \text{Blue}$$



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Saturation



$$\text{I_dontwant} = \text{vec3}(\text{luminance}, \text{luminance}, \text{luminance});$$



T = 0.



T = 1.



T = 3.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Blur



Blur Convolution:

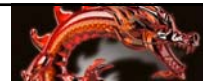
$$B = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Sharpening



Blur Convolution:

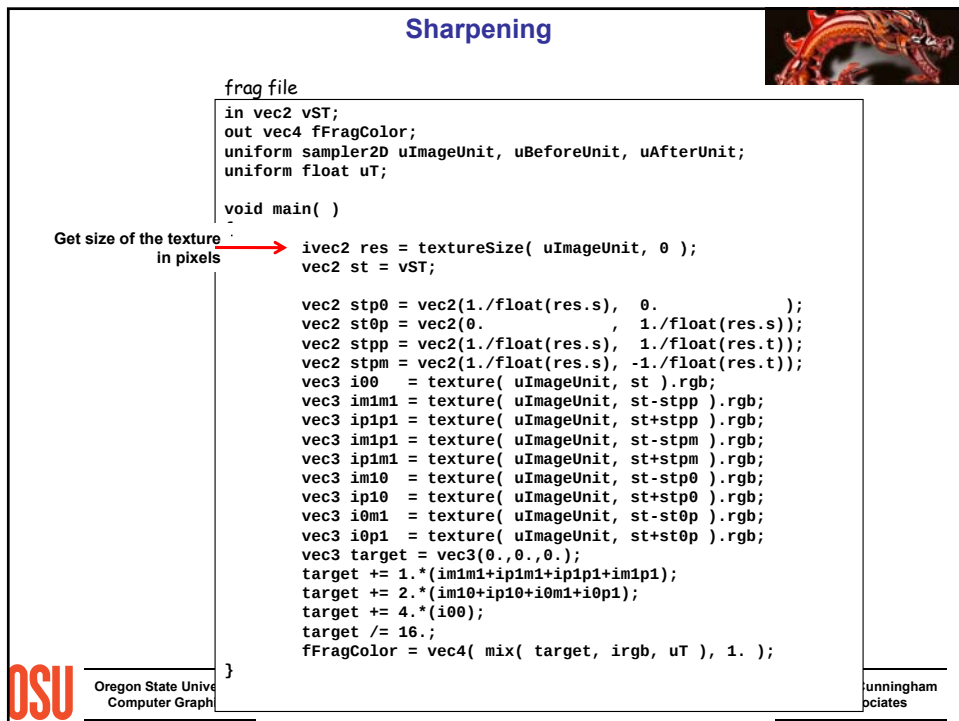
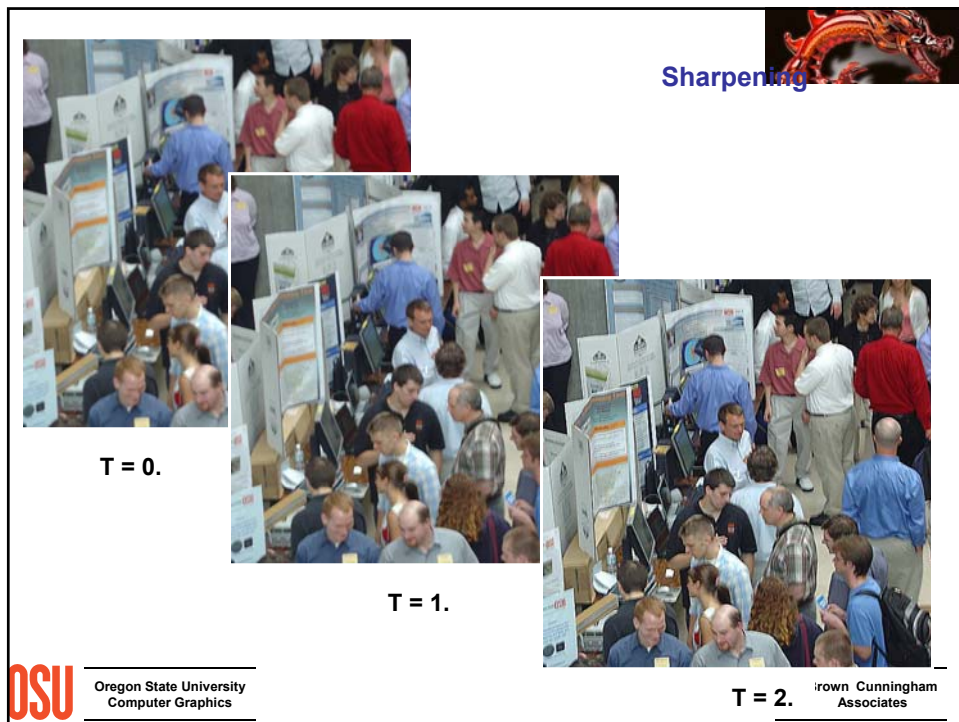
$$B = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$I_{\text{dontwant}} = I_{\text{blur}}$$



Oregon State University
Computer Graphics

Brown Cunningham
Associates



Edge Detection



Horizontal and Vertical Sobel Convolutions:

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

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

$$S = \sqrt{H^2 + V^2}$$

$$\Theta = \text{atan2}(V, H)$$



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Edge Detection



```
vec2 stp0 = vec2(1./float(res.s), 0.);
vec2 st0p = vec2(0., 1./float(res.s));
vec2 stpp = vec2(1./float(res.s), 1./float(res.t));
vec2 stpm = vec2(1./float(res.s), -1./float(res.t));

float i00 = dot( texture( uImageUnit, st ).rgb, vec3(0.2125,0.7154,0.0721) );
float im1m1 = dot( texture( uImageUnit, st-stpp ).rgb, vec3(0.2125,0.7154,0.0721) );
float ip1p1 = dot( texture( uImageUnit, st+stpp ).rgb, vec3(0.2125,0.7154,0.0721) );
float im1p1 = dot( texture( uImageUnit, st-stpm ).rgb, vec3(0.2125,0.7154,0.0721) );
float ip1m1 = dot( texture( uImageUnit, st+stpm ).rgb, vec3(0.2125,0.7154,0.0721) );
float im10 = dot( texture( uImageUnit, st-stp0 ).rgb, vec3(0.2125,0.7154,0.0721) );
float ip10 = dot( texture( uImageUnit, st+stp0 ).rgb, vec3(0.2125,0.7154,0.0721) );
float i0m1 = dot( texture( uImageUnit, st-st0p ).rgb, vec3(0.2125,0.7154,0.0721) );
float i0p1 = dot( texture( uImageUnit, st+st0p ).rgb, vec3(0.2125,0.7154,0.0721) );
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, uT ), 1. );
```



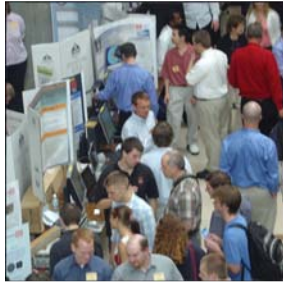
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Edge Detection



edge.glib



$T = 0.$



$T = 0.5$



$T = 1.$



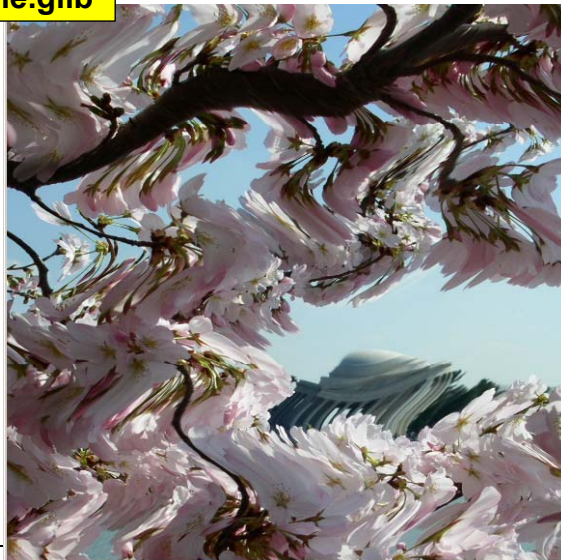
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Bonus Demo



imageripple.glib



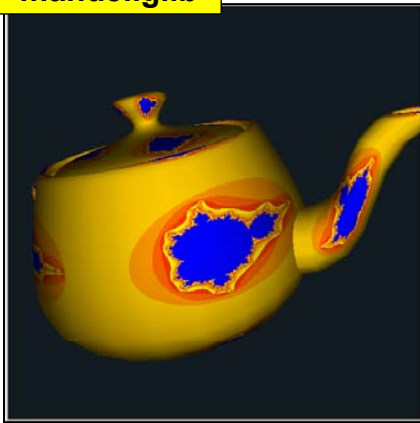
Oregon State University
Computer Graphics

Brown Cunningham
Associates

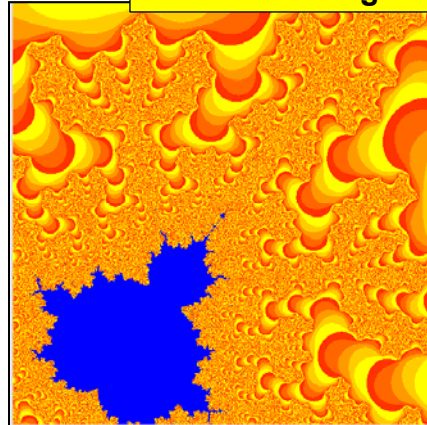
Bonus Demos



mandel.glib



mandelzoom.glib



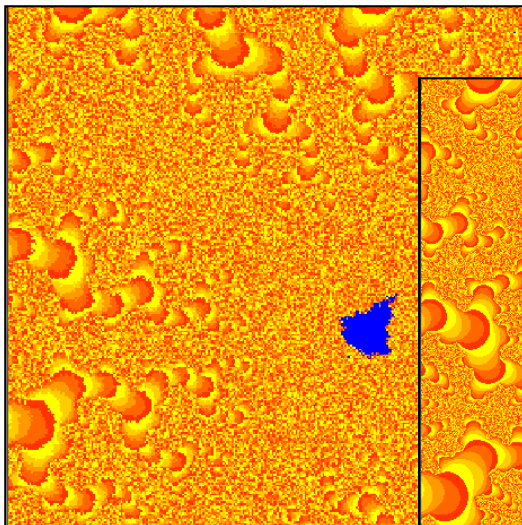
OSU

Oregon State University
Computer Graphics

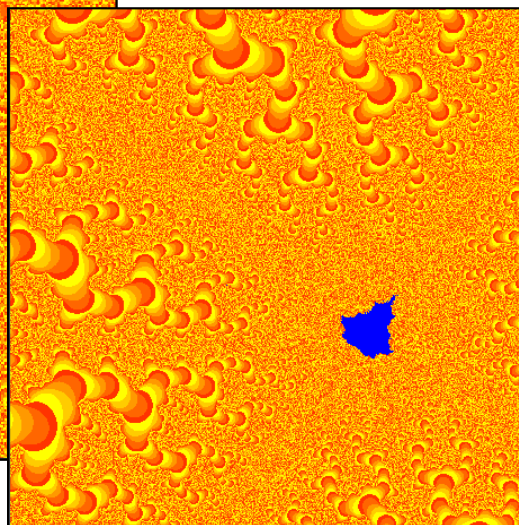
Using Double Precision in a Shader



float



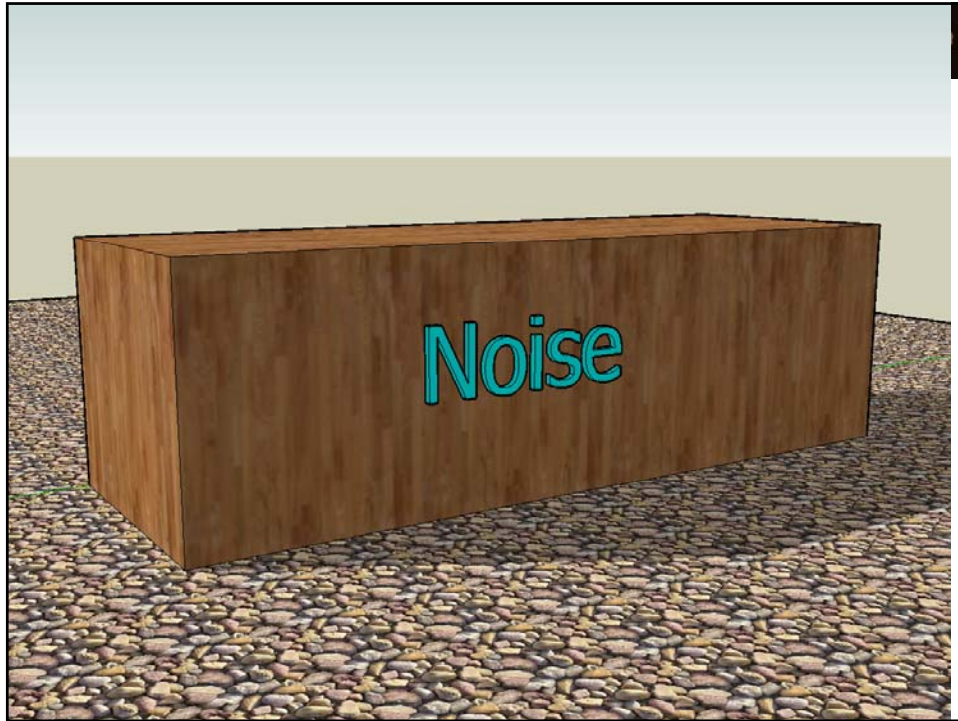
double



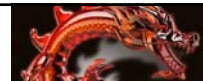
OSU

Oregon State University
Computer Graphics

Brown-Cunningham
Associates



Noise:

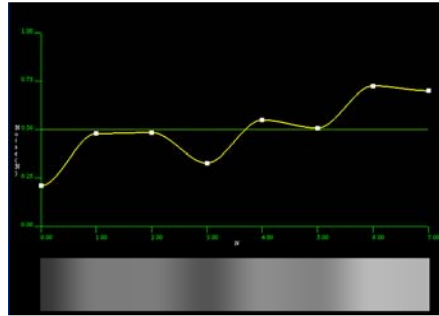
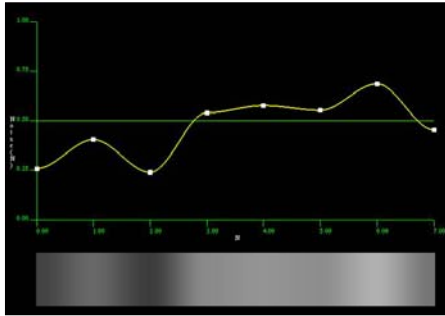


- Is a function of input value(s)
- Ranges from -1. to +1. or from 0. to 1.
- Looks random, but really isn't
- Has continuity
- Is repeatable
- Has statistical properties that are translational and rotational invariant

Positional Noise: Two sets of random numbers



noisegraph.exe



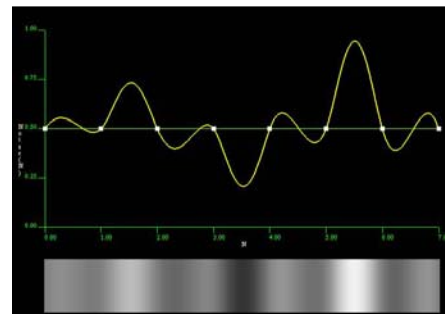
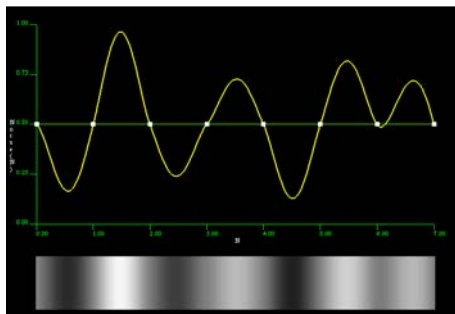
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Gradient Noise: Two sets of random numbers



noisegraph.exe



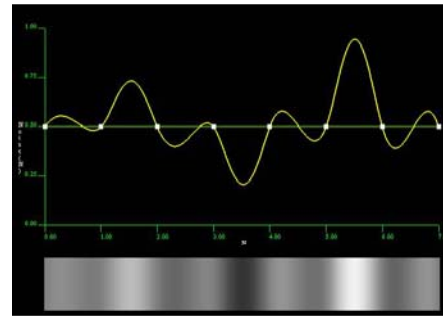
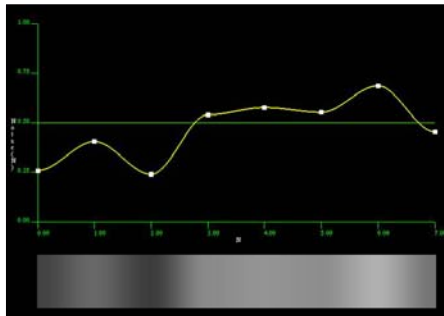
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Positional vs. Gradient Noise: Gradient has more variation



noisegraph.exe



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Coefficients for Noise



$$N(t) = C_{N0}N_0 + C_{N1}N_1 + C_{G0}G_0 + C_{G1}G_1$$

Noise values

Gradients

$$C_{N0} = 1 - 3t^2 + 2t^3$$

$$C_{N1} = 3t^2 - 2t^3 = 1 - C_{N0}$$

$$C_{G0} = t - 2t^2 + t^3$$

$$C_{G1} = -t^2 + t^3$$



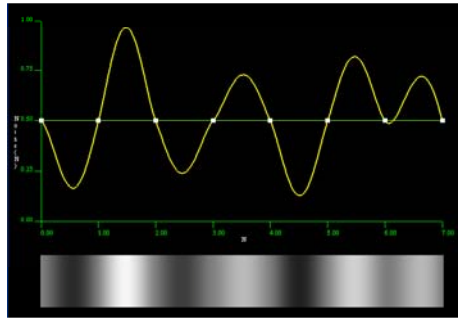
Oregon State University
Computer Graphics

Brown Cunningham
Associates

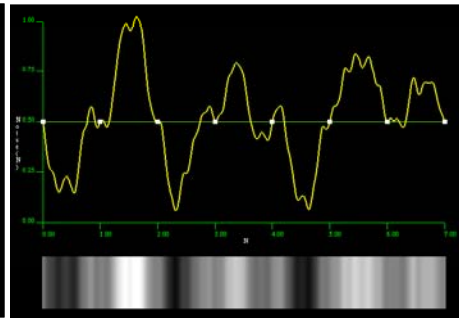
Noise Octaves



Idea: Add multiple noise waves, each one twice the frequency and half the amplitude of the previous one



1 Octave



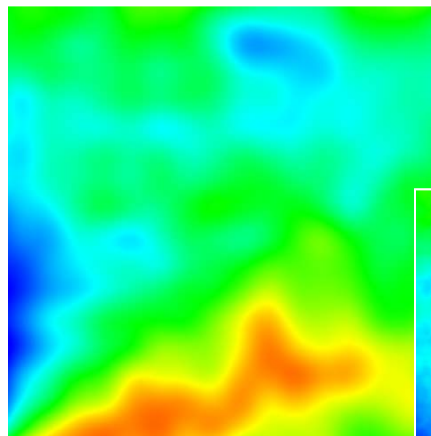
4 Octaves



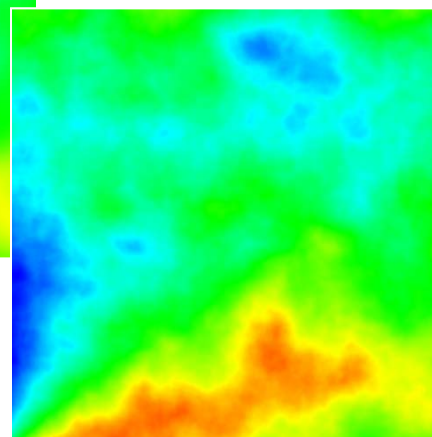
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Image Representation of 2D Noise



1 Octave



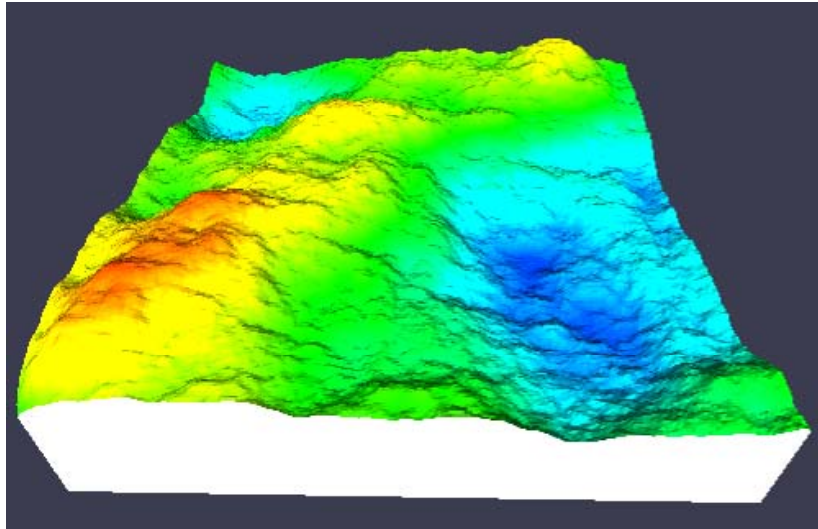
4 Octaves



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Surface Representation of 2D Noise



OSU

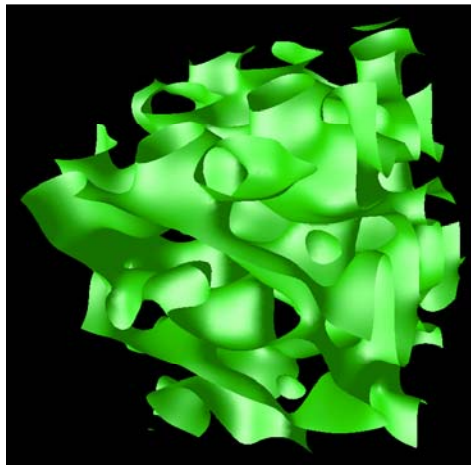
Oregon State University
Computer Graphics

4 Octaves gham
Associates

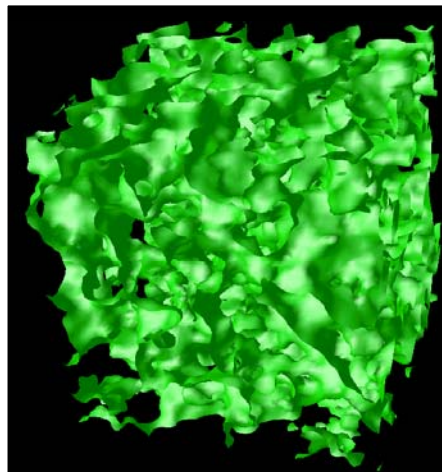
3D Isosurfaces of 3D Noise



S^* = Mid-value



1 Octave



4 Octaves

OSU

Oregon State University
Computer Graphics

Built-In GLSL Noise Functions



<code>float noise1(genType x)</code>	Returns a 1D noise value based on the input value <i>x</i> . At this time, this function is not available in GLSL.
<code>vec2 noise2(genType x)</code>	Returns a 2D noise value based on the input value <i>x</i> . At this time, this function is not available in GLSL.
<code>vec3 noise3(genType x)</code>	Returns a 3D noise value based on the input value <i>x</i> .
<code>vec4 noise4(genType x)</code>	Returns a 4D noise value based on the input value <i>x</i> .

Note: as of this writing, these functions don't work on all graphics systems!
To compensate, *glman* has a built-in noise texture.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

glman has a built-in 3D Noise Texture



glman automatically creates a 3D noise texture and places it into Texture Unit 3.

Your vertex, geometry, or fragment shader can get at it through the pre-created uniform variable called **Noise3**.

You can reference it in your shader as:

```
uniform sampler3D Noise3;  
...  
vec3 stp = ...  
vec4 nv = texture( Noise3, stp );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

glman has a built-in 3D Noise Texture



The noise texture is a vec4 whose components have separate meanings.

The [0] component is the low frequency noise.

The [1] component is twice the frequency and half the amplitude of the [0] component, and so on for the [2] and [3] components.

Each component is centered around a value of .5, so that if you want a plus-or-minus effect, subtract .5 from each component. To get a nice four-octave noise value between 0 and 1 (useful for noisy mixing), add up all four components, subtract 1 and divide the result by 2, as shown in the following table and GLSL code:.

Component	Term	Term Range
0	nv.r	$0.5 \pm .5000$
1	nv.g	$0.5 \pm .2500$
2	nv.b	$0.5 \pm .1250$
3	nv.a	$0.5 \pm .0625$
	sum	$2.0 \pm \sim 1.0$
	sum - 1	$1.0 \pm \sim 1.0$
	(sum - 1) / 2	$0.5 \pm \sim 0.5$

```
float sum = nv.r + nv.g + nv.b + nv.a;
sum = ( sum - 1. ) / 2.;
```

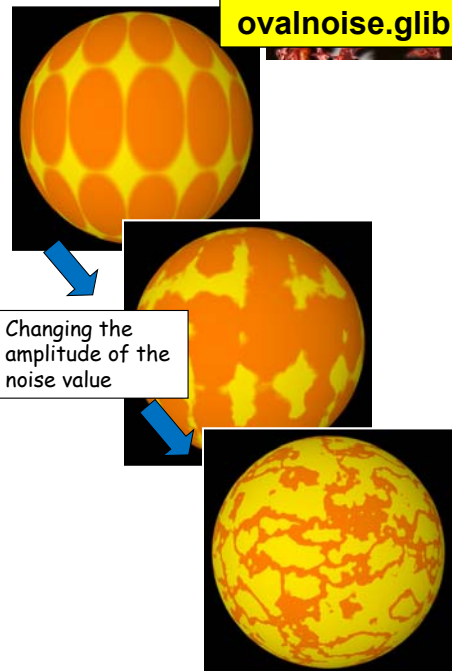
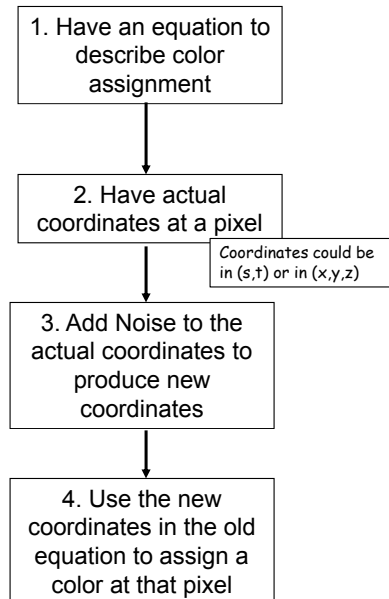
```
// range is 1. -> 3.
// range is now 0. -> 1.
```



Oregon State University
Computer Graphics

Associates

How to Apply Noise



Oregon State University
Computer Graphics

Associates

frag file, I



```
in vec3 vMCposition;           // model coord position from the vertex shader
in float vLightIntensity;      // light intensity from the vertex shader
in vec2 vST;                   // texture coords from the vertex shader

out vec4 fFragColor;

uniform float uAd;
uniform float uBd;
uniform float uNoiseAmp;
uniform float uNoiseFreq;
uniform float uAlpha;
uniform float uTol;
uniform float uBlend;
uniform sampler3D Noise3;

const vec3 ORANGE = vec3( 1., .5, 0. );
const vec3 YELLOW = vec3( 1., .9, 0. );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

frag file, II



```
void
main( )
{
    vec4 noisevec = texture( Noise3, uNoiseFreq*vMCposition );
    float n = noisevec.r + noisevec.g + noisevec.b + noisevec.a;    // 1. -> 3.
    n = ( n - 2. );          // -1. -> 1.
    n *= uNoiseAmp;

    vec2 st = vST;
    st.s *= 2.;

    float Ar = uAd / 2.;
    float Br = uBd / 2.;

    int numinu = int( st.s / uAd );
    int numinv = int( st.t / uBd );

    vec3 theColor = YELLOW;

    st.s -= float(numinu) * uAd;
    st.t -= float(numinv) * uBd;
    vec3 upvp = vec3( st, 0. );
    vec3 cntr = vec3( Ar, Br, 0. );
    vec3 delta = upvp - cntr;
    float oldrad = length( delta );
    float newrad = oldrad + n;
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

frag file, III

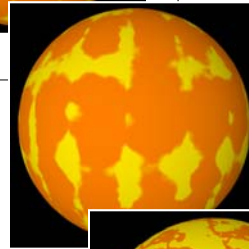
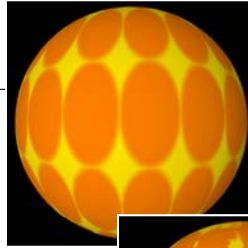
```

delta = delta * newrad / oldrad;
float du = delta.x/Ar;
float dv = delta.y/Br;
float d = du*du + dv*dv;

float t = smoothstep( 1.-uTol, 1.+uTol, d );
theColor = mix( ORANGE, YELLOW, t );

fFragColor = vec4( vLightIntensity*theColor, 1. );
}

```

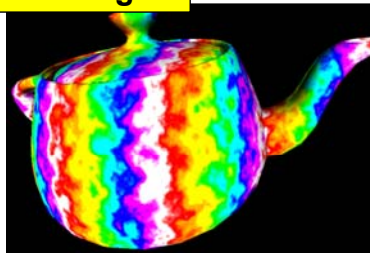


Oregon State University
Computer Graphics

ASSOCIATES

Noise Examples

rainbow.glib



More Interesting Stripe Blending

fire.glib



Fire Effect

clouds.glib



Cloud Effect

eroded.glib

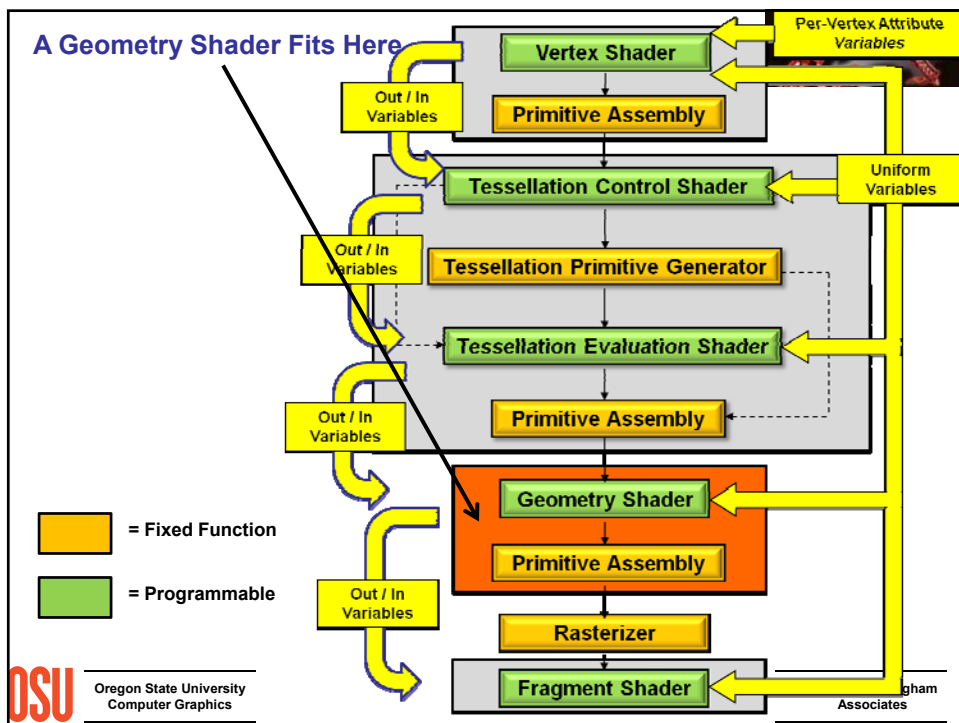
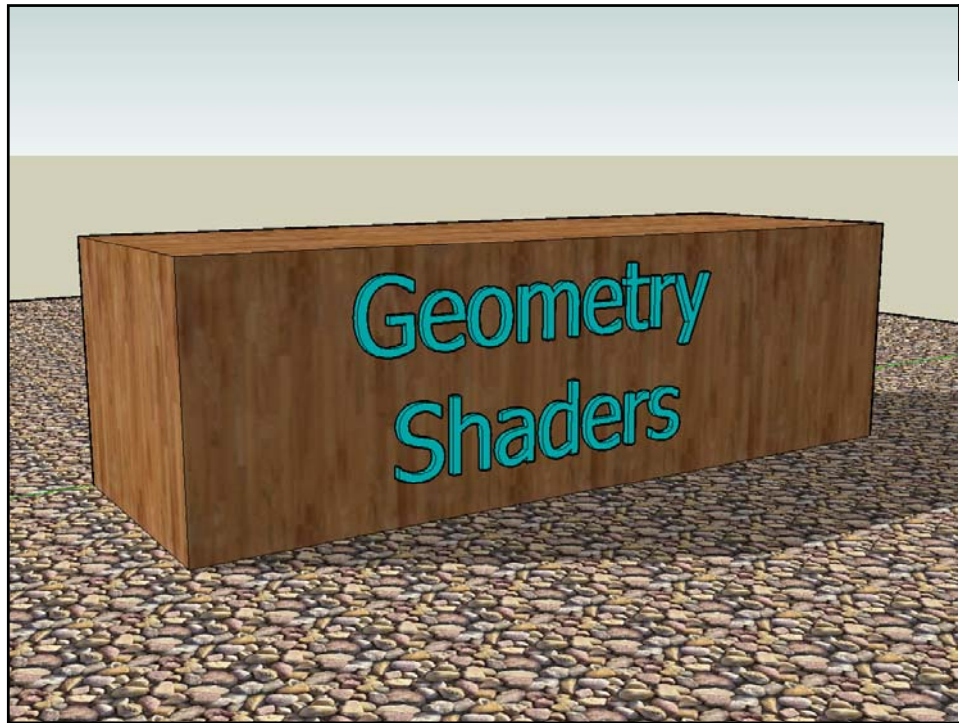


Deciding when to Discard
for Erosion

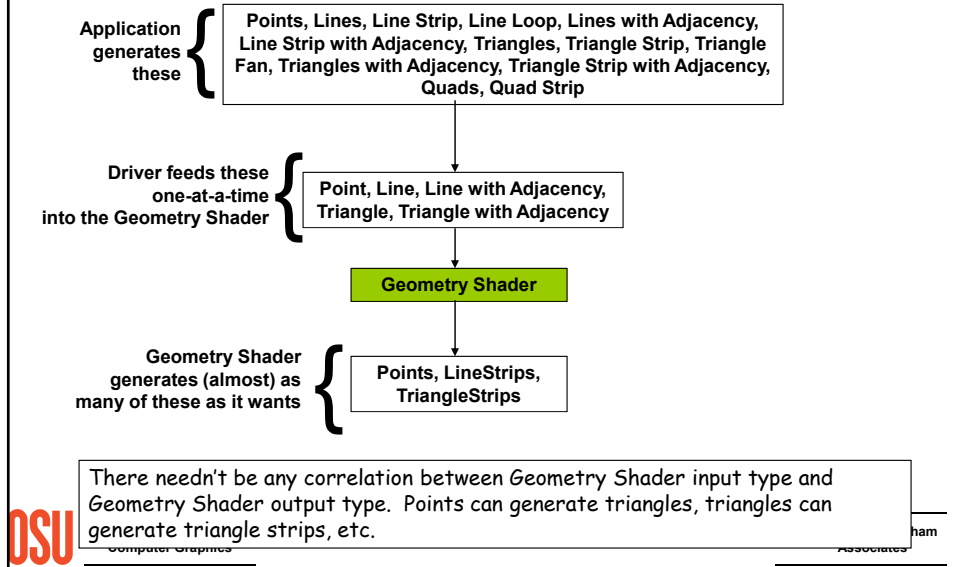
wn Cunningham
Associates



Oregon
State



The Geometry Shader: What Does it Do?



Additional Arguments are Available for glBegin() – glEnd():



GL_LINES_ADJACENCY

GL_LINE_STRIP_ADJACENCY

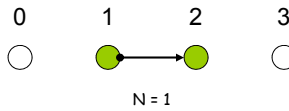
GL_TRIANGLES_ADJACENCY

GL_TRIANGLE_STRIP_ADJECENCY

New Adjacency Primitives

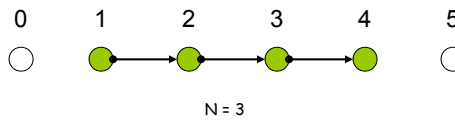


Lines with Adjacency



4N vertices are given.
(where N is the number of line segments to draw).
A line segment is drawn between #1 and #2.
Vertices #0 and #3 are there to provide adjacency information.

Line Strip with Adjacency



N+3 vertices are given
(where N is the number of line segments to draw).
A line segment is drawn between #1 and #2, #2 and #3, ..., #N and #N+1.
Vertices #0 and #N+2 are there to provide adjacency information.



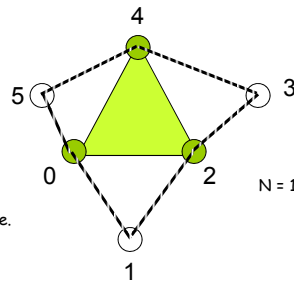
Oregon State University
Computer Graphics

Brown Cunningham
Associates

New Adjacency Primitives

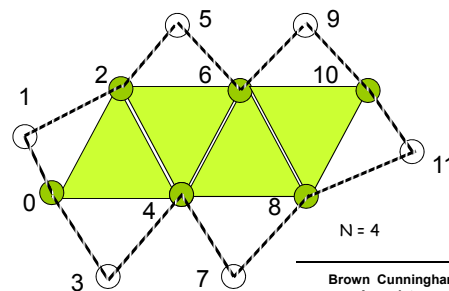


Triangles with Adjacency



6N vertices are given
(where N is the number of triangles to draw).
Points 0, 2, and 4 define the triangle.
Points 1, 3, and 5 tell where adjacent triangles are.

Triangle Strip with Adjacency



4+2N vertices are given
(where N is the number of triangles to draw).
Points 0, 2, 4, 6, 8, 10, ... define the triangles.
Points 1, 3, 5, 7, 9, 11, ... tell where adjacent triangles are.



Oregon State University
Computer Graphics

Brown Cunningham
Associates



If a Vertex Shader writes variables as: then the Geometry Shader will read them as: and will write them as:

gl_Position	→	gl_PositionIn[■]	→	gl_Position
gl_PointSize	→	gl_PointSizeIn	→	gl_PointSize
vST	→	vST[■]	→	gST
vColor	→	vColor[■]	→	gColor
"out"		"in"		"out"

In the Geometry Shader, the dimensions indicated by ■ are given by the variable *gl_VerticesIn*, although you will already know this by the type of geometry you are inputting

1	GL_POINTS
2	GL_LINES
4	GL_LINES_ADJACENCY_EXT
3	GL_TRIANGLES
6	GL_TRIANGLES_ADJACENCY_EXT



Oregon State University
Computer Graphics

Brown Cunningham
Associates



The Geometry Shader Can Assign These Variables:

- gl_Position
- User-defined

When the Geometry Shader calls

EmitVertex()

this set of variables is copied to a slot in the shader's Primitive Assembly step

When the Geometry Shader calls

EndPrimitive()

the vertices that have been saved in the Primitive Assembly step are then assembled, rasterized, etc.

Note: there is no "BeginPrimitive()" routine. It is implied by (1) the start of the Geometry Shader, or (2) returning from the EndPrimitive() call.

Note: there is no need to call EndPrimitive() at the end of the Geometry Shader - it is implied.



Computer Graphics

Brown Cunningham
Associates

Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments



bezier.glib

```
Gstap

Vertex    bezier.vert
Geometry  bezier.geom
Fragment  bezier.frag
Program   Bezier uNum <2 10 50>

LineWidth 3.
LinesAdjacency [0. 0. 0.] [1. 1. 1.] [2. 1. 2.] [3. -1. 0.]
```

bezier.vert

```
void main( )
{
    gl_Position = uModelViewProjectionMatrix * aVertex;
}
```

bezier.frag

```
out vec4 fFragColor;

void main( )
{
    fFragColor = vec4( 0., 1., 0., 1. );
}
```



Oregon State
Computer G

Brown Cunningham
Associates

Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments



bezier.geom

```
#version 120
#extension GL_EXT_geometry_shader4: enable
layout( lines_adjacency ) in;
layout( lines, max_vertices=128 ) out;
uniform int uNum;
void main( )
{
    float dt = 1. / float(uNum);
    float t = 0.;
    for( int i = 0; i <= uNum; i++ )
    {
        float omt = 1. - t;
        float omt2 = omt * omt;
        float omt3 = omt * omt2;
        float t2 = t * t;
        float t3 = t * t2;
        vec4 xyzw =

            omt3 * gl_PositionIn[0].xyzw +
            3. * t * omt2 * gl_PositionIn[1].xyzw +
            3. * t2 * omt * gl_PositionIn[2].xyzw +
            t3 * gl_PositionIn[3].xyzw;

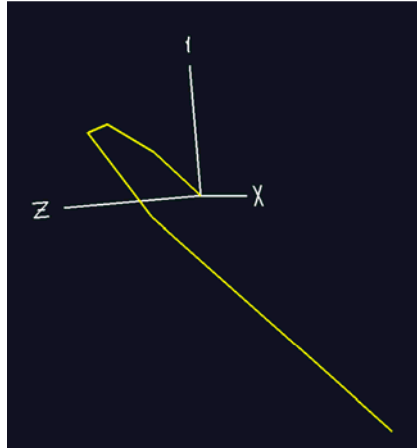
        gl_Position = xyzw;
        EmitVertex();
        t += dt;
    }
}
```

Used to declare the geometry shader's
input and output topology

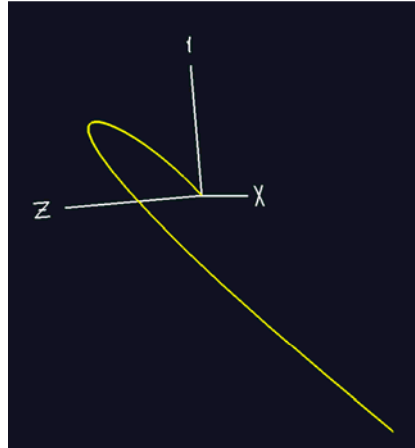
$$P(t) = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t) P_2 + t^3 P_3$$



Example: Expanding 4 Points into a Bezier Curve with a Variable Number of Line Segments



uNum = 5



uNum = 25



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Shrinking Triangles



shrink.glib



Oregon State University
Computer Graphics

shrink.geom

```
#version 120
#extension GL_EXT_geometry_shader4: enable
layout( triangles ) in;
layout( triangle_strip, max_vertices=32 ) out;

uniform float uShrink;
in vec3 vNormal[ ];
out float gLightIntensity;

const vec3 LIGHTPOS = vec3( 0., 10., 0. );
vec3 V[3];
vec3 CG;

void
ProduceVertex( int v )
{
    gLightIntensity = dot( normalize( LIGHTPOS - V[v] ), vNormal[v] );
    gLightIntensity = abs( gLightIntensity );

    gl_Position = uModelViewProjectionMatrix * vec4( CG + uShrink * ( V[v] - CG ), 1. );
    EmitVertex();
}

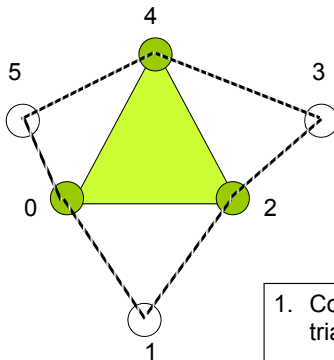
void
main( )
{
    V[0] = gl_PositionIn[0].xyz;
    V[1] = gl_PositionIn[1].xyz;
    V[2] = gl_PositionIn[2].xyz;
    CG = ( V[0] + V[1] + V[2] ) / 3.;
    ProduceVertex( 0 );
    ProduceVertex( 1 );
    ProduceVertex( 2 );
}
```



Oregon
Comp

ningham
ates

Example: Silhouette Geometry Shader



1. Compute the normals of each of the four triangles
2. If there is a sign difference between the z component of the center triangle and the z component of an adjacent triangle, draw their common edge



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Silhouette Geometry Shader



silh.glib

```
Obj bunny.obj  
  
Vertex    silh.vert  
Geometry  silh.geom  
Fragment  silh.frag  
Program Silhouette uColor { 0. 1. 0. }  
ObjAdj    bunny.obj
```

This creates triangle-adjacency
information when the file is read



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Silhouette Geometry Shader



silh.vert

```
void main( )  
{  
    gl_Position = uModelViewMatrix * aVertex;  
}
```

silh.frag

```
uniform vec4 uColor;  
out vec4    fFragColor;  
  
void  
main( )  
{  
    fFragColor = vec4( uColor.rgb, 1. );  
}
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Silhouette Geometry Shader



silh.geom, I

```
#version 120
#extension GL_EXT_geometry_shader4: enable

layout( triangles_adjacency ) in;
layout( line_strip, max_vertices=32 ) out;

void
main()
{
    vec3 V0 = gl_PositionIn[0].xyz;
    vec3 V1 = gl_PositionIn[1].xyz;
    vec3 V2 = gl_PositionIn[2].xyz;
    vec3 V3 = gl_PositionIn[3].xyz;
    vec3 V4 = gl_PositionIn[4].xyz;
    vec3 V5 = gl_PositionIn[5].xyz;

    vec3 N042 = cross( V4-V0, V2-V0 );
    vec3 N021 = cross( V2-V0, V1-V0 );
    vec3 N243 = cross( V4-V2, V3-V2 );
    vec3 N405 = cross( V0-V4, V5-V4 );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Silhouette Geometry Shader



silh.geom, II

```
    if( N042.z * N021.z < 0. )
    {
        gl_Position = uProjectionMatrix * vec4( V0, 1. );
        EmitVertex( );
        gl_Position = uProjectionMatrix * vec4( V2, 1. );
        EmitVertex( );
        EndPrimitive( );
    }

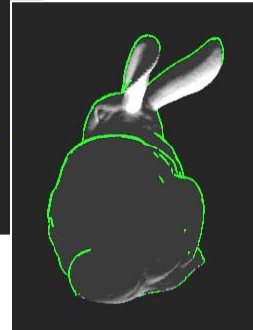
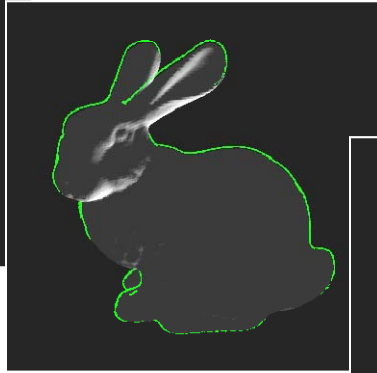
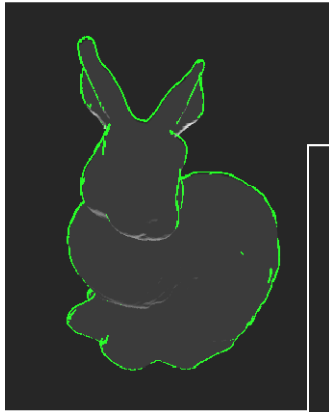
    if( N042.z * N243.z < 0. )
    {
        gl_Position = uProjectionMatrix * vec4( V2, 1. );
        EmitVertex( );
        gl_Position = uProjectionMatrix * vec4( V4, 1. );
        EmitVertex( );
        EndPrimitive( );
    }

    if( N042.z * N405.z < 0. )
    {
        gl_Position = uProjectionMatrix * vec4( V4, 1. );
        EmitVertex( );
        gl_Position = uProjectionMatrix * vec4( V0, 1. );
        EmitVertex( );
        EndPrimitive( );
    }
}
```

Example: Bunny Silhouettes



silh.glib

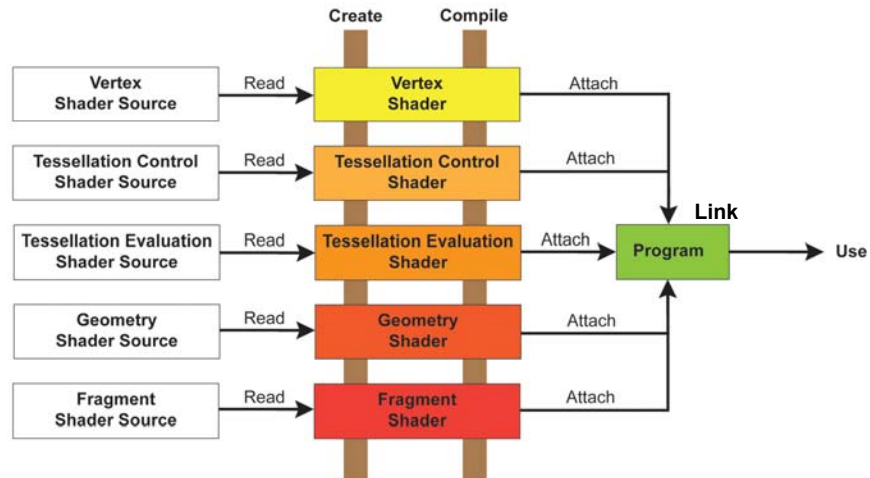


Oregon State University
Computer Graphics

Brown Cunningham
Associates



The GLSL Shader-creation Process



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Initializing the GL Extension Wrangler (GLEW)



```

#include "glew.h"

...

GLenum err = glewInit( );
if( err != GLEW_OK )
{
    fprintf( stderr, "glewInit Error\n" );
    exit( 1 );
}

fprintf( stderr, "GLEW initialized OK\n" );
fprintf( stderr, "Status: Using GLEW %s\n", glewGetString(GLEW_VERSION) );
  
```

<http://glew.sourceforge.net>



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Reading a Vertex, Tessellation, Geometry, or Fragment Shader source file into a character buffer



```
#include <stdio.h>

FILE *fp = fopen( filename, "r" );
if( fp == NULL ) { . . . }

fseek( fp, 0, SEEK_END );
int numBytes = ftell( fp ); // length of file

GLchar * buffer = new GLchar [numBytes+1];

rewind( fp ); // same as: "fseek( in, 0, SEEK_SET )"
fread( buffer, 1, numBytes, fp );
fclose( fp );
buffer[numBytes] = '\0'; // the entire file is now in a byte string
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Creating and Compiling a Vertex Shader from that character buffer (Tessellation, Geometry, and Fragment files work the same way)



```
int status;
int logLength;

GLuint vertShader = glCreateShader( aVertex_SHADER );

glShaderSource( vertShader, 1, (const GLchar **)&buffer, NULL );
delete [ ] buffer;
glCompileShader( vertShader );
CheckGLErrors( "Vertex Shader 1" );

glGetShaderiv( vertShader, GL_COMPILE_STATUS, &status );
if( status == GL_FALSE )
{
    fprintf( stderr, "Vertex shader compilation failed.\n" );
    glGetShaderiv( vertShader, GL_INFO_LOG_LENGTH, &logLength );
    GLchar * log = new GLchar [logLength];
    glGetShaderInfoLog( vertShader, logLength, NULL, log );
    fprintf( stderr, "\n%s\n", log );
    delete [ ] log;
    exit( 1 );
}
CheckGLErrors( "Vertex Shader 2" );
```



Computer Graphics

Associates

How does that array of strings thing work?



```
GLchar * ArrayOfStrings[3];
ArrayOfStrings[0] = "#define SMOOTH_SHADING";
ArrayOfStrings[1] = "... a commonly-used procedure ... ";
ArrayOfStrings[2] = "... the real vertex shader code ... ";
glShaderSource( vertShader, 3, ArrayofStrings, NULL );
```

These are two ways to provide a *single* character buffer:

```
GLchar *buffer[1];
buffer[0] = "... the entire shader code ... ";
glShaderSource( vertShader, 1, buffer, NULL );
```

```
GLchar *buffer = "... the entire shader code ... ";
glShaderSource( vertShader, 1, (const GLchar **)&buffer, NULL );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Why use an array of strings as the shader input, instead of just a single string?



- You can use the same shader source and insert the appropriate #defines at the beginning
- You can insert a common header file (≈ a .h file)
- You can simulate a #include to re-use common pieces of code

If-tests versus preprocessing

```
if( Mode == PerVertexShading )
{ ... }
else if( Mode == PerFragmentShading )
{ ... }
```

```
#ifdef PER_VERTEX_SHADING
{ ... }
#endif

#ifdef PER_FRAGMENT_SHADING
{ ... }
#endif
```



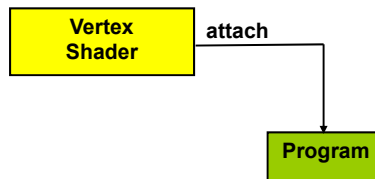
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Creating the Program and Attaching the Shaders to It



```
GLuint program = glCreateProgram( );  
glAttachShader( program, vertShader );  
glAttachShader( program, tessControlShader );  
glAttachShader( program, tessEvaluationShader );  
glAttachShader( program, geomShader );  
glAttachShader( program, fragShader );
```



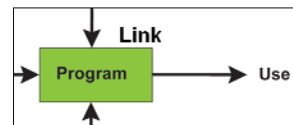
Oregon State University
Computer Graphics

Brown Cunningham
Associates

Linking the Program and Checking its Validity



```
glLinkProgram( program );  
CheckGLErrors( "Shader Program 1" );  
glGetProgramiv( program, GL_LINK_STATUS, &status );  
if( status == GL_FALSE )  
{  
    fprintf( stderr, "Link failed.\n" );  
    glGetProgramiv( program, GL_INFO_LOG_LENGTH, &logLength );  
    log = new GLchar [logLength];  
    glGetProgramInfoLog( program, logLength, NULL, log );  
    fprintf( stderr, "\n%s\n", log );  
    delete [ ] log;  
    exit( 1 );  
}  
CheckGLErrors( "Shader Program 2" );  
  
glValidateProgram( program );  
glGetProgramiv( program, GL_VALIDATE_STATUS, &status );  
fprintf( stderr, "Program is %s.\n", status == GL_TRUE ? "valid" : "invalid" );
```



Oregon State University
Computer Graphics

Making the Program Active



```
glUseProgram( program );
```

This is now an "attribute", i.e., this shader combination is in effect until you change it

Making the Program Inactive (use the fixed function pipeline instead)

```
glUseProgram( 0 );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Passing in Uniform Variables



```
float lightLoc[3] = { 0., 100., 0. };  
  
GLint location = glGetUniformLocation( program, "uLightLocation" );  
  
if( location < 0 )  
    fprintf( stderr, "Cannot find Uniform variable 'uLightLocation'\n" );  
else  
    glUniform3fv( location, 3, lightLoc );
```

In the shader, this would be declared as:

```
uniform vec3 uLightLocation;
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Passing in Vertex Attribute Variables



```

Glint location = glGetAttribLocation( program, "aArray" );

if( location < 0 )
{
    fprintf( stderr, "Cannot find Attribute variable 'aArray'\n" );
}
else
{
    glBegin( GL_TRIANGLES );
    glVertexAttrib2f( location, a0, b0 );
    glVertex3f( x0, y0, z0 );

    glVertexAttrib2f( location, a1, b1 );
    glVertex3f( x1, y1, z1 );

    glVertexAttrib2f( location, a2, b2 );
    glVertex3f( x2, y2, z2 );

    glEnd( );
}
    
```

We are using the deprecated glBegin-glVertex-glEnd process here for to keep this code concise and clear.

In the vertex shader, this would be declared as:

in vec2 aArray;



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Checking for Errors



```

void
CheckGLErrors( const char* caller )
{
    unsigned int glerr = glGetError();
    if( glerr == GL_NO_ERROR )
        return;
    fprintf( stderr, "GL Error discovered from caller '%s': ", caller );
    switch( glerr )
    {
        case GL_INVALID_ENUM:
            fprintf( stderr, "Invalid enum.\n" );
            break;
        case GL_INVALID_VALUE:
            fprintf( stderr, "Invalid value.\n" );
            break;
        case GL_INVALID_OPERATION:
            fprintf( stderr, "Invalid Operation.\n" );
            break;
        case GL_STACK_OVERFLOW:
            fprintf( stderr, "Stack overflow.\n" );
            break;
        case GL_STACK_UNDERFLOW:
            fprintf( stderr, "Stack underflow.\n" );
            break;
        case GL_OUT_OF_MEMORY:
            fprintf( stderr, "Out of memory.\n" );
            break;
        case GL_INVALID_FRAMEBUFFER_OPERATION:
            fprintf( stderr, "Framebuffer object is not complete\n" );
            break;
        default:
            fprintf( stderr, "Unknown OpenGL error: %d (0x%0x)\n", glerr, glerr );
    }
}
    
```

This is not a bad idea to do all through your OpenGL programs, even without shaders!



Writing a C++ Class to Handle Everything is Fairly Straightforward



Setup:

```
int Polar;  
float K;  
GLSLProgram *Hyper = new GLSLProgram("hyper.vert", "hyper.geom", "hyper.frag" );
```

This loads, compiles, and links the shader.
It prints error messages and returns NULL if something failed.

Using the GPU program during display:

```
Hyper->Use( );  
Hyper->SetUniform( "Polar", Polar );  
Hyper->SetUniform( "K", K );
```

Reverting to the fixed-function pipeline during display:

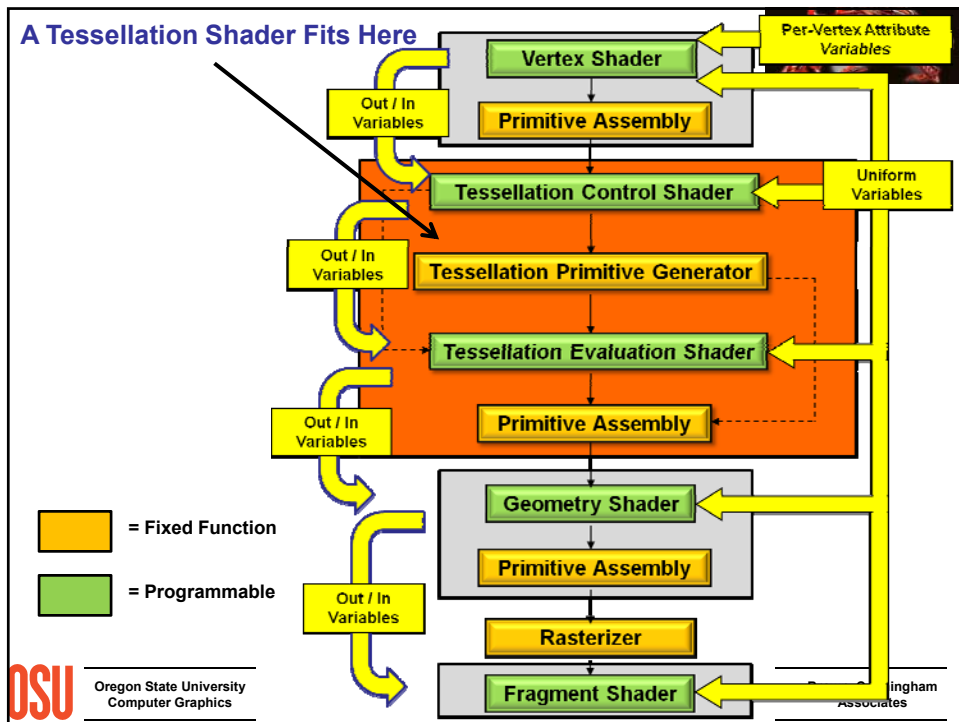
```
Hyper->Use( 0 );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates



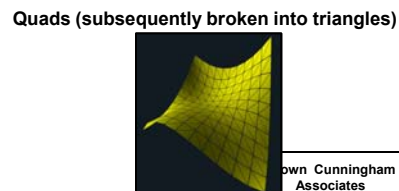
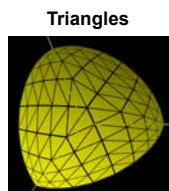
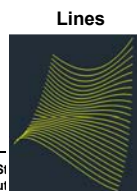


Why do we need a Tessellation step right in the pipeline?



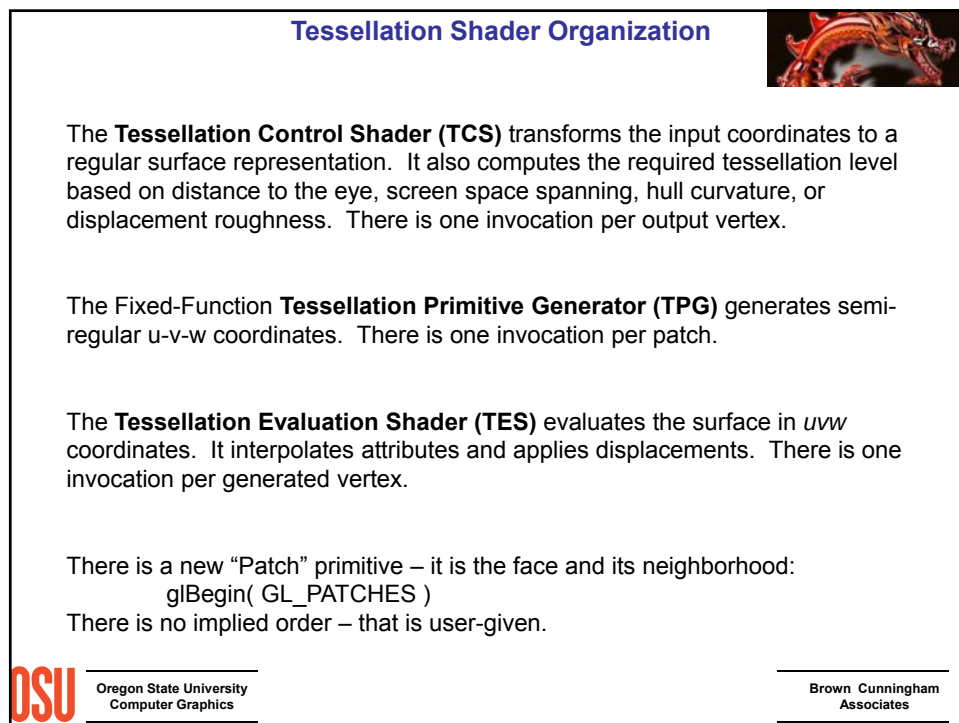
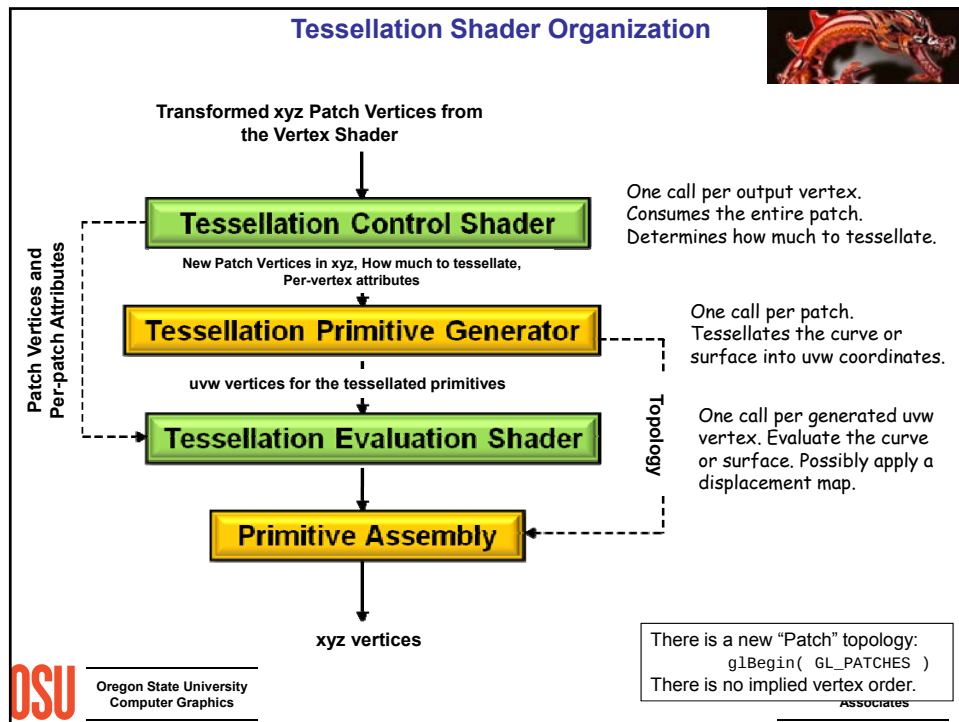
- You can perform adaptive subdivision based on a variety of criteria (size, curvature, screen extent, etc.)
- You can provide coarser models ($\approx$ geometric compression)
- You can apply detailed displacement maps without supplying equally detailed geometry
- You can adapt visual quality to the required level of detail
- You can create smoother silhouettes
- You can perform skinning easier

What patterns can Tessellation shaders use?



Oregon State University
Computer Graphics

Ingham Associates

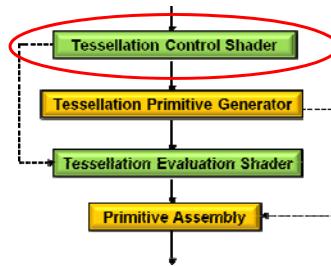


TCS Outputs



`gl_out[ ]` is an array of structures containing:

```
gl_Position;  
gl_PointSize;  
gl_ClipDistance[ ];
```



All invocations of the TCS have read-only access to all the output information. **barrier()** causes all instances of TCS's to wait here

layout(vertices = n) out; Used to specify the number of output vertices

gl_TessLevelOuter[4] is an array containing up to 4 edges of tessellation levels

gl_TessLevelInner[2] is an array containing up to 2 edges of tessellation levels



Oregon State University
Computer Graphics

Brown Cunningham
Associates

In the TCS

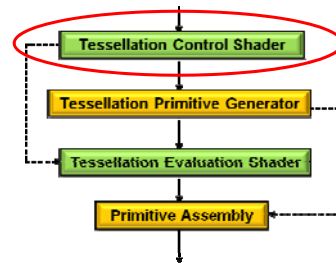


User-defined variables defined per-vertex are qualified as "out"

User-defined variables defined per-patch are qualified as "patch out"

Defining how many vertices this patch will output:

layout(vertices = 16) out;



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Tessellation Primitive Generator



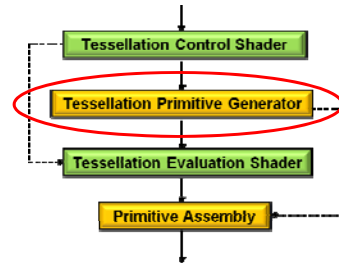
Is “fixed-function”, i.e., you can’t change its operation except by setting parameters

Consumes all vertices from the TCS and emits tessellated triangles, quads, or lines

Outputs positions as coordinates in barycentric (u,v,w)

All three coordinates (u,v,w) are used for triangles

Just (u,v) are used for quads and isolines



Oregon State University
Computer Graphics

Brown Cunningham
Associates

TES Inputs



Reads one vertex of $0 \leq (u,v,w) \leq 1$ coordinates in variable `vec3 gl_TessCoord`

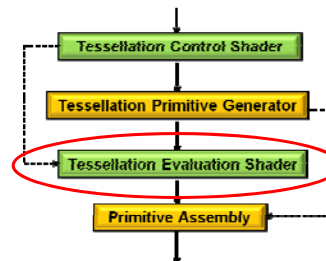
User-defined variables defined per-vertex are qualified as “out”

User-defined variables defined per-patch are qualified as “patch out”

`gl_in[ ]` is an array of structures coming from the TCS containing:

```

gl_Position;
gl_PointSize;
gl_ClipDistance[ ];
  
```



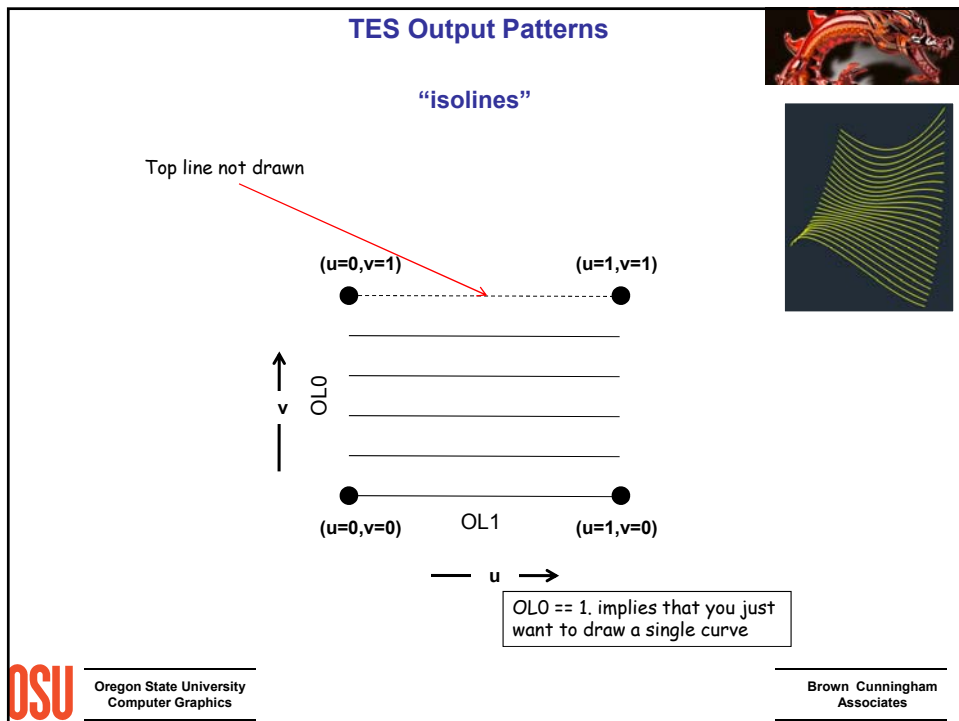
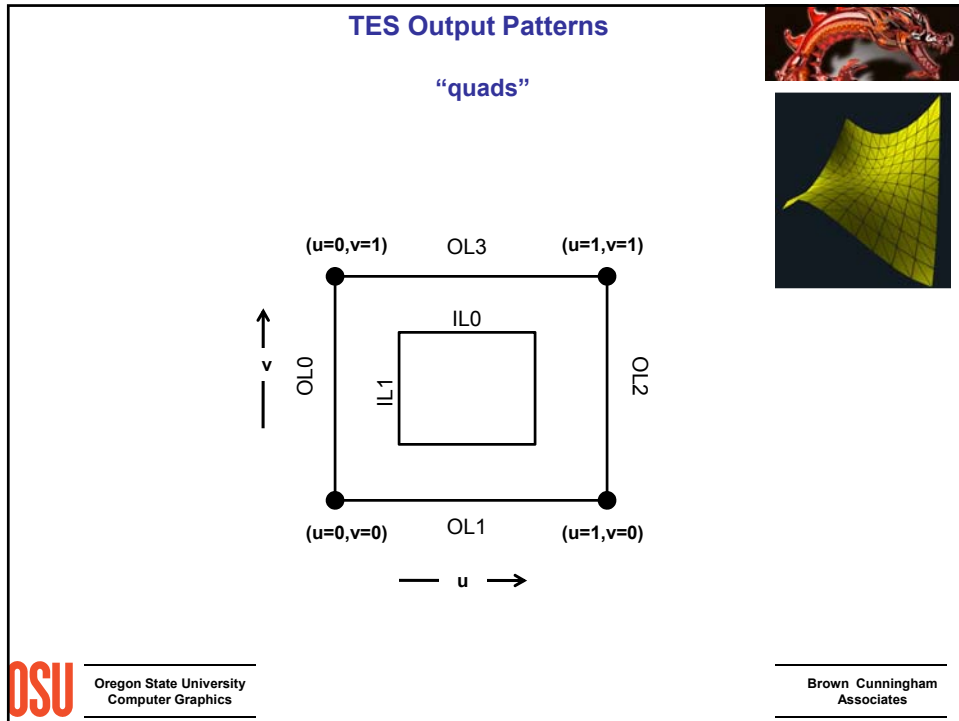
```

layout( { triangles
         quads
         isolines }, { equal_spacing
                      fractional_even_spacing
                      fractional_odd_spacing }, { ccw
         cw }, point_mode ) in;
  
```



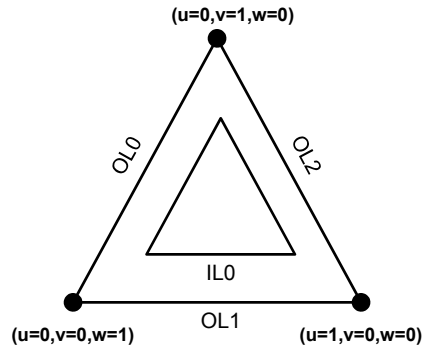
Oregon State University
Computer Graphics

Brown Cunningham
Associates

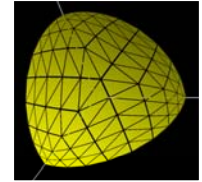
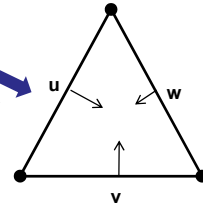


TES Output Patterns

“triangles”



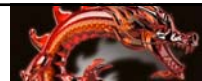
How triangle
barycentric
coordinates work



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Examples



In these examples:

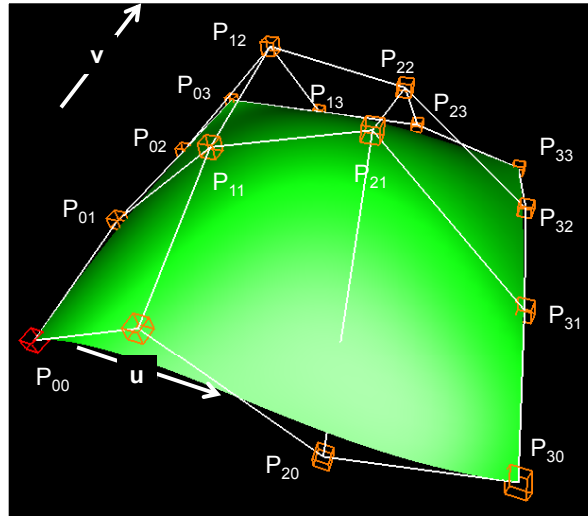
1. We are using *glman* to run them. The only necessary input files are the *glman* .glib file and the shader files. If you aren't using *glman*, you can easily also do this from a full OpenGL program.
2. All of the surface examples use the Geometry Shader *triangle-shrink* shader. This isn't necessary, but is instructional to really see how much and where the surfaces have been tessellated.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

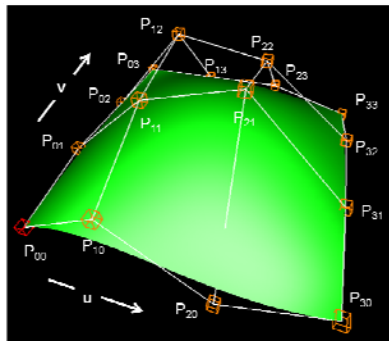
Example: A Bézier Surface



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Bézier Surface Parametric Equations



$$P(u,v) = \begin{bmatrix} (1-u)^3 & 3u(1-u)^2 & 3u^2(1-u) & u^3 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} & P_{02} & P_{03} \\ P_{10} & P_{11} & P_{12} & P_{13} \\ P_{20} & P_{21} & P_{22} & P_{23} \\ P_{30} & P_{31} & P_{32} & P_{33} \end{bmatrix} \begin{Bmatrix} (1-v)^3 \\ 3v(1-v)^2 \\ 3v^2(1-v) \\ v^3 \end{Bmatrix}$$



Oregon State University
Computer Graphics

Brown Cunningham
Associates

In the OpenGL Program



```
glPatchParameteri( GL_PATCH_VERTICES, 16 );
```

```
glBegin( GL_PATCHES );
    glVertex3f( x00, y00, z00 );
    glVertex3f( x10, y10, z10 );
    glVertex3f( x20, y20, z20 );
    glVertex3f( x30, y30, z30 );
    glVertex3f( x01, y01, z01 );
    glVertex3f( x11, y11, z11 );
    glVertex3f( x21, y21, z21 );
    glVertex3f( x31, y31, z31 );
    glVertex3f( x02, y02, z02 );
    glVertex3f( x12, y12, z12 );
    glVertex3f( x22, y22, z22 );
    glVertex3f( x32, y32, z32 );
    glVertex3f( x03, y03, z03 );
    glVertex3f( x13, y13, z13 );
    glVertex3f( x23, y23, z23 );
    glVertex3f( x33, y33, z33 );
glEnd();
```

← This order is unimportant. Pick a convention yourself and stick to it! GLSL doesn't care as long as you are consistent.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

In the .glib File



```
##OpenGL GLIB
Perspective 70

GeometryInput    gl_triangles
GeometryOutput   gl_triangle_strip

Vertex           beziersurface.vert
Fragment         beziersurface.frag
TessControl      beziersurface.tcs
TessEvaluation   beziersurface.tes
Geometry         beziersurface.geom
Program BezierSurface uOuter02 <1 10 50> uOuter13 <1 10 50> uInner0 <1 10 50> uInner1 <1 10 50> \
                  uShrink <0. 1. 1.> \
                  u LightX <-10. 0. 10.> u LightY <-10. 10. 10.> u LightZ <-10. 10. 10.>

Color [1. 1. 0.]

NumPatchVertices 16

glBegin gl_patches
glVertex 0. 2. 0.
glVertex 1. 1. 0.
glVertex 2. 1. 0.
glVertex 3. 2. 0.

glVertex 0. 1. 1.
glVertex 1. -2. 1.
glVertex 2. 1. 1.
glVertex 3. 0. 1.

glVertex 0. 0. 2.
glVertex 1. 1. 2.
glVertex 2. 0. 2.
glVertex 3. -1. 2.

glVertex 0. 0. 3.
glVertex 1. 1. 3.
glVertex 2. -1. 3.
glVertex 3. -1. 3.
glEnd
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

In the TCS Shader

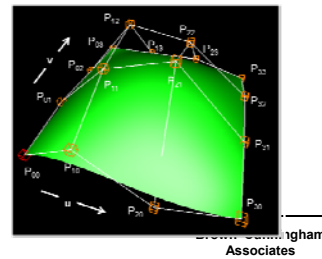
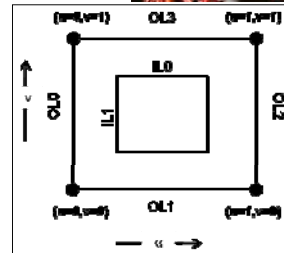
```
#version 400
#extension GL_ARB_tessellation_shader : enable

uniform float uOuter02, uOuter13, ulnner0, ulnner1;

layout( vertices = 16 ) out;

void main( )
{
    gl_out[ gl_InvocationID ].gl_Position = gl_in[ gl_InvocationID ].gl_Position;

    gl_TessLevelOuter[0] = gl_TessLevelOuter[2] = uOuter02;
    gl_TessLevelOuter[1] = gl_TessLevelOuter[3] = uOuter13;
    gl_TessLevelInner[0] = ulnner0;
    gl_TessLevelInner[1] = ulnner1;
}
```



Oregon State University
Computer Graphics

In the TES Shader

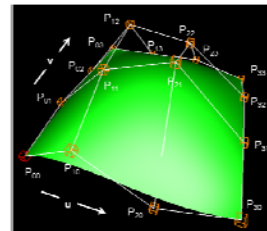
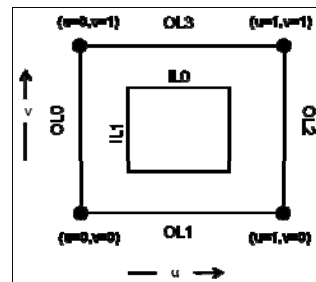
```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

layout( quads, equal_spacing, ccw ) in;

out vec3 teNormal;

void main( )
{
    vec4 p00 = gl_in[ 0 ].gl_Position;
    vec4 p10 = gl_in[ 1 ].gl_Position;
    vec4 p20 = gl_in[ 2 ].gl_Position;
    vec4 p30 = gl_in[ 3 ].gl_Position;
    vec4 p01 = gl_in[ 4 ].gl_Position;
    vec4 p11 = gl_in[ 5 ].gl_Position;
    vec4 p21 = gl_in[ 6 ].gl_Position;
    vec4 p31 = gl_in[ 7 ].gl_Position;
    vec4 p02 = gl_in[ 8 ].gl_Position;
    vec4 p12 = gl_in[ 9 ].gl_Position;
    vec4 p22 = gl_in[ 10 ].gl_Position;
    vec4 p32 = gl_in[ 11 ].gl_Position;
    vec4 p03 = gl_in[ 12 ].gl_Position;
    vec4 p13 = gl_in[ 13 ].gl_Position;
    vec4 p23 = gl_in[ 14 ].gl_Position;
    vec4 p33 = gl_in[ 15 ].gl_Position;

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
}
```



Assigning the intermediate pij's is here to make the code more readable. We assume that the compiler will optimize this away.



Oregon State University
Computer Graphics

In the TES Shader – Computing the Position



// the basis functions:

```
float bu0 = (1.-u) * (1.-u) * (1.-u);
float bu1 = 3. * u * (1.-u) * (1.-u);
float bu2 = 3. * u * u * (1.-u);
float bu3 = u * u * u;
```

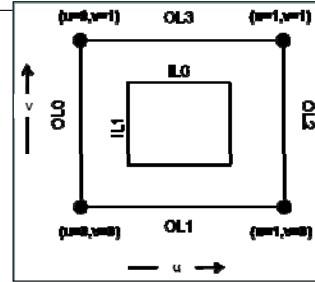
```
float dbu0 = -3. * (1.-u) * (1.-u);
float dbu1 = 3. * (1.-u) * (1.-3.*u);
float dbu2 = 3. * u * (2.-3.*u);
float dbu3 = 3. * u * u;
```

```
float bv0 = (1.-v) * (1.-v) * (1.-v);
float bv1 = 3. * v * (1.-v) * (1.-v);
float bv2 = 3. * v * v * (1.-v);
float bv3 = v * v * v;
```

```
float dbv0 = -3. * (1.-v) * (1.-v);
float dbv1 = 3. * (1.-v) * (1.-3.*v);
float dbv2 = 3. * v * (2.-3.*v);
float dbv3 = 3. * v * v;
```

// finally, we get to compute something:

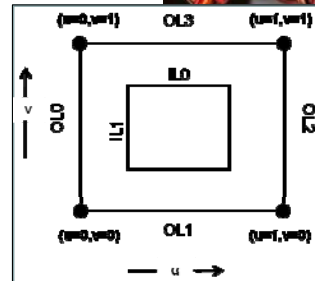
```
gl_Position =
    bu0 * ( bv0*p00 + bv1*p01 + bv2*p02 + bv3*p03 )
  + bu1 * ( bv0*p10 + bv1*p11 + bv2*p12 + bv3*p13 )
  + bu2 * ( bv0*p20 + bv1*p21 + bv2*p22 + bv3*p23 )
  + bu3 * ( bv0*p30 + bv1*p31 + bv2*p32 + bv3*p33 );
```



Oregon State University
Computer Graphics

Brown Cunningham
Associates

In the TES Shader – Computing the Normal



```
vec4 dpdu =
    dbu0 * ( bv0*p00 + bv1*p01 + bv2*p02 + bv3*p03 )
  + dbu1 * ( bv0*p10 + bv1*p11 + bv2*p12 + bv3*p13 )
  + dbu2 * ( bv0*p20 + bv1*p21 + bv2*p22 + bv3*p23 )
  + dbu3 * ( bv0*p30 + bv1*p31 + bv2*p32 + bv3*p33 );
```

```
vec4 dpdv =
    dbv0 * ( dbv0*p00 + dbv1*p01 + dbv2*p02 + dbv3*p03 )
  + dbv1 * ( dbv0*p10 + dbv1*p11 + dbv2*p12 + dbv3*p13 )
  + dbv2 * ( dbv0*p20 + dbv1*p21 + dbv2*p22 + dbv3*p23 )
  + dbv3 * ( dbv0*p30 + dbv1*p31 + dbv2*p32 + dbv3*p33 );
```

```
teNormal = normalize( cross( dpdu.xyz, dpdv.xyz ) );
```

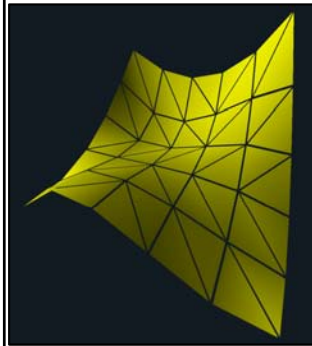
```
}
```



Oregon State University
Computer Graphics

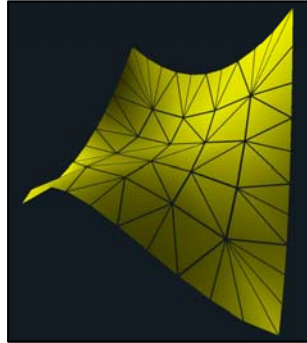
Brown Cunningham
Associates

Example: A Bézier Surface

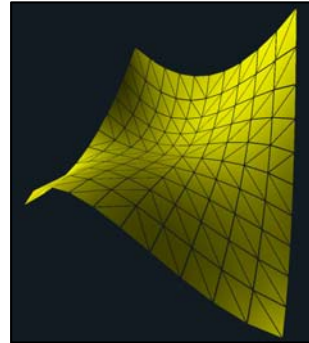


$u_{Outer02} = u_{Outer13} = 5$
 $u_{Inner0} = u_{Inner1} = 5$

$u_{Outer02} = u_{Outer13} = 10$
 $u_{Inner0} = u_{Inner1} = 5$

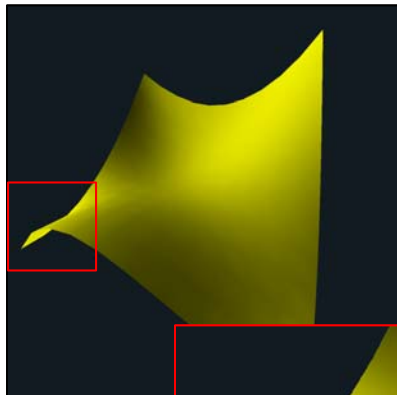


$u_{Outer02} = u_{Outer13} = 10$
 $u_{Inner0} = u_{Inner1} = 10$

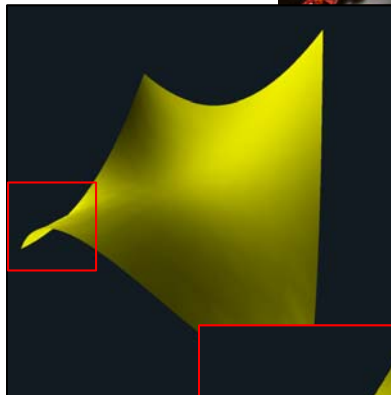


Oregon State University
Computer Graphics

Tessellation Levels and Smooth Shading



$u_{Outer02} = u_{Outer13} = 10$
 $u_{Inner0} = u_{Inner1} = 10$



$u_{Outer02} = u_{Outer13} = 30$
 $u_{Inner0} = u_{Inner1} = 10$



Oregon State University
Computer Graphics

Smoothing edge boundaries is one of the reasons that you can set Outer and Inner tessellation levels separately

John Cunningham
Associates

Example: Whole-Sphere Subdivision



spheresubd.glib

```
##OpenGL GLIB

Vertex      spheresubd.vert
Fragment    spheresubd.frag
TessControl  spheresubd.tcs
TessEvaluation spheresubd.tes
Geometry    spheresubd.geom
Program SphereSubd
    uDetail <1 30 200>
    uScale <0.1 1. 10.>
    uShrink <0. 1. 1.>
    uFlat <false>
    uColor {1. 1. 0. 0.}
    uLightX <-10. 5. 10.> uLightY <-10. 10. 10.> uLightZ <-10. 10. 10.>

Color 1. 1. 0.

NumPatchVertices 1

glBegin gl_patches
    glVertex 0. 0. 0. .2
    glVertex 0. 1. 0. .3
    glVertex 0. 0. 1. .4
glEnd
```

Using the x, y, z, and w to
specify the center and
radius of the sphere



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Whole-Sphere Subdivision



spheresubd.vert

```
#version 400 compatibility

out vec3 vCenter;
out float vRadius;

void main( )
{
    vCenter = aVertex.xyz;
    vRadius = aVertex.w;

    gl_Position = vec4( 0., 0., 0., 1. );
}
```

Using the x, y, z, and w to
specify the center and
radius of the sphere



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Whole-Sphere Subdivision



spheresubd.tcs

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

in float vRadius[ ];
in vec3 vCenter[ ];

patch out float tcRadius;
patch out vec3 tcCenter;

uniform float uDetail;
uniform float uScale;

layout( vertices = 1 ) out;

void main( )
{
    gl_out[ gl_InvocationID ].gl_Position = gl_in[ 0 ].gl_Position; // (0,0,0,1)

    tcCenter = vCenter[ 0 ];
    tcRadius = vRadius[ 0 ];

    gl_TessLevelOuter[0] = 2.;
    gl_TessLevelOuter[1] = uScale * tcRadius * uDetail;
    gl_TessLevelOuter[2] = 2.;
    gl_TessLevelOuter[3] = uScale * tcRadius * uDetail;
    gl_TessLevelInner[0] = uScale * tcRadius * uDetail;
    gl_TessLevelInner[1] = uScale * tcRadius * uDetail;
}
```

Using the scale and the radius to help set the tessellation detail

Outer[0] and Outer[2] are the number of divisions at the poles. Outer[1] and Outer[3] are the number of divisions at the vertical seams. Inner[0] and Inner[1] are the inside sphere detail.



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Example: Whole-Sphere Subdivision



spheresubd.tes

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

uniform float uScale;

layout( quads, equal_spacing, ccw ) in;

patch in float tcRadius;
patch in vec3 tcCenter;

out vec3 teNormal;

const float PI = 3.14159265;

void main( )
{
    vec3 p = gl_in[0].gl_Position.xyz;

    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    float w = gl_TessCoord.z;

    float phi = PI * ( u - .5 );
    float theta = 2. * PI * ( v - .5 );

    float cosphi = cos(phi);
    vec3 xyz = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );
    teNormal = xyz;

    xyz *= ( uScale * tcRadius );
    xyz += tcCenter;

    gl_Position = uModelViewMatrix * vec4( xyz, 1. );
}
```

$$-\frac{\pi}{2} \leq \phi \leq +\frac{\pi}{2}$$

$$-\pi \leq \theta \leq +\pi$$

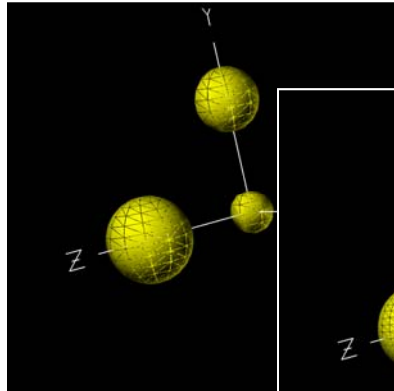
Turning u and v into spherical coordinates



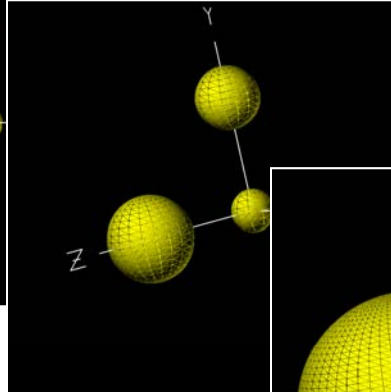
Computer Graphics

Associates

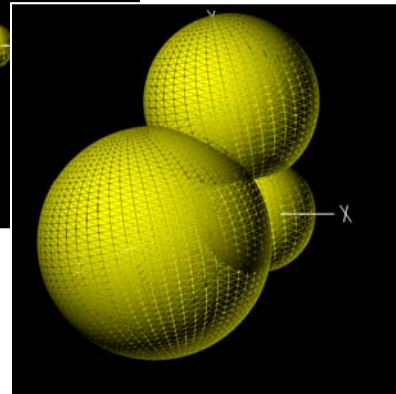
Example: Whole-Sphere Subdivision



Detail=30, Scale=1.



Detail=50, Scale=1.

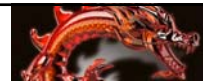


Detail=50, Scale=2.5



Oregon State University
Computer Graphics

Making the Whole-Sphere Subdivision Adapt to Screen Coverage



sphereadapt.tcs, I

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable

in float  vRadius[ ];
in vec3   vCenter[ ];

patch out float  tcRadius;
patch out vec3   tcCenter;

uniform float uDetail;

layout( vertices = 1 ) out;

void main( )
{
    gl_out[ gl_InvocationID ].gl_Position = gl_in[ 0 ].gl_Position;    // (0,0,0,1)

    tcCenter = vCenter[ 0 ];
    tcRadius = vRadius[ 0 ];

    vec4 mx = vec4( vCenter[0] - vec3( vRadius[0], 0., 0. ), 1. );
    vec4 px = vec4( vCenter[0] + vec3( vRadius[0], 0., 0. ), 1. );
    vec4 my = vec4( vCenter[0] - vec3( 0., vRadius[0], 0. ), 1. );
    vec4 py = vec4( vCenter[0] + vec3( 0., vRadius[0], 0. ), 1. );
    vec4 mz = vec4( vCenter[0] - vec3( 0., 0., vRadius[0] ), 1. );
    vec4 pz = vec4( vCenter[0] + vec3( 0., 0., vRadius[0] ), 1. );
```

Extreme points of the sphere



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Making the Whole-Sphere Subdivision Adapt to Screen Coverage



sphereadapt.tcs, II

```
mx = uModelViewProjectionMatrix * mx;
px = uModelViewProjectionMatrix * px;
my = uModelViewProjectionMatrix * my;
py = uModelViewProjectionMatrix * py;
mz = uModelViewProjectionMatrix * mz;
pz = uModelViewProjectionMatrix * pz;
```

Extreme points of the sphere in Clip space

```
mx.xy /= mx.w;
px.xy /= px.w;
my.xy /= my.w;
py.xy /= py.w;
mz.xy /= mz.w;
pz.xy /= pz.w;
```

Extreme points of the sphere in NDC space

```
float dx = distance( mx.xy, px.xy );
float dy = distance( my.xy, py.xy );
float dz = distance( mz.xy, pz.xy );
float dmax = sqrt( dx*dx + dy*dy + dz*dz );
```

How large are the lines between the extreme points?

```
gl_TessLevelOuter[0] = 2.;
gl_TessLevelOuter[1] = dmax * uDetail;
gl_TessLevelOuter[2] = 2.;
gl_TessLevelOuter[3] = dmax * uDetail;
gl_TessLevelInner[0] = dmax * uDetail;
gl_TessLevelInner[1] = dmax * uDetail;
```

We no longer use uScale or tcRadius. But, we do use uDetail to provide a way to convert from NDC to Screen Space or to indicate the quality you'd like

(I.e., uDetail depends on how good you want the spheres to look and on how large the window is in pixels.)



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Making the Whole-Sphere Subdivision Adapt to Screen Coverage



sphereadapt.tes

```
#version 400 compatibility
#extension GL_ARB_tessellation_shader : enable
```

```
layout( quads, equal_spacing, ccw ) in;
```

```
patch in float tcRadius;
patch in vec3 tcCenter;
```

```
out vec3 teNormal;
```

```
const float PI = 3.14159265;
```

```
void main( )
```

```
{
```

```
vec3 p = gl_in[0].gl_Position.xyz;
```

```
float u = gl_TessCoord.x;
float v = gl_TessCoord.y;
float w = gl_TessCoord.z;
```

$$-\frac{\pi}{2} \leq \phi \leq +\frac{\pi}{2}$$

$$-\pi \leq \theta \leq +\pi$$

```
float phi = PI * ( u - .5 );
float theta = 2. * PI * ( v - .5 );
```

```
float cosphi = cos(phi);
vec3 xyz = vec3( cosphi*cos(theta), sin(phi), cosphi*sin(theta) );
teNormal = xyz;
```

Spherical coordinates

```
xyz *= tcRadius;
xyz += tcCenter;
```

No longer uses uScale

```
gl_Position = uModelViewMatrix * vec4( xyz, 1. );
```

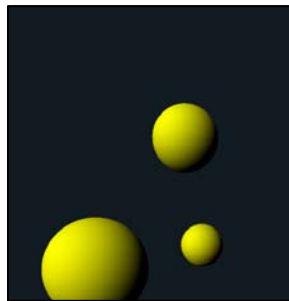
```
}
```



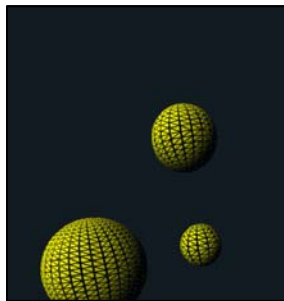
Computer Graphics

Associates

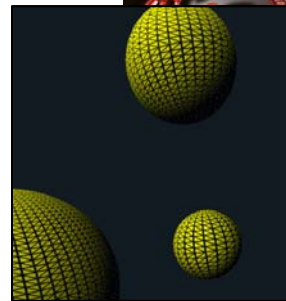
Making the Whole-Sphere Subdivision Adapt to Screen Coverage



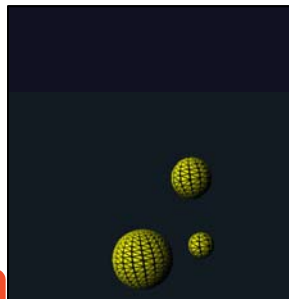
Original



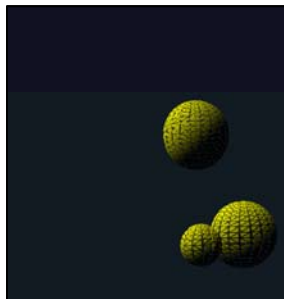
Triangles Shrunk



Zoomed In



Zoomed Out



Rotated

Notice that the number of triangles adapts to the screen coverage of each sphere, and that the size of the tessellated triangles stays about the same, regardless of radius or transformation

Brown Cunningham
Associates

The Difference Between Tessellation Shaders and Geometry Shaders

By now, you are probably confused about when to use a Geometry Shader and when to use a Tessellation Shader. Both are capable of creating new geometry from existing geometry. See if this helps.

Use a **Geometry Shader** when:

1. You need to convert geometry topologies, such as the silhouette and hedgehog shaders (triangles→lines) or the explosion shader (triangles→points)
2. You need some sort of geometry processing to come after the Tessellation Shader (such as how the shrink shader was used here)

Use a **Tessellation Shader** when you need to generate many new vertices and one of the tessellation topologies will suit your needs.

Use a **Tessellation Shader** when you need more than 6 input vertices to define the surface being tessellated..



Oregon State University
Computer Graphics

Brown Cunningham
Associates



Two Windows Program Executables and Lots of Shader Files



Many of you have them on the *glman* CD

For those who don't, you can get a .zip file of everything by going to:

<http://cs.oregonstate.edu/~mjb/glman>

and following the link that says "SIGGRAPH Asia 2011 Attendees"



Oregon State University
Computer Graphics

Brown Cunningham
Associates

Mike Bailey
Oregon State University
mjb@cs.oregonstate.edu

Steve Cunningham
Brown Cunningham Associates
rsc@cs.csustan.edu

Computer Graphics Shaders
using OpenGL 4.x