Lab 6
Tessellation Shaders

CSE 4431/5331.03M

Advanced Topics in 3D Computer Graphics

TA: Margarita Vinnikov

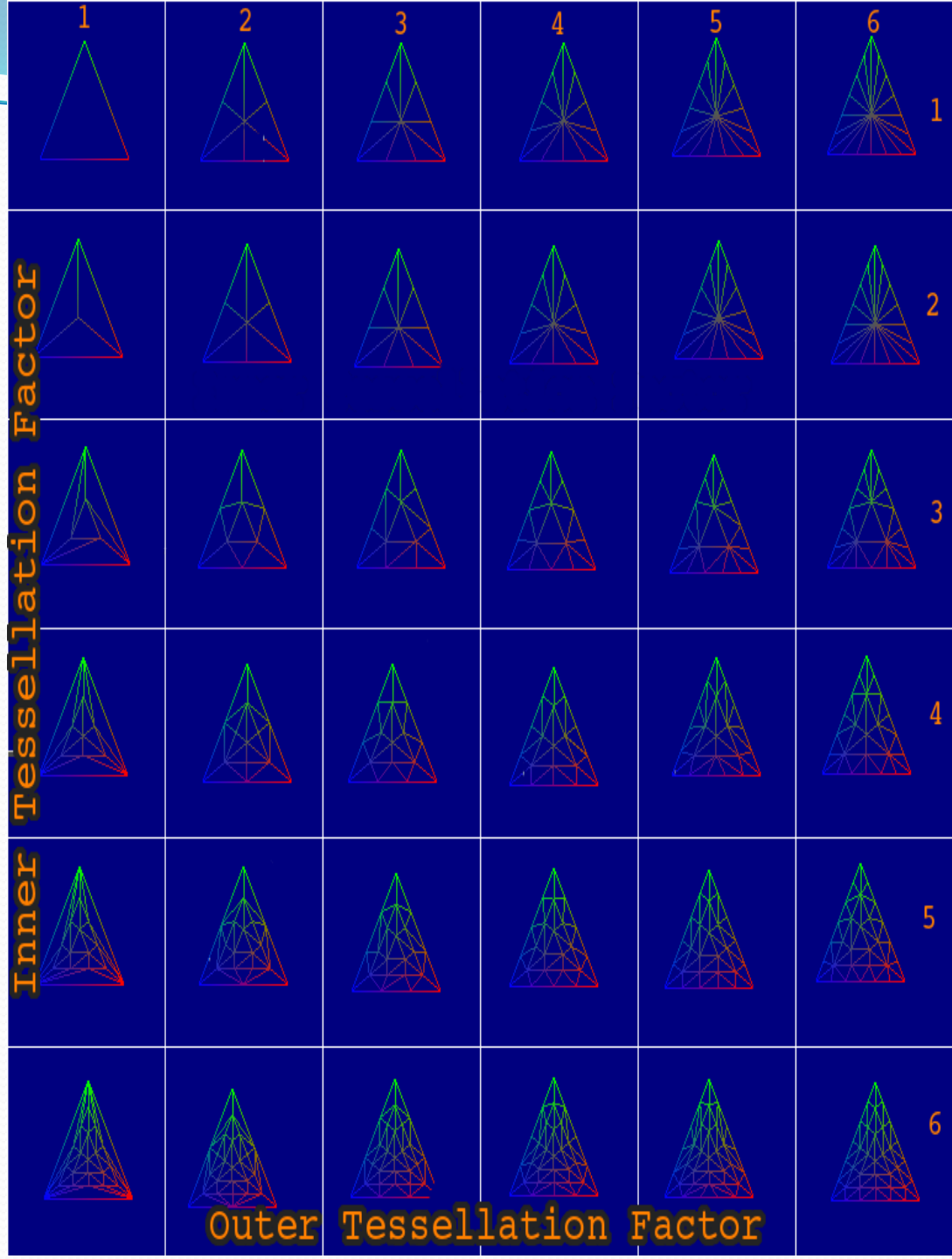
mvinni@cse.yorku.ca

The OpenGL 4.x pipeline

- 2 new Programmable stages
  - Tessellation Control Shader(GL_TESS_CONTROL_SHADER)
  - Tessellation Evaluation Shader(GL_TESS_EVALUATION_SHADER)
- 1 new Fixed function stage
  - Tessellation primitive generator aka tessellator
- 1 new primitive type
  - Patches (GL_PATCHES)

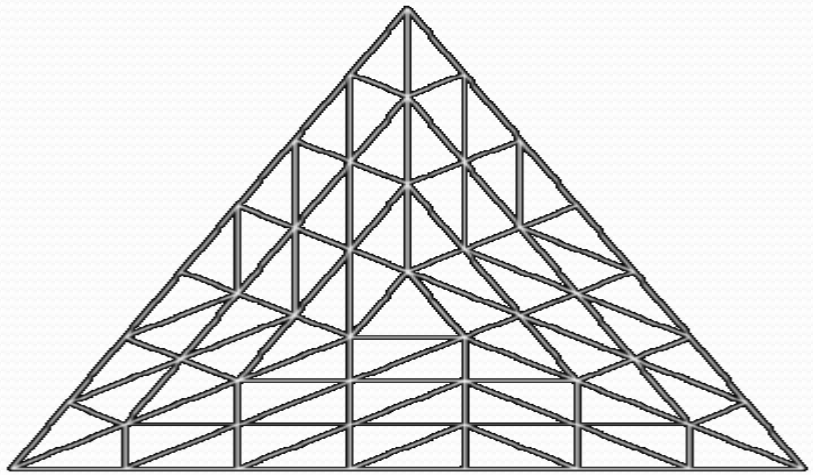
Tessellator

- Uses tessellation levels to decompose a patch into a new set of primitive
- Each vertex is assigned a (u, v) or (u, v, w) coordinate



The patch primitive

- The input vertices are often referred to as *control points*
- Arbitrary number of vertices (1 to 32)
  - `glPatchParameteri(GL_PATCH_VERTICES, patchVCount)`
- Only primitive type allowed when a tessellation control shaderis active
- No implied geometric ordering



Tessellation Control Shader(TCS)

- Output of a tessellation control shader is a new patch
  - i.e., a new collection of vertices
- Responsible for generating:
  - The per-patch inner and outer tessellation factors
    - Computes LOD per patch
      - gl_TessLevelOuter[4]
      - gl_TessLevelInner[2]
  - The position and other attributes for each output control point
  - Per-patch user-defined varyings
- Runs once for each vertex
- Patch discarded if
  - `gl_TessLevelOuter[x] <= 0;// (Usefullfor Culling)`
  - `gl_TessLevelOuter[x] = NaN;//`
- Optional
  - If not present tessellation level will be set to their default value
  - Default value can be changed using:
    - `glPatchParameterfv(GL_PATCH_DEFAULT_OUTER_LEVEL, outerLevels)`
    - `glPatchParameterfv(GL_PATCH_DEFAULT_INNER_LEVEL, innerLevels)`

Tessellation Control Shader: Sample

```
layout(vertices = 3) out;
uniform float tessLevelOuter;
uniform float tessLevelInner;
void main()
{
   if (gl_InvocationID == 0) {
      gl_TessLevelOuter[0] = tessLevelOuter;
      gl_TessLevelOuter[1] = tessLevelOuter
      gl_TessLevelOuter[2] = tessLevelOuter;
      gl_TessLevelInner[0] = tessLevelInner;
   }
   gl_out[gl_InvocationID].gl_Position= gl_in[gl_InvocationID].gl_Position;
}
```

Name

gl_InvocationID — contains the invocation index of the current shader

Declaration

```
in int gl_InvocationID ;
```

Description

In the tessellation control language, *gl_InvocationID* contains the index of the output patch vertex assigned to the shader invocation. It is assigned an integer value in the range [0, N-1] where N is the number of output patch vertices.

In the geometry language, *gl_InvocationID* identifies the invocation number assigned to the geometry shader invocation. It is assigned an integer value in the range [0, N-1] where N is the number of geometry shader invocations per primitive.

Version Support

[1] Versions 1.50 to 3.30 - geometry shaders only.

	OpenGL Shading Language Version									
Variable Name	**1.10**	**1.20**	**1.30**	**1.40**	**1.50**	**3.30**	**4.00**	**4.10**	**4.20**	**4.30**
gl_InvocationID	-	-	-	-	✔	✔	✔	✔	✔	✔

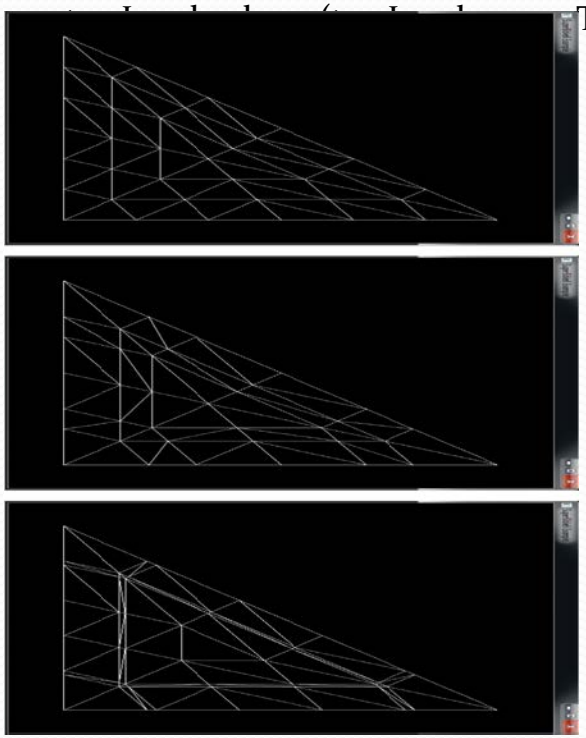
1. Determine that it is a first invocation by
    1. checking that gl_InvocationID is zero
2. Calculate the tessellation levels for the whole patch

Tessellation Evaluation Shader(TES)

- Compute the position of each vertex produced by the tessellator
- Control the tessellation pattern
- Can specify orientation of generated triangles
  - ccw(default)
  - cw
- Capable of generating points instead of lines or triangles
  - point_mode

Tessellation Subdivision Modes

- equal_spacing
  - tessLevel= clamp(tessLevel, 1, maxTessLevel)
    - Rounded to nearest integer
- fractional_even_spacing
  - tessLevel= clamp(tessLevel, 2, maxTessLevel)
    - Rounded to next even integer
- fractional_odd_spacing
  - TessLevel-1)



- **Example:**
  - Inner and outer tessellation factors are set to 5.3.
  - equal_spacing mode
    - the number of segments along each of the outer edges of the triangle is 6
      - the next integer after 5.3.
  - fractional_even_spacing spacing
    - there are 4 equal-sized segments
      - 4 is the next lower even integer to 5.3)
      - 2 additional smaller segments.
  - fractional_odd_spacing,
    - there are 5 equal-sized segments
      - (5 being the next lower odd integer to 5.3)
    - 2 very skinny segments that make up the rest.

Tessellation Primitive Modes

- Use to determine how OpenGL breaks up patches into primitives before passing them on to rasterization.
- Set using an input **layout** qualifier in the tessellation evaluation shader
- One of
  - **quads**, **triangles**, or **isolines**.
- Controls the form of the primitives produced by
  - the tessellator,
  - the interpretation of the gl_TessCoord input variable in the TES.

Tessellation Evaluation Shader: Sample

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

//use viewmat and projmat;

void main()
{
   gl_Position= vec4(gl_In[0].gl_Position.xyz* gl_TessCoord.x+
        gl_In[1].gl_Position.xyz* gl_TessCoord.y+
        gl_In[2].gl_Position.xyz* gl_TessCoord.z+
        1.0);
}
```

OpenGL Tessellation -Setup

```
char *tcsSource; // Null terminated string
GLuinttcs= glCreateShader(GL_TESS_CONTROL_SHADER);
glShaderSource(tcs, 1, tcsSource, NULL);
glCompileShader(tcs);
glAttachShader(program, tcs);

char* tesSource; // Null terminated string
GLuinttes= glCreateShader(GL_TESS_EVALUATION_SHADER);
glShaderSource(tes, 1, tesSource, NULL);
glCompileShader(tes);
glAttachShader(program, tes);
glLinkProgram(program);
```

glPatchParameteri(GL_PATCH_VERTICES, n); // tell OpenGL that every patch has n verts

glDrawArrays(GL_PATCHES, firstVert, vertCount); // draw a bunch of patches

Water-tight tessellation

- Cracks may occur due to floating point precision
  - a + b + c != c + b + a
- Use GLSL **precise** qualifier
  - Ensure s that computations are done in their stated order

```
precise out vec4 position;
out vec4 position;
precise position; // make existing variable precise
```

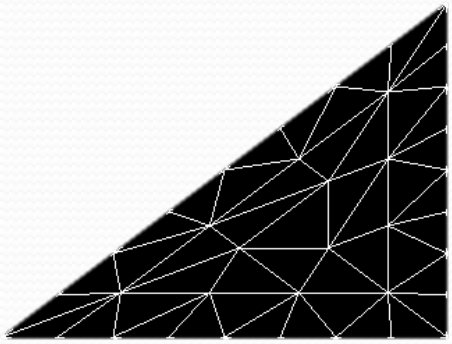
Notes

- Don’t forget to switch to GL_TRIANGLES when disabling tessellation
- **Frutum & Occlusion Culling**
  - Consider using the Tessellation Control Shaderto Cull patches not in frustum
    - gl_TessLevelOuter[x] = 0
    - Don’t forget to take displacement into consideration
  - Don’t render occluded patches
    - Use occlusion queries

1nd Task

Simple tessellation shaders

- Create simple triangle tessellation control shader example with
  - gl_TessLevelInner[x] = 5.0;
  - gl_TessLevelOuter[x] = 8.0;
- Create simple triangle tessellation evaluation shader
- Pass model view and projection matrices for the model to rotate the model from your application
- Bonus:
  - Pass inner and outer parameters from openGL application
  - Create dynamic tessellation change
  - Do the same for quade
  - Adopt to 3d models – such as icosahedron
    - http://prideout.net/blog/?p=48



2nd Task

Putting it all together - **Terrain Rendering**

Setup and Vertex Array

- We will generate a grid of 64 × 64 patches
- A possible solution is by generating coordinates vertex shader :

```
const vec4 vertices[] = vec4[](vec4(-0.5, 0.0, -0.5, 1.0),
                               vec4( 0.5, 0.0, -0.5, 1.0),
                               vec4(-0.5, 0.0, 0.5, 1.0),
                               vec4( 0.5, 0.0, 0.5, 1.0));
```

  - You can access each vertex with gl_VertexID
  - Use glDrawArraysInstanced(GL_PATCHES, 0, 4, 64 * 64) instead of glDrawArrays
  - `Use glPatchParameteri(GL_PATCH_VERTICES, 16)`
- *x* and *y* offsets for the patch are calculated by
  - gl_InstanceID modulo 64
  - gl_InstanceID divided by 64

```
int x = gl_InstanceID & 63;
int y = gl_InstanceID >> 6;
vec2 offs = vec2(x, y);
vs_out.tc = (vertices[gl_VertexID].xz + offs + vec2(0.5)) / 64.0;
gl_Position = vertices[gl_VertexID] + vec4(float(x - 32), 0.0,
        float(y - 32), 0.0);
```

- Output will be defined as follows:

```
out VS_OUT {
          vec2 tc;
} vs_out;
```

Name

gl_VertexID — contains the index of the current vertex

Declaration

in highp int gl_VertexID ;

Description

gl_VertexID is a vertex language input variable that holds an integer index for the vertex. The index is impliclty generated by **glDrawArrays** and other commands that do not reference the content of the GL_ELEMENT_ARRAY_BUFFER, or explicitly generated from the content of the GL_ELEMENT_ARRAY_BUFFERby commands such as **glDrawElements**.

Version Support

	OpenGL ES Shading Language Version	
Variable Name	**1.00**	**3.00**
gl_VertexID	-	✔

Name

gl_InstanceID — contains the index of the current primitive in an instanced draw command

Declaration

in highp int gl_InstanceID ;

Description

gl_InstanceID is a vertex language input variable that holds the integer index of the current primitive in an instanced draw command such as**glDrawArraysInstanced**. If the current primitive does not originate from an instanced draw command, the value of *gl_InstanceID* is zero.

Version Support

	OpenGL ES Shading Language Version	
Variable Name	**1.00**	**3.00**
gl_VertexID	-	✔

Tessellation control shader

- Project the corners of the patch into normalized device coordinates
  - Multiply incoming coordinates by model-view-projection matrix
  - Divide each of the 4 points by their own .w component
- Check that all of the *z* coordinates of the projected control points are less than zero
  - Set the outer tessellation levels to zero
    - This is an optimization that culls entire patches that are behind the viewer
- Project 4 into xy plane by ignoring *z* components
- Calculate the length of each of the 4 edges of the patch in normalized device space
- Calculate the tessellation levels of each edge
  - As a function of its length by using a scale and bias
  - Set outer tessellation factors
    - By calculating the edge lengths in the horizontal or vertical directions
  - Set the inner tessellation factors to the minimum of the outer tessellation factors
    - By calculated the edge lengths in the horizontal or vertical directions

```
#version 430 core
layout (vertices = 4) out;
in VS_OUT { vec2 tc; } tcs_in[];
out TCS_OUT { vec2 tc; } tcs_out[];
uniform mat4 mvp;
void main(void){
    if (…){
        vec4 p0 =
        ..
        vec4 p3 =
        p0 /=
        ..
        p3 /= p3.w;
        if (…){
            gl_TessLevelOuter[0] =
            ..
            gl_TessLevelOuter[3] =
         }Else{
            float l0 = length(p2.xy -
``````````````````p0.xy)*16.0+1.0;
            float l1 =
            float l2 =
            float l3 =
            gl_TessLevelOuter[0] = l0;
            …
            gl_TessLevelInner[0] = min(l1, l3);
            gl_TessLevelInner[1] =
         }
    }
gl_out[gl_InvocationID].gl_Position =
      gl_in[gl_InvocationID].gl_Position;
tcs_out[gl_InvocationID].tc =
      tcs_in[gl_InvocationID].tc; }
```
```

Tessellation evaluation shader

- Calculate texture coordinate of the generated vertex
  - Linearly interpolating the texture coordinates from TCS
- Calculate position of the outgoing vertex
  - Interpolate the incoming control point positions
  - Uses texture coordinate to offset the vertex in the *y* direction
  - Multiply the result by the model-view-projection matrix
- Make sure you pass required info from your openGL application

```
#version 430 core

layout (quads, fractional_odd_spacing)in;

uniform sampler2D tex_displacement;
uniform mat4 mvp;
uniform float dmap_depth;

in TCS_OUT { vec2 tc; } tes_in[];
out TES_OUT { vec2 tc; } tes_out;

void main(void) {
    vec2 tc1 = mix(tes_in[0].tc, tes_in[1].tc,
                   gl_TessCoord.x);
    vec2 tc2 =

    vec2 tc = mix(tc2, tc1, gl_TessCoord.y);

    vec4 p1 = mix(gl_in[0].gl_Position,
    gl_in[1].gl_Position, gl_TessCoord.x);

    vec4 p2 =
    vec4 p =
    p.y += texture(tex_displacement, tc).r *
    dmap_depth;

    gl_Position = ;
    tes_out.tc = ;
}
```

Fragment shader

- Use texture coordinate from the TES to look up a color for the fragment.

```
#version 430 core out vec4 color;
uniform sampler2D tex_color;
in TES_OUT {
        vec2 tc;
} fs_in;
out  vec4 color;
void main(void) {
        color = texture(tex_color, fs_in.tc);
 }
```

Bonus

- Add lighting
- Add bump mapping
- Make dynamic scene


vancouver 2010