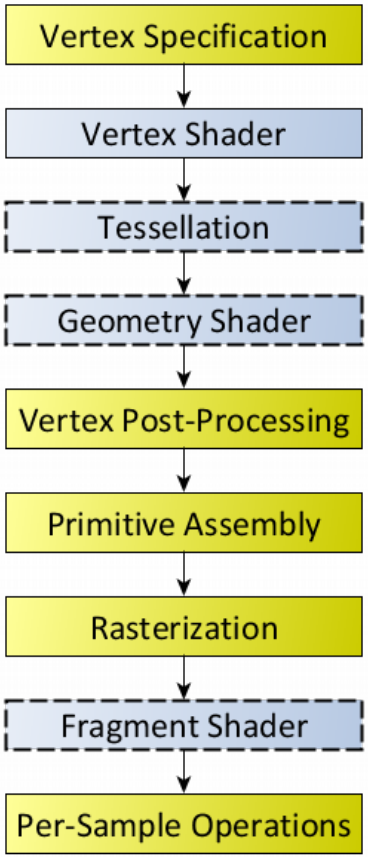
CPSC 453

Talking with Shaders

Demo 4 and Tesselation Shaders

UNIVERSITY OF CALGARY

“The Fun Stuff”



https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

UNIVERSITY OF CALGARY

What can we do with Shaders?

UNIVERSITY OF CALGARY

How can we communicate with Shaders?

Vertex Shaders

```
// Vertex shader
#version 410

layout(location = 0) in vec2 input;
out vec4 output;

void main() {                          // just copy it over :'(
  output = vec4(input, 0.0, 1.0);
}
```

UNIVERSITY OF CALGARY

Communication

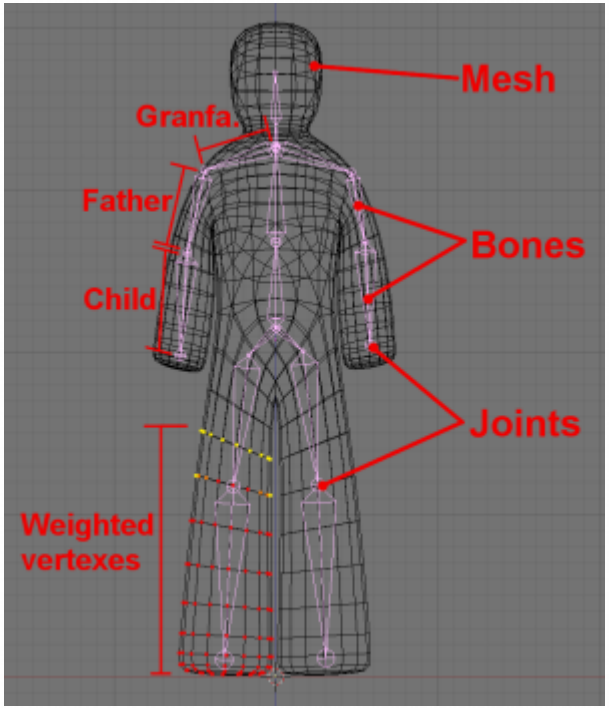
1) Indirect / Direct
2) glUniform()
3) Textures
4) Uniform Buffer Objects
5) Shader Storage Buffer Objects

Skeletal Animation

Magnenat-Thalmann, Nadia, Richard Laperrire, and Daniel Thalmann. "Joint-dependent local deformations for hand animation and object grasping." In Proceedings on Graphics interface'88. 1988.
http://graphicsinterface.org/wp-content/uploads/gi1988-4.pdf

Indirect

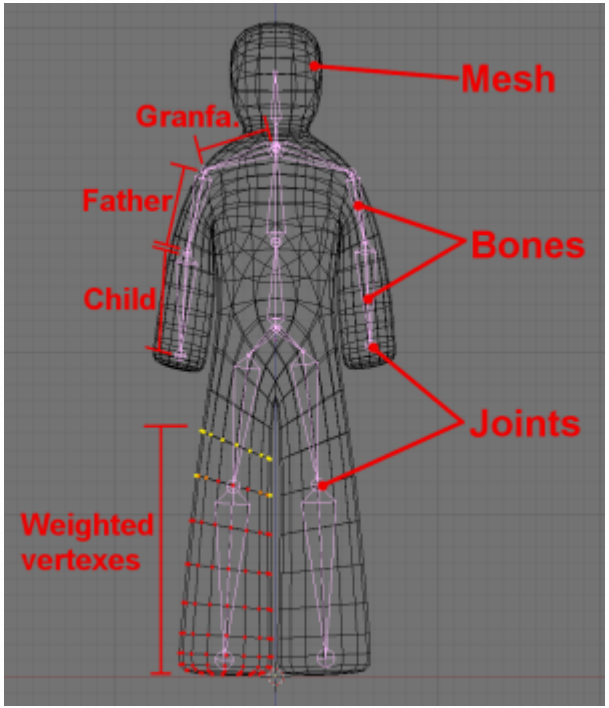
https://web.archive.org/web/20141105204302/http://content.gpwiki.org/index.php?title=OpenGL:Tutorials:Basic_Bones_System



https://voxels.blogspot.ca/2014/03/skinned-skeletal-animation-tutorial.html

Indirect

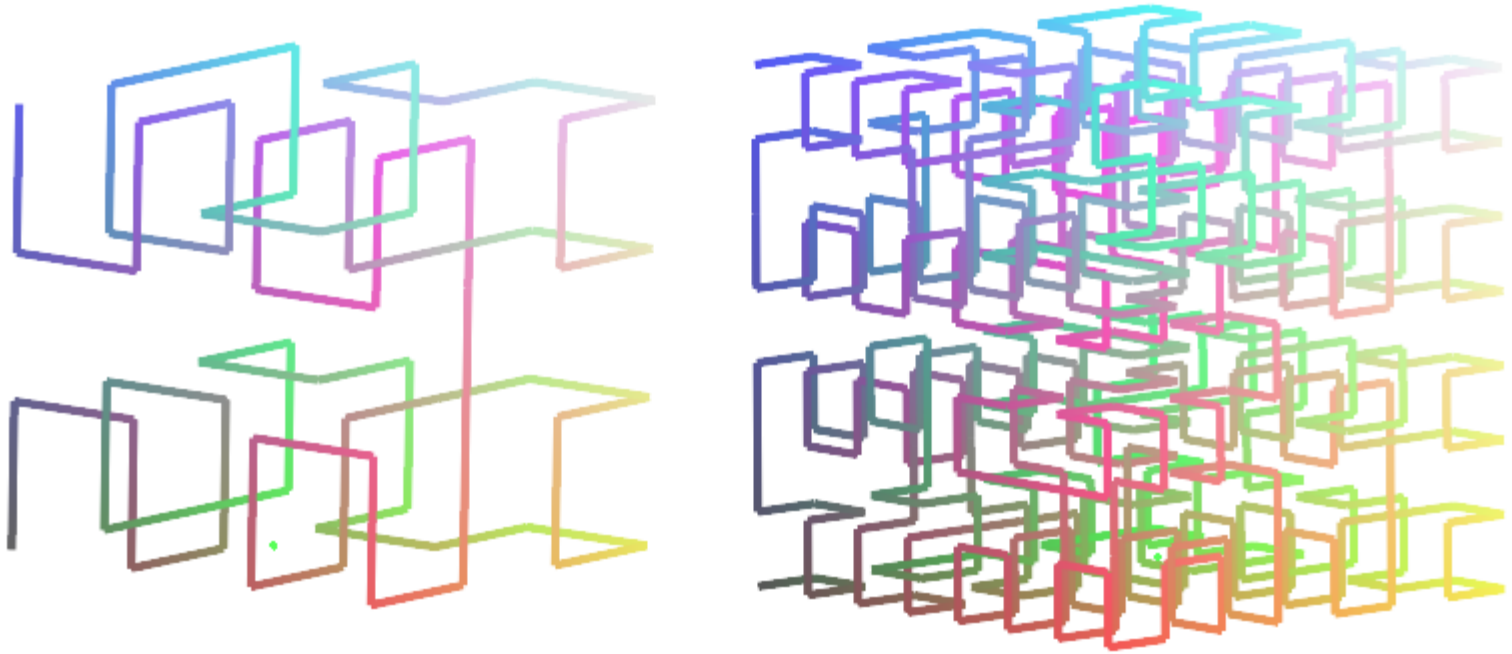
https://web.archive.org/web/20141105204302/http://content.gpwiki.org/index.php?title=OpenGL:Tutorials:Basic_Bones_System




```
uniform mat4 bones[100];
uniform int  use_skinning;

void main(void)
{
  mat4 mfinal = gl_ModelViewMatrix ;

  // skinning
  if(use_skinning==1)
  {
    vec3 weights= gl_Color.xyz;
    vec3 boneid = gl_Vertex.w * vec3( 1.0/128.0 , 1.0 , 128.0 );
    boneid = (boneid - floor(boneid))*128.0;

    mat4 mskin  = bones[int(boneid.x)]*weights.x+
                  bones[int(boneid.y)]*weights.y+
                  bones[int(boneid.z)]*weights.z;
    mfinal = mfinal * mskin;
  }
  gl_Position = gl_ProjectionMatrix * mfinal * vec4(gl_Vertex.xyz,1.0);
}
```

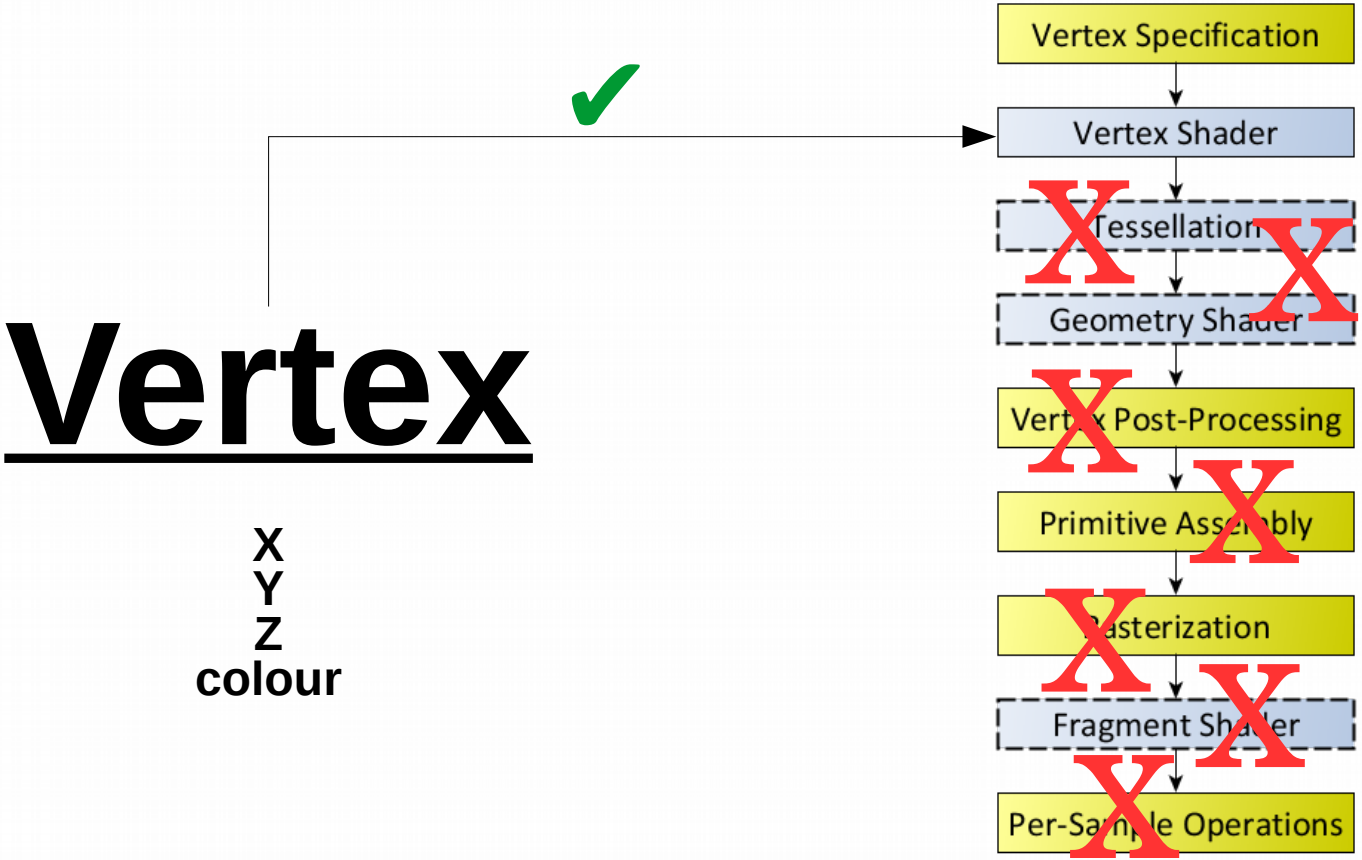
https://voxels.blogspot.ca/2014/03/skinned-skeletal-animation-tutorial.html

Indirect

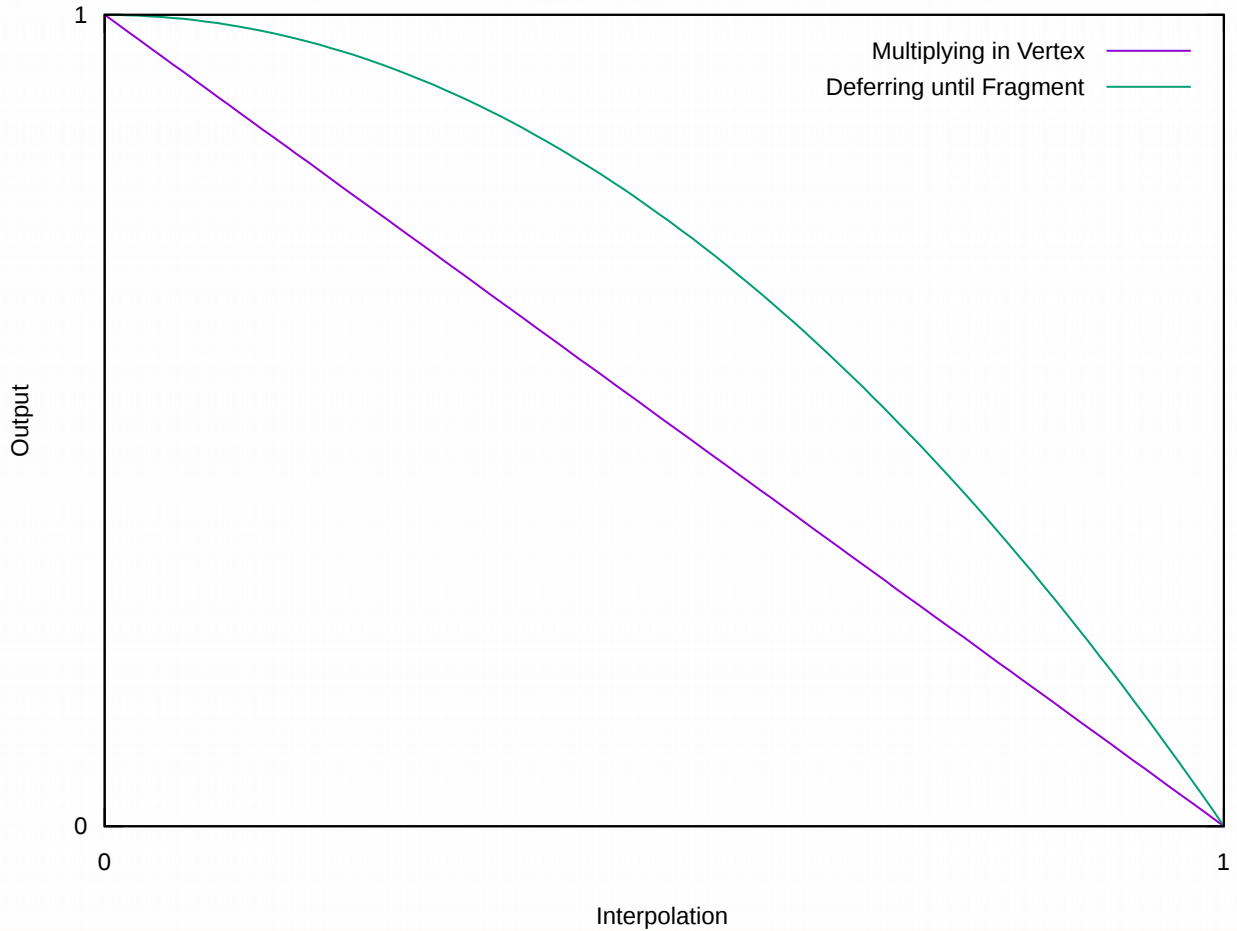


http://www.space-filling-curves.org/

Indirect


Vertex
X
Y
Z
colour
Vertex Specification
Vertex Shader
Tessellation
Geometry Shader
Vertex Post-Processing
Primitive Assembly
Rasterization
Fragment Shader
Per-Sample Operations

Leave Z and W alone!



Inter-Shader Variables

```
// Vertex shader
#version 410

layout(location = 0) in vec4 input; // W = colour
out vec4 position;                  // screen position

uniform mat4 MVP;                   // transform matrix

out float distance;                 // … oh?

void main() {

     distance = input.w;
     position = MVP * vec4(input.xyz, 1.0);
     }


// Fragment Shader
#version 410

in vec4 position;
in float distance;                  // oh-ho!
out vec4 colour;

void main() {

     colour = mix( vec4(1,1,0,1), vec4(0,0,1,1), distance );
     }
```

https://www.khronos.org/opengl/wiki/Shader_Compilation#Interface_matching

Dragon

(see D2L)

UNIVERSITY OF CALGARY

Tesselation Shaders

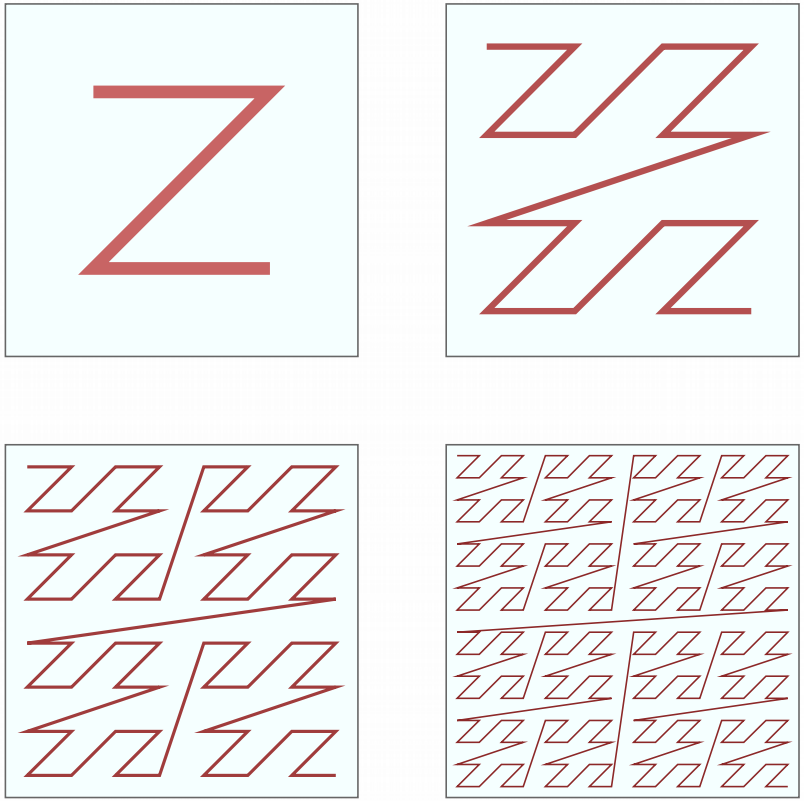
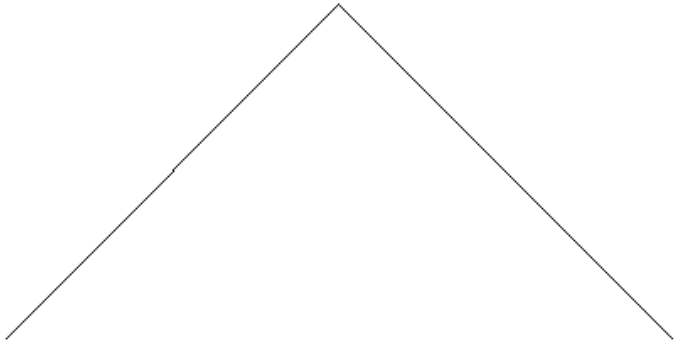


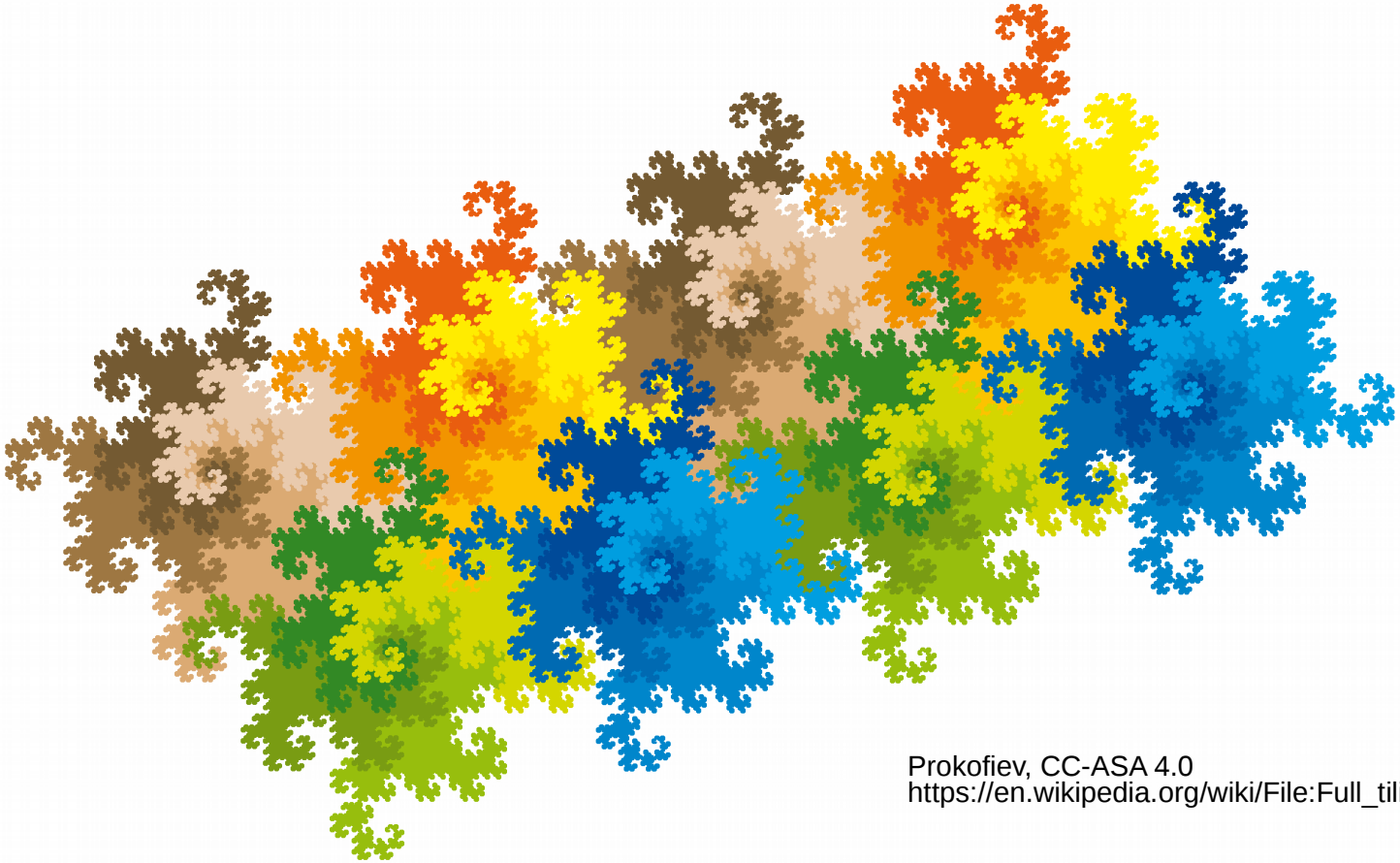
Illustration by David Eppstein
https://en.wikipedia.org/wiki/File:Four-level_Z.svg

Dragon Curve



Guillaume Jacquenot, CC-ASA 3.0
https://en.wikipedia.org/wiki/File:DragonCurve_animation.gif

Dragon is Space Filling

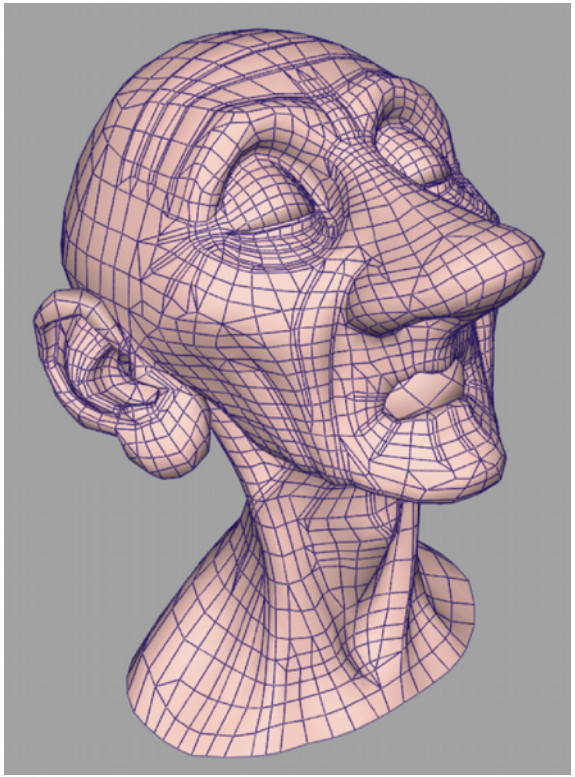


Prokofiev, CC-ASA 4.0
https://en.wikipedia.org/wiki/File:Full_tiling_dragon2.svg

UNIVERSITY OF CALGARY

Tesselation Shaders

Tesselation

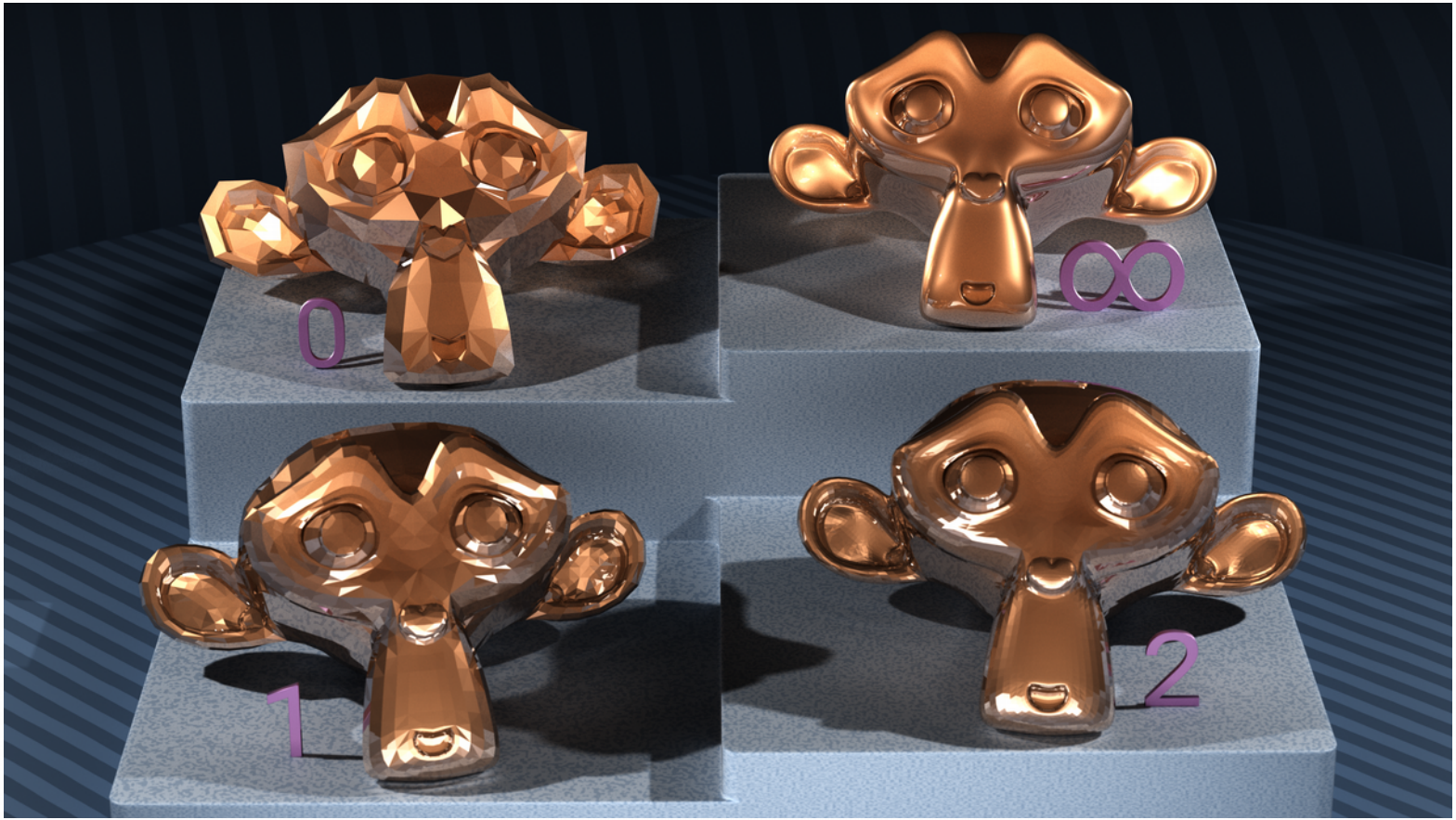


DeRose, Tony, Michael Kass, and Tien Truong. “Subdivision Surfaces in Character Animation.” In Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques, 85–94. ACM, 1998. http://dl.acm.org/citation.cfm?id=280826.

https://graphics.pixar.com/library/Geri/paper.pdf

UNIVERSITY OF CALGARY

Tesselation



Tesselation Control

How do I break up a patch?

https://www.khronos.org/opengl/wiki/Tessellation

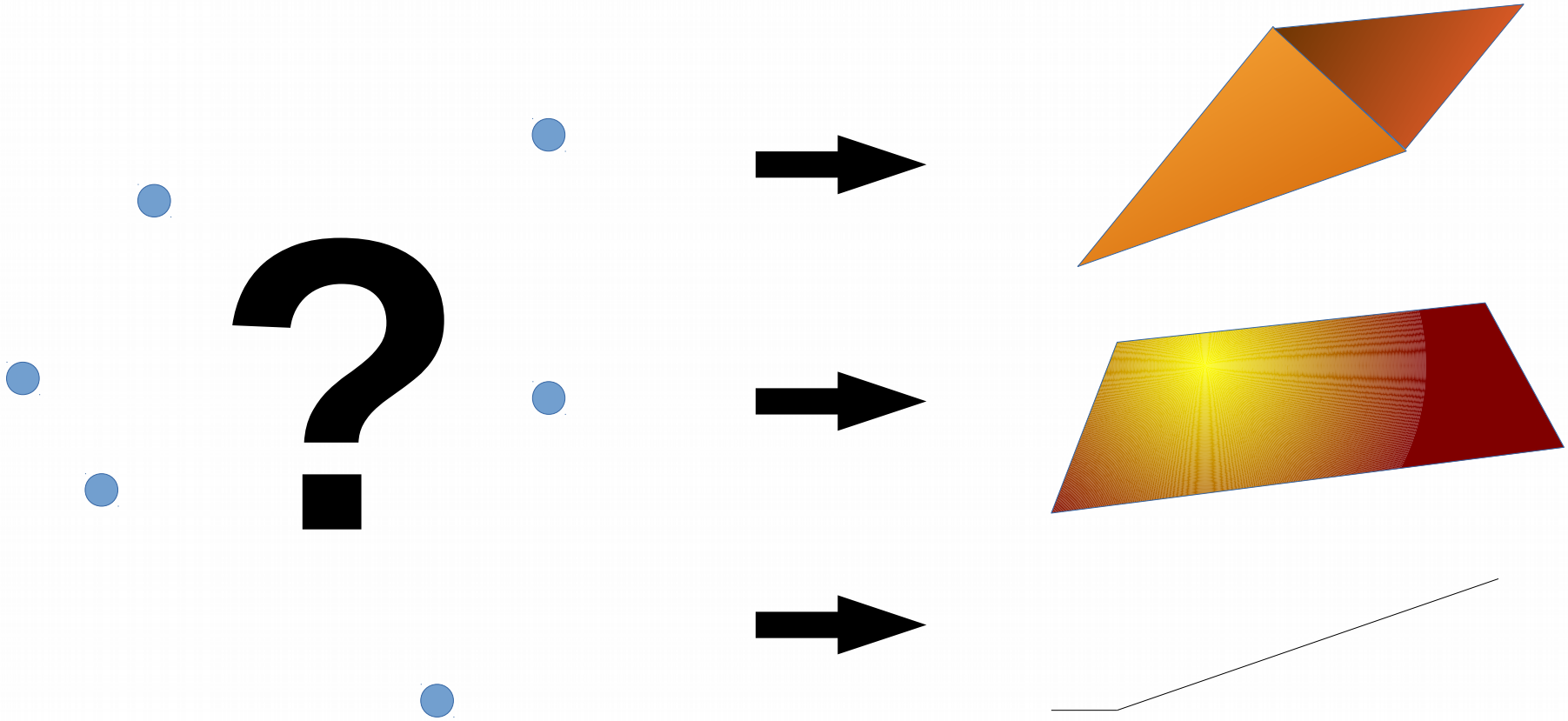
Tesselation Evaluation

How do I convert a patch back to geometry?

https://www.khronos.org/opengl/wiki/Tessellation_Evaluation_Shader

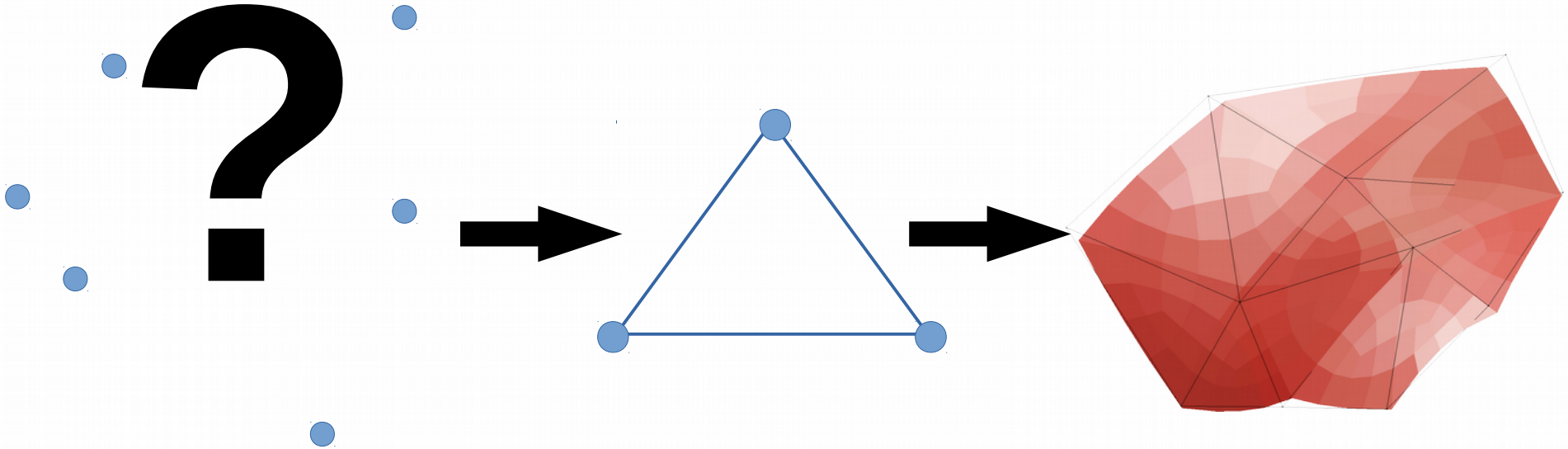
UNIVERSITY OF CALGARY

Abstract Patches



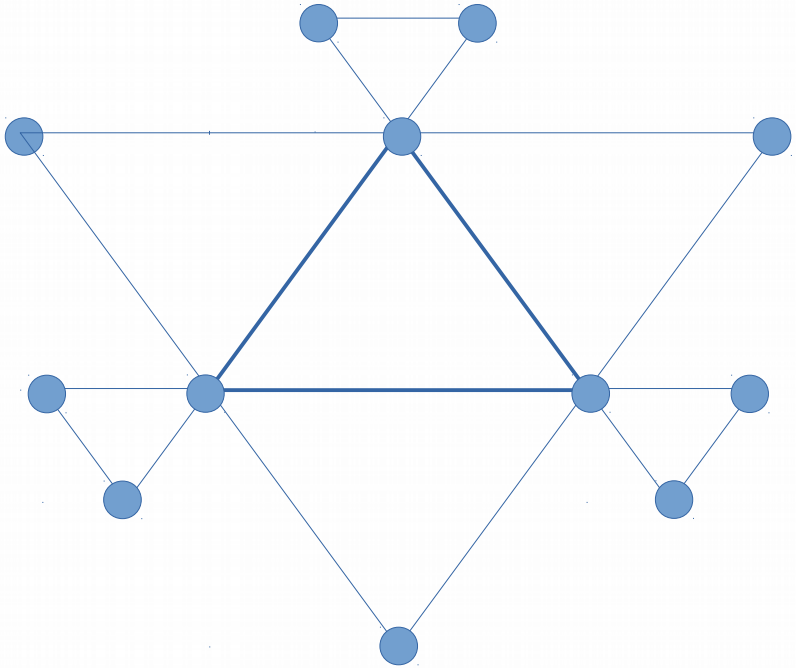
UNIVERSITY OF CALGARY

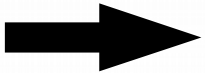
Abstract Patches

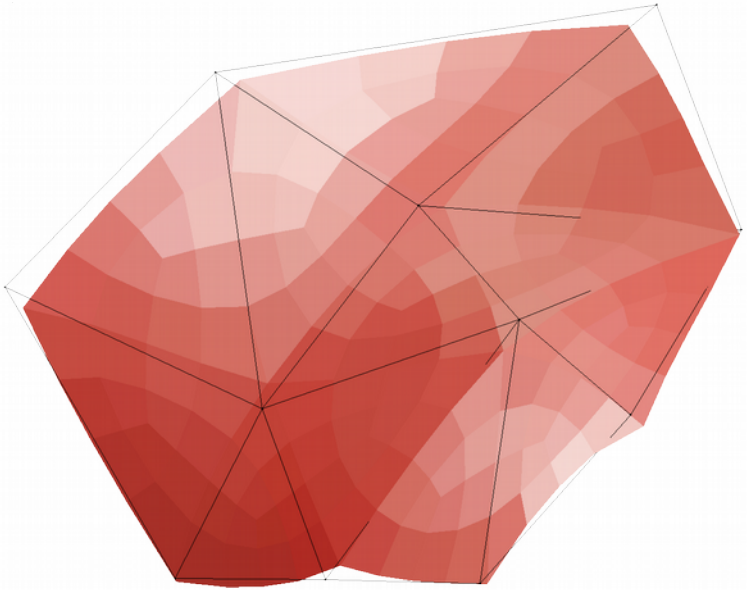


UNIVERSITY OF CALGARY

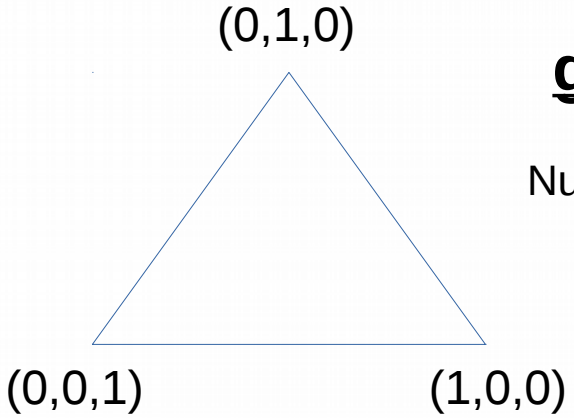
Abstract Patches







Tesselation Control - Triangles



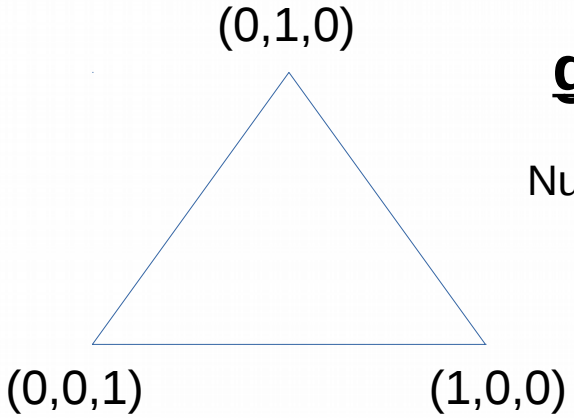
gl_TessLevelInner

Number of inner triangles/points

gl_TessLevelOuter

Number of segments along triangle edges

Tesselation Control - Triangles

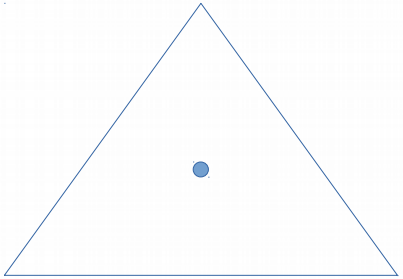


gl_TessLevelInner

Number of inner triangles/points

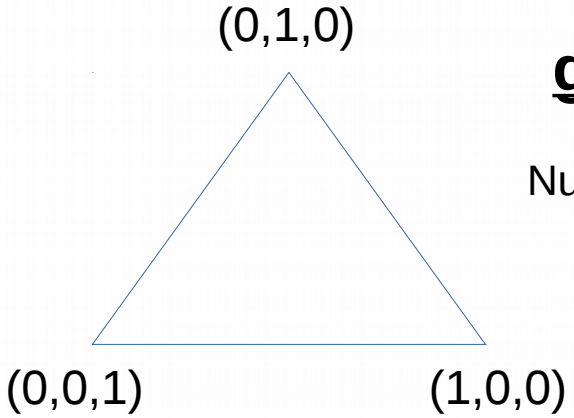
gl_TessLevelOuter

Number of segments along triangle edges



```
gl_TessLevelInner[0] = 2;
```

Tesselation Control - Triangles

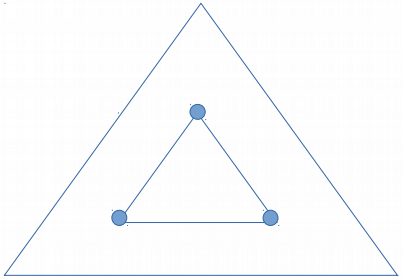


gl_TessLevelInner

Number of inner triangles/points

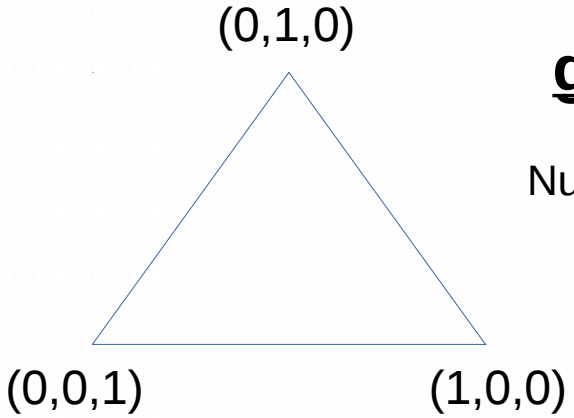
gl_TessLevelOuter

Number of segments along triangle edges



```
gl_TessLevelInner[0] = 3;
```

Tesselation Control - Triangles

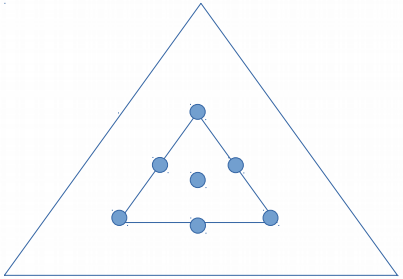


gl_TessLevelInner

Number of inner triangles/points

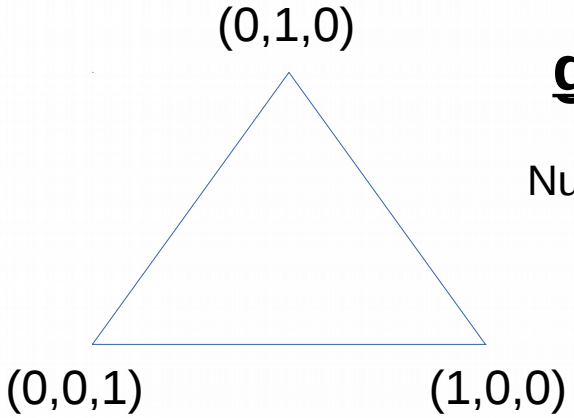
gl_TessLevelOuter

Number of segments along triangle edges



```
gl_TessLevelInner[0] = 4;
```

Tesselation Control - Triangles

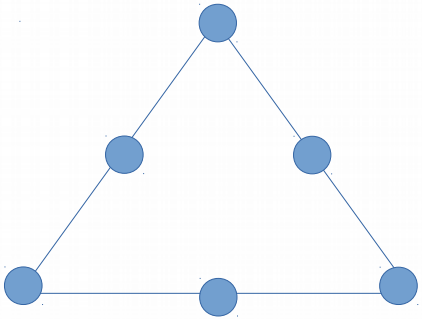


gl_TessLevelInner

Number of inner triangles/points

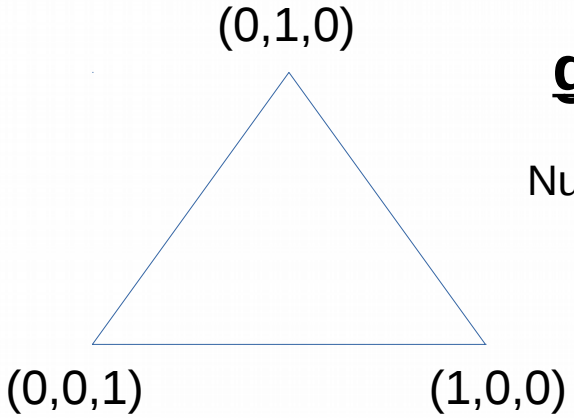
gl_TessLevelOuter

Number of segments along triangle edges



```
gl_TessLevelOuter[0] = 2;
gl_TessLevelOuter[1] = 2;
gl_TessLevelOuter[2] = 2;
```

Tesselation Control - Triangles

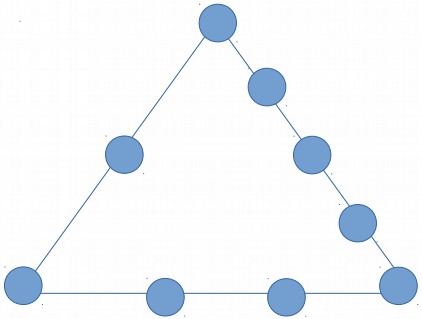


gl_TessLevelInner

Number of inner triangles/points

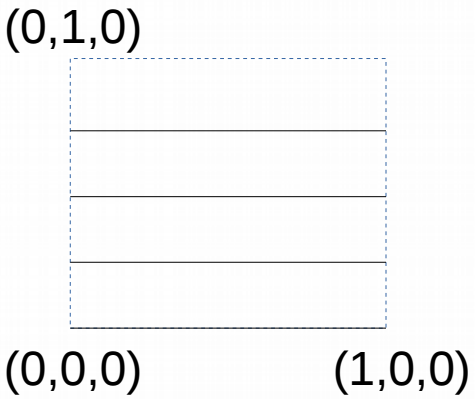
gl_TessLevelOuter

Number of segments along triangle edges



```
gl_TessLevelOuter[0] = 2;
gl_TessLevelOuter[1] = 3;
gl_TessLevelOuter[2] = 4;
```

Tesselation Control – Isolines



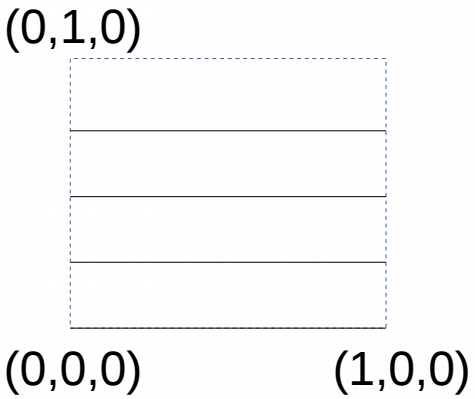
gl_TessLevelInner

Ignored

gl_TessLevelOuter

Number of lines and segments

Tesselation Control – Isolines



gl_TessLevelInner

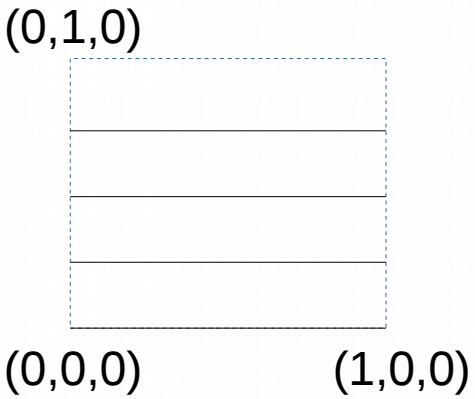
Ignored

gl_TessLevelOuter

Number of lines and segments

```
gl_TessLevelOuter[0] = 1;
gl_TessLevelOuter[1] = 2;
```

Tesselation Control – Isolines



gl_TessLevelInner

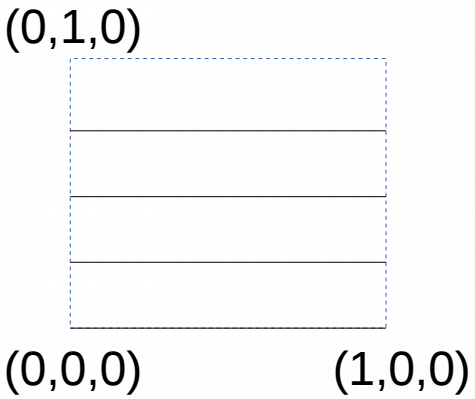
Ignored

gl_TessLevelOuter

Number of lines and segments

```
gl_TessLevelOuter[0] = 1;
gl_TessLevelOuter[1] = 4;
```

Tesselation Control – Isolines

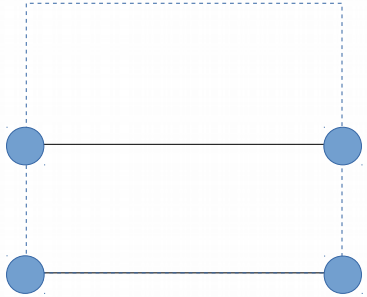


gl_TessLevelInner

Ignored

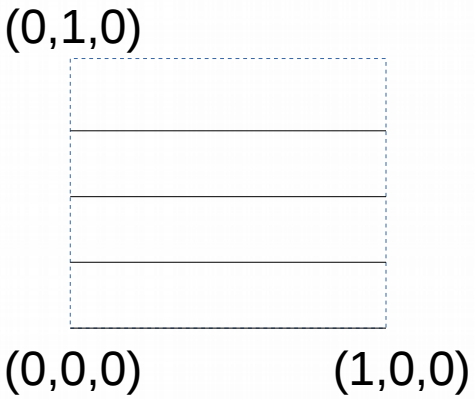
gl_TessLevelOuter

Number of lines and segments



```
gl_TessLevelOuter[0] = 2;
gl_TessLevelOuter[1] = 1;
```

Tesselation Control – Isolines

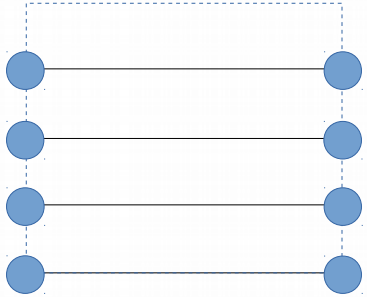


gl_TessLevelInner

Ignored

gl_TessLevelOuter

Number of lines and segments



```
gl_TessLevelOuter[0] = 4;
gl_TessLevelOuter[1] = 1;
```

UNIVERSITY OF CALGARY

Dragon

(see D2L)

Setting Recursion

```
uniform int maxDivs = 2;                    // in tesselation shaders
```

Setting Recursion

```
uniform int maxDivs = 2;                // in tesselation shaders


ShaderProgram program;                  // CPU
// … skipping ahead ...
program.setInt( "maxDivs", recursion );
```

Setting Recursion

```
uniform int maxDivs = 2;                // in tesselation shaders


ShaderProgram program;                  // CPU
// … skipping ahead ...
program.setInt( "maxDivs", recursion );


bool ShaderProgram::setInt( string variable, GLint value ) {

        // grab the uniform's location, if possible
        GLint location = glGetUniformLocation( id,
variable.c_str() );
        if ( location < 0 )
                return false;

        glUniform1i( location, value );
        return true;
         }
```

View Transformations

```
// tesselation evaluation shader (?!)
uniform float width = 1.0;      // handle window aspect ratio
uniform float height = 1.0;

void main()
{

        // scale coordinates to compensate for window aspect
        vec4 scale = vec4( width, height, 1.0, 1.0 );

        // ...

        // some easy cases
        if (u == 0.0) {

                gl_Position = vec4( left.xy, 0.0, 1.0 ) *
scale;
                fragDistance = TEdistance[0];
                return;
                }
        // ...
```

UNIVERSITY OF CALGARY

glUniform

```
// update a vec4
void glUniform4f( GLint location,
    GLfloat v0, GLfloat v1, GLfloat v2, GLfloat v3 );

// update an array of ivec3
void glUniform3iv( GLint location, GLsizei count,
    const GLint *value);

// update an array of mat4
void glUniformMatrix4fv( GLint location, GLsizei count,
    GLboolean transpose, const GLfloat *value);

// and more!
// https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/glUniform.xhtml
```

Assigning Colour

```
// CPU
vector<GLfloat> pointsIn = {    -5./12., 1./6., 0.0,        // three coords?
                                7./12., 1./6., 1.0 };
```

Assigning Colour

```
// vertex shader
layout(location = 0) in vec3 input;      // the incoming vertex data

out vec2 TCposition;                     // the outgoing position data
out float TCdistance;                    //  and distance along the line

void main() {

        // the tesselation shader handles transformations (?!)
        TCposition = input.xy;
        TCdistance = input.z;
}
```

Assigning Colour

```
// tesselation control
in vec2 TCposition[];              // incoming vertex data. Note the array!
in float TCdistance[];

layout (vertices = 2) out;         // two outgoing verticies
out vec2 TEposition[];             // outgoing vertex data
out float TEdistance[];            //  attributes are also in array form

uniform int maxDivs = 2;           // how many subdivisions?

void main() {

        gl_TessLevelOuter[0] = 1;                  // how to tesselate the patch
        gl_TessLevelOuter[1] = 1 << maxDivs;

                                                   // pass these through
        TEposition[gl_InvocationID] = TCposition[gl_InvocationID];
        TEdistance[gl_InvocationID] = TCdistance[gl_InvocationID];

        }
```

Assigning Colour

```
// tesselation evaluation
layout (isolines) in;                   // are we dealing with lines?

// in vec3 gl_TessCoord;                // where on the abstract patch this eval is located
in vec2 TEposition[];                   // incoming vertex data (again in an array)
in float TEdistance[];                  //  and curve distance

out gl_PerVertex {                      // stuck with old-style position data :'(
        vec4 gl_Position;
        float gl_PointSize;
        float gl_ClipDistance[];
        };

out float fragDistance;                 // but other attributes are different
```

Assigning Colour

```
// tesselation evaluation (again)
void main()
{
        // ...
        float u = gl_TessCoord.x;       // we don't use y

        vec3 left = vec3( TEposition[0], TEdistance[0] );
        vec3 right = vec3( TEposition[1], TEdistance[1] );

        // some easy cases
        if (u == 0.0) {
                gl_Position = vec4( left.xy, 0.0, 1.0 ) * scale;
                fragDistance = TEdistance[0];
                return;
                }

        // ...
```

Assigning Colour

```
// tesselation evaluation (still)
                midpoint = ( left + right ) * 0.5;
                vec3 perpen = vec3( (midpoint - left).yx, 0.0 );
                perpen.x *= -1.0;
                midpoint += sign*perpen;

                // branch, depending on location along the curve
                // ...
                else {
                        // hit it precisely? This point never moves
                        gl_Position = vec4( midpoint.xy, 0.0, 1.0 ) * scale;
                        fragDistance = midpoint.z;
                        return;
                        }

                // ...
```

Assigning Colour

```
// fragment
// ...
in float fragDistance;          // how far along the curve?
out vec4 fragColour;

void main() {
        float wavelength = (fragDistance * 7000.0) + 2000.0;
        fragColour = vec4( kelvinToRGB(wavelength), 1.0 );
        }
```

Communication

1) Indirect / Direct ✔
2) glUniform() ✔
3) Textures
4) Uniform Buffer Objects
5) Shader Storage Buffer Objects

Texture Size

Name

textureSize — retrieve the dimensions of a level of a texture

Declaration

```
int textureSize( gsampler1D sampler,
                 int lod);

ivec2 textureSize( gsampler2D sampler,
                   int lod);

ivec3 textureSize( gsampler3D sampler,
                   int lod);
```

Description

textureSize returns the dimensions of level *lod* (if present) of the texture bound to *sampler*. The components in the return value are filled in, in order, with the width, height and depth of the texture. For the array forms, the last component of the return value is the number of layers in the texture array.

https://www.khronos.org/registry/OpenGL-Refpages/gl4/html/textureSize.xhtml

Uniform Buffer Objects

```
// Tesselation Evaluation output
// Vertex output
out gl_PerVertex {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
    };
```

Uniform Buffer Objects

```
// Tesselation Evaluation output
// Vertex output
out gl_PerVertex {
    vec4 gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
    };
```

http://www.geeks3d.com/20140704/gpu-buffers-introduction-to-opengl-3-1-uniform-buffers-objects/

UNIVERSITY OF CALGARY

Shader Storage Buffer Objects

SSBOs are a lot like Uniform Buffer Objects. Shader storage blocks are defined by Interface Block (GLSL)s in almost the same way as uniform blocks. Buffer objects that store SSBOs are bound to SSBO binding points, just as buffer objects for uniforms are bound to UBO binding points. And so forth.

The major differences between them are:

1. SSBOs can be much larger. The OpenGL spec guarantees that UBOs can be up to 16KB in size (implementations can allow them to be bigger). The spec guarantees that SSBOs can be up to 128**MB**. Most implementations will let you allocate a size up to the limit of GPU memory.
2. SSBOs are writable, even atomically; UBOs are `uniforms`. SSBOs reads and writes use incoherent memory accesses, so they need the appropriate barriers, just as Image Load Store operations.
3. SSBOs can have variable storage, up to whatever buffer range was bound for that particular buffer; UBOs must have a specific, fixed storage size. This means that you can have an array of arbitrary length in an SSBO (at the end, rather). The actual size of the array, based on the range of the buffer bound, can be queried at runtime in the shader using the `length` function on the unbounded array variable.
4. SSBO access, all things being equal, will likely be slower than UBO access. SSBOs generally are accesses like buffer textures, while UBO data is accessed through internal shader-accessible memory reads. At the very least, UBOs will be no slower than SSBOs.

Functionally speaking, SSBOs can be thought of as a much nicer interface to Buffer Textures when accessed via Image Load Store.

https://www.khronos.org/opengl/wiki/Shader_Storage_Buffer_Object

UNIVERSITY OF CALGARY

Summary

Indirect / Direct	A few per-vertex values.
glUniform()	A handful of constants.
Textures	Large dataset, accessed via fragment shader.
Uniform Buffer Objects	A structured handful of constants.
Shader Storage Buffer Objects	Large read/write dataset.

Summary

Indirect / Direct !?!?	A few per-vertex values.
glUniform()	A handful of constants.
Textures	Large dataset, accessed via fragment shader.
Uniform Buffer Objects	A structured handful of constants.
Shader Storage Buffer Objects	Large read/write dataset.