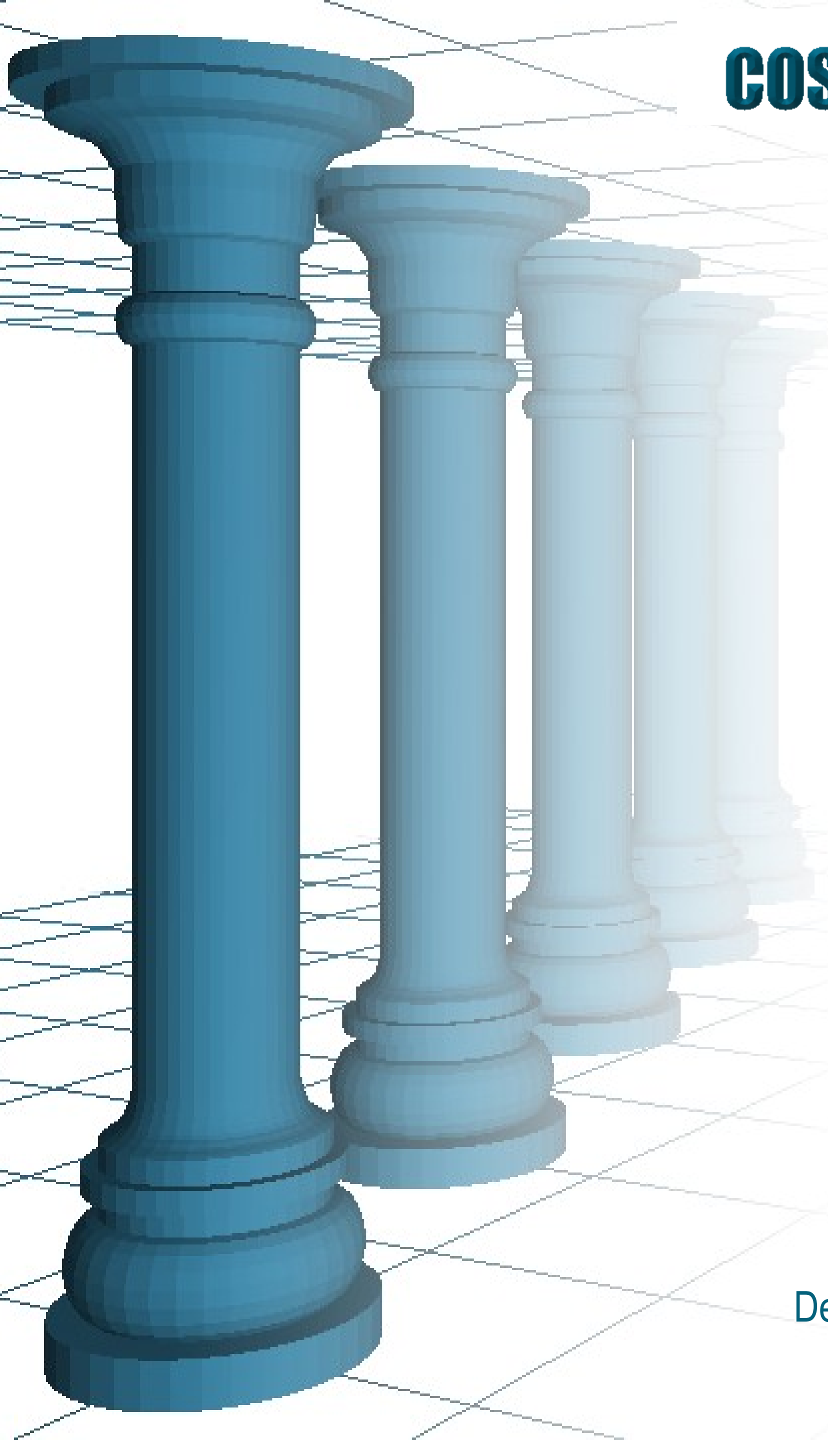


COSC363 Computer Graphics

11

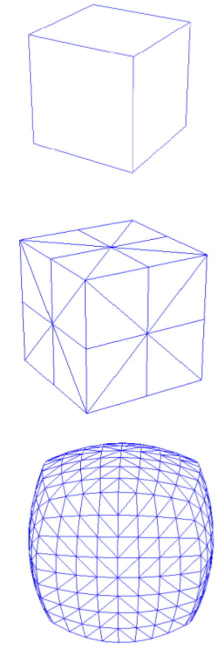
Tessellation Shaders

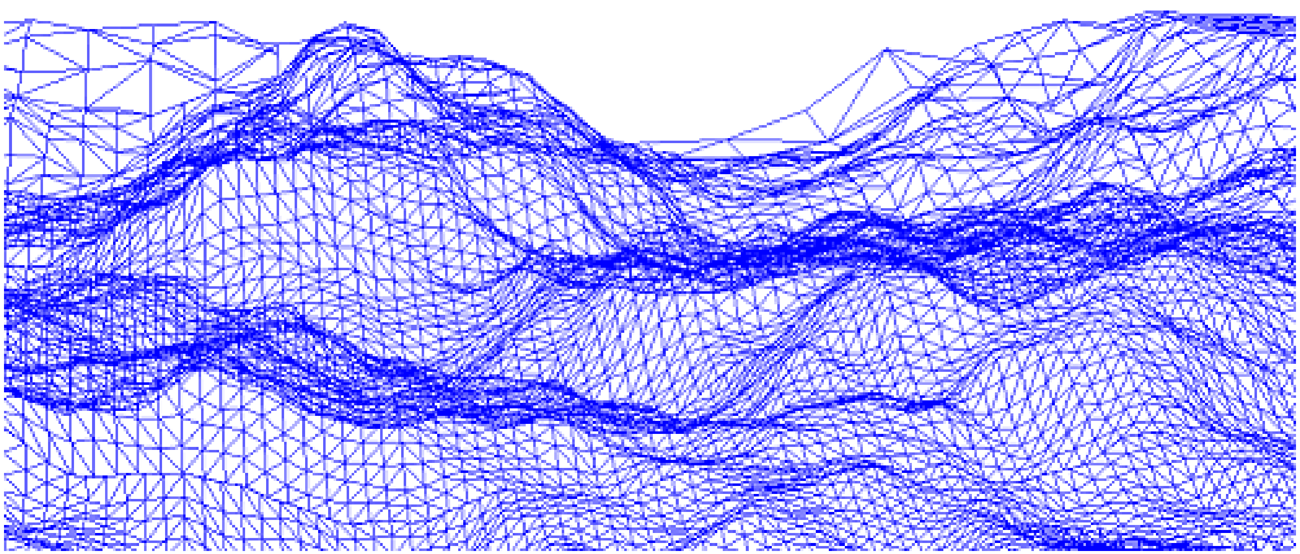
The birth place of terrains

R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Applications of Tessellation

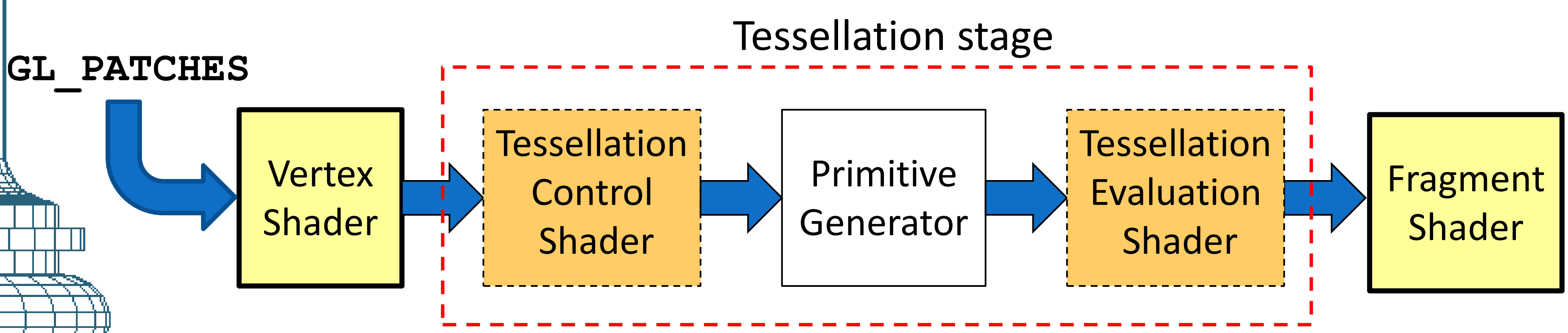
- Mesh subdivision
- Surface generation
- Continuous Level of Detail (CLOD)
- Real-time terrain rendering (Terrain LoD)
- Adaptive Mesh Refinement
- Mesh Morphing





Tessellation of Patches

- The tessellation stage of the OpenGL-4 pipeline can be used to generate a **mesh of triangles** based on vertices of a **patch** (a new geometric primitive).
- There are two shading stages used in tessellation:
  - Tessellation controller (optional): Sets tessellation parameters and any additional patch vertices.
  - Tessellation evaluator: Positions the vertices of the generated mesh on the patch using mapping equations defined by user.



Linear Blending Functions

- The quad domain mapping method combines a set of vertices P_0, P_1, P_2, P_3 using linear blending functions in u, v.

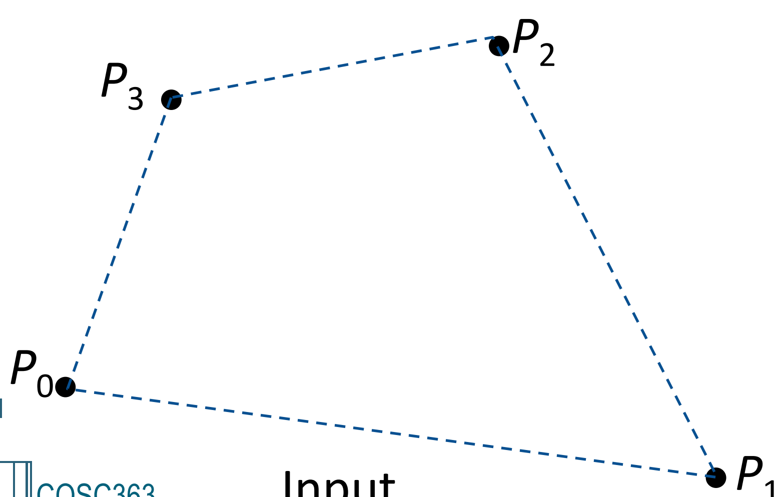
$$\boxed{\boldsymbol{Q} = (1-v)\,(1-u)\,\boldsymbol{P_0} + (1-v)\,u\,\boldsymbol{P_1} + v\,(1-u)\,\boldsymbol{P_3} + v\,u\,\boldsymbol{P_2}}$$

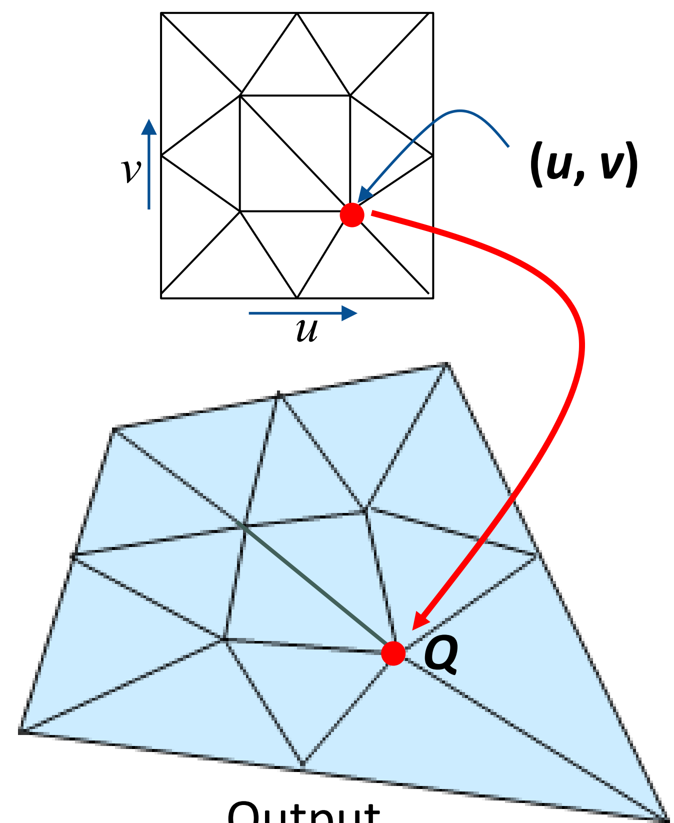
- **Blending functions**: $(1-u)$, u, $(1-v)$, v.

$0 \le \boldsymbol{u} \le 1$
$0 \le \boldsymbol{v} \le 1$

- **Patch vertices**: P_0, P_1, P_2, P_3

The patch vertices define the shape of the tessellated surface





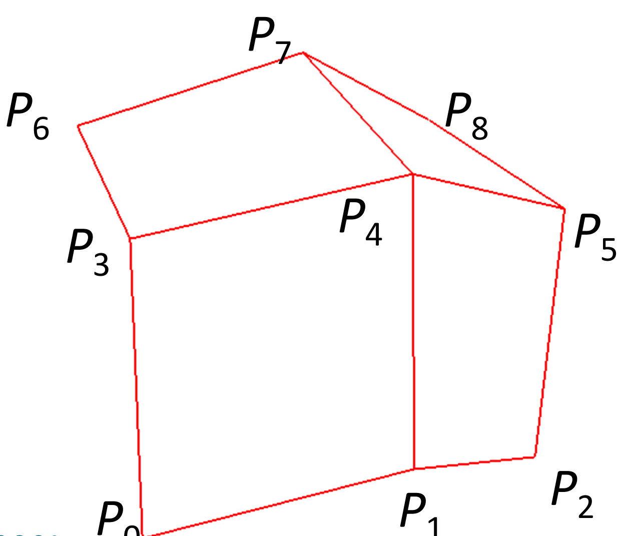
Higher Order Blending Functions

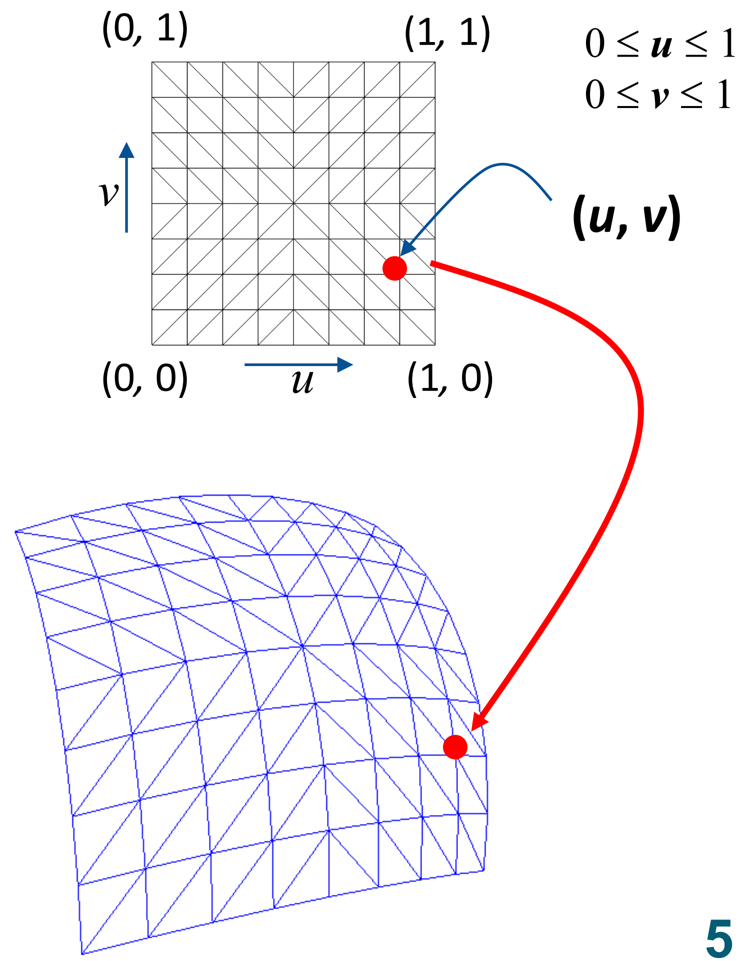
- Example:

$$
\begin{aligned}
\boldsymbol{Q} = \; & (1-v)^2 \; \{ (1-u)^2 P_0 + 2u(1-u) \, P_1 + u^2 P_2 \} \\
& + 2v(1-v) \; \{ (1-u)^2 P_3 + 2u(1-u) \, P_4 + u^2 P_5 \} \\
& + \; v^2 \; \{ (1-u)^2 P_6 + 2u(1-u) \, P_7 + u^2 P_8 \}
\end{aligned}
$$

- **Blending functions**:
  $(1-u)^2$, $2u(1-u)$, u^2, $(1-v)^2$, $2v(1-v)$, v^2.

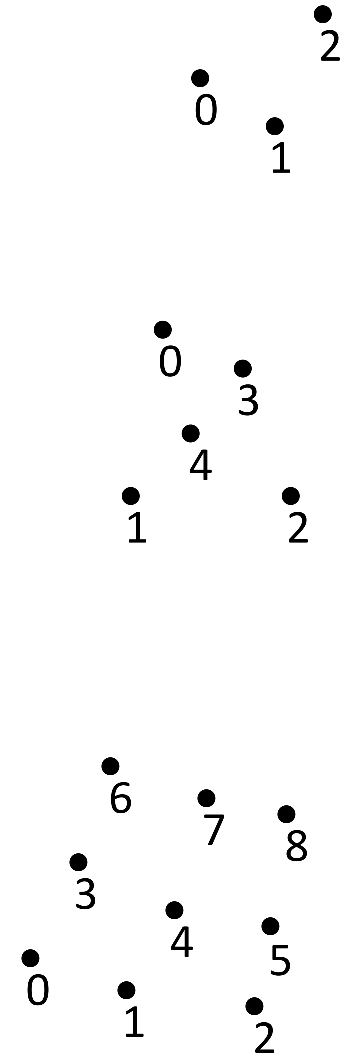
- **Patch vertices**: $P_0, \ldots, P_8$





Patches

Examples of patches



- A patch is simply an ordered list of vertices, the order determined by the user.
- A patch is cannot be directly rendered (it is not a renderable primitive)
- If the tessellation stage is active, the input must be a patch.
- For a patch, the rendering command is

```
glDrawArrays(GL_PATCHES, 0, n);
```

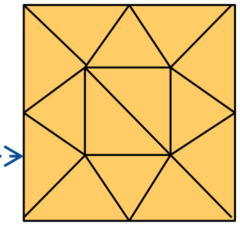
or,
```
glDrawElements(GL_PATCHES,..);
```

- You should also specify in your application, the number of vertices in each patch:

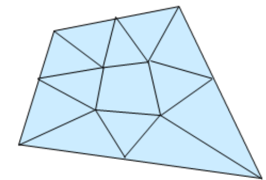
```
glPatchParameteri(GL_PATCH_VERTICES, 9);
```

From Patches to Triangles

Tessellation Coordinates
u, v



Vertices of Triangles



A simple pass-through shader

Modify patch vertices, if needed. Specify tessellation levels

Patches

Vertex Shader → Tessellation Control Shader → Primitive Generator → Tessellation Evaluation Shader →

- The primitive generator can output only **triangles**.
- The triangles form a tessellation of either a square or a triangle domain.
- The vertices of every triangle in the tessellation will have normalized coordinates.

The evaluation shader converts the primitive vertices (*u*, *v*) to 3D points using use-defined functions, and outputs them in clip coordinate space.

Vertex Shader

- This vertex shader does nothing! It simply passes the patch vertices to the tessellation control shader (TCS)
- Since a patch is not a renerable primitve, its vertices are <u>not</u> converted to clip coordinate space.

Input
patch vertices

```
#version 330

layout (location = 0) in vec4 position;

void main()
{
    gl_Position = position;
}
```

Tessellation Levels

- The amount of tessellation of a domain (quad or triangle) is determined by tessellation levels.

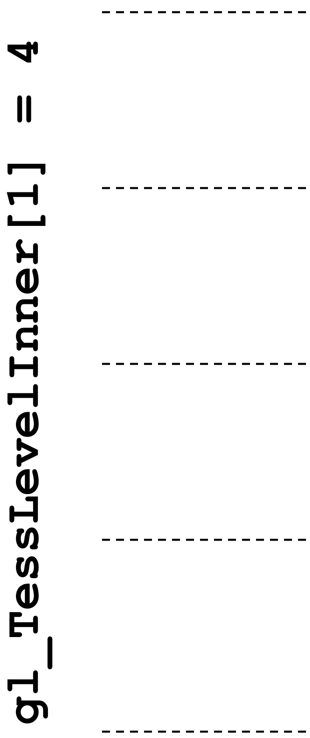
- **Outer tessellation level**:
  - 4 values (one for each side of the quad; for a triangle the last value is 0) stored in arrays

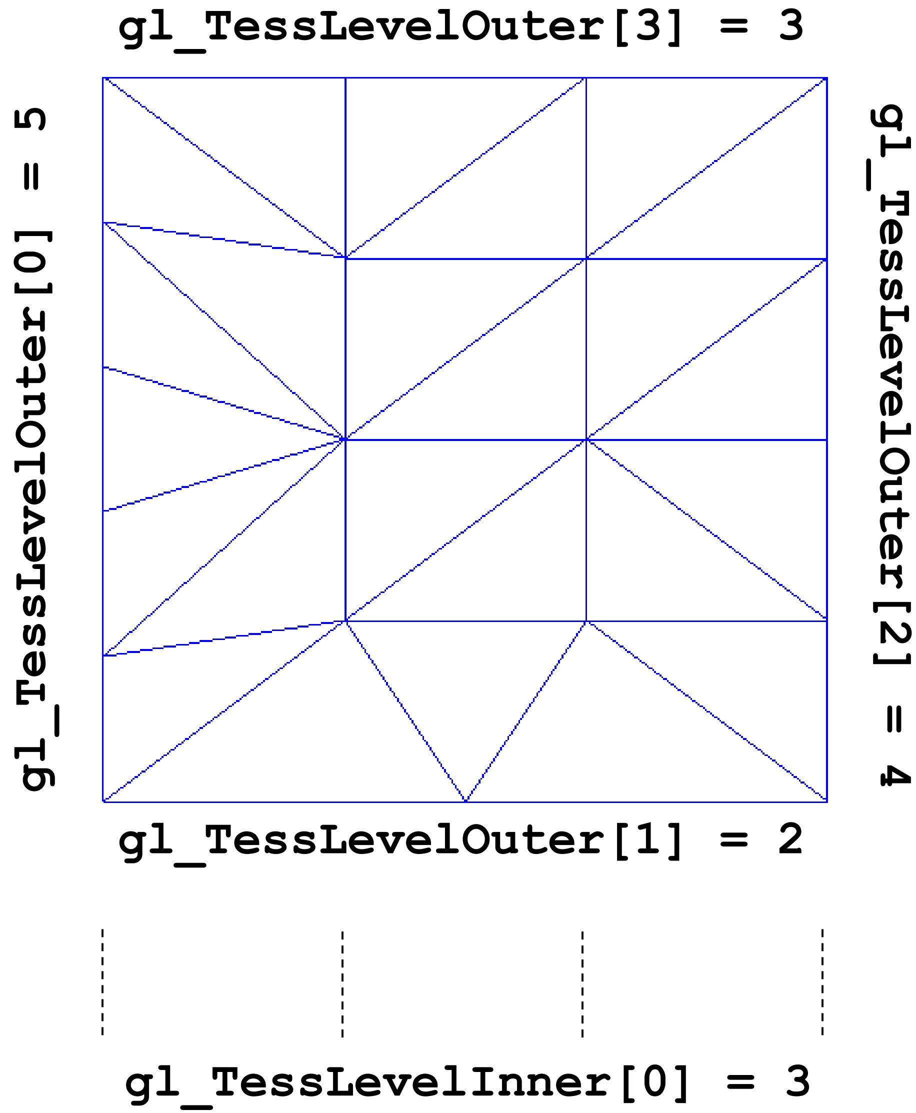
    `gl_TesslevelOuter[0]... gl_TesslevelOuter[3]`

- **Inner tessellation level**:
  - 2 values for a quad, 1 for a triangle stored in arrays

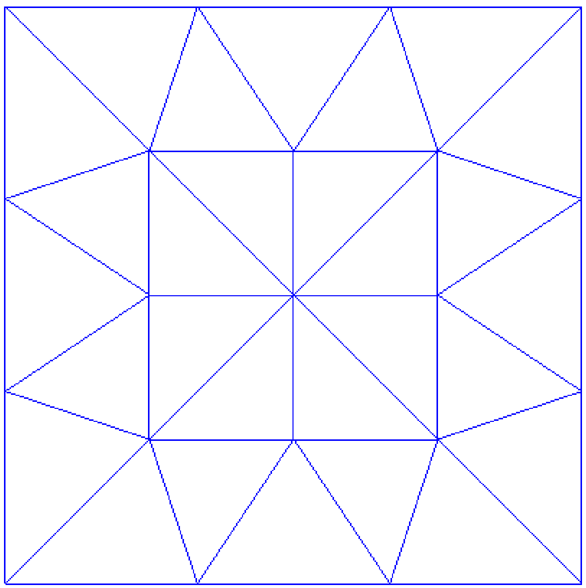
    `gl_TesslevelInner[0], gl_TesslevelInner[1]`

Tessellation Levels: Quad



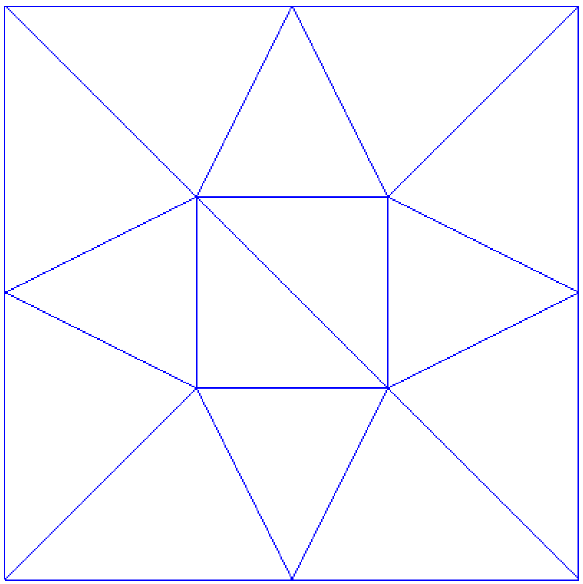


Tessellation Levels: Quad

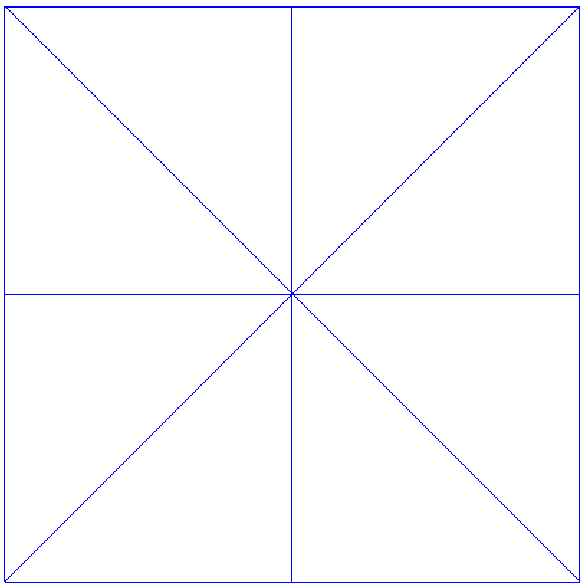


3333; 44

Outer   Inner



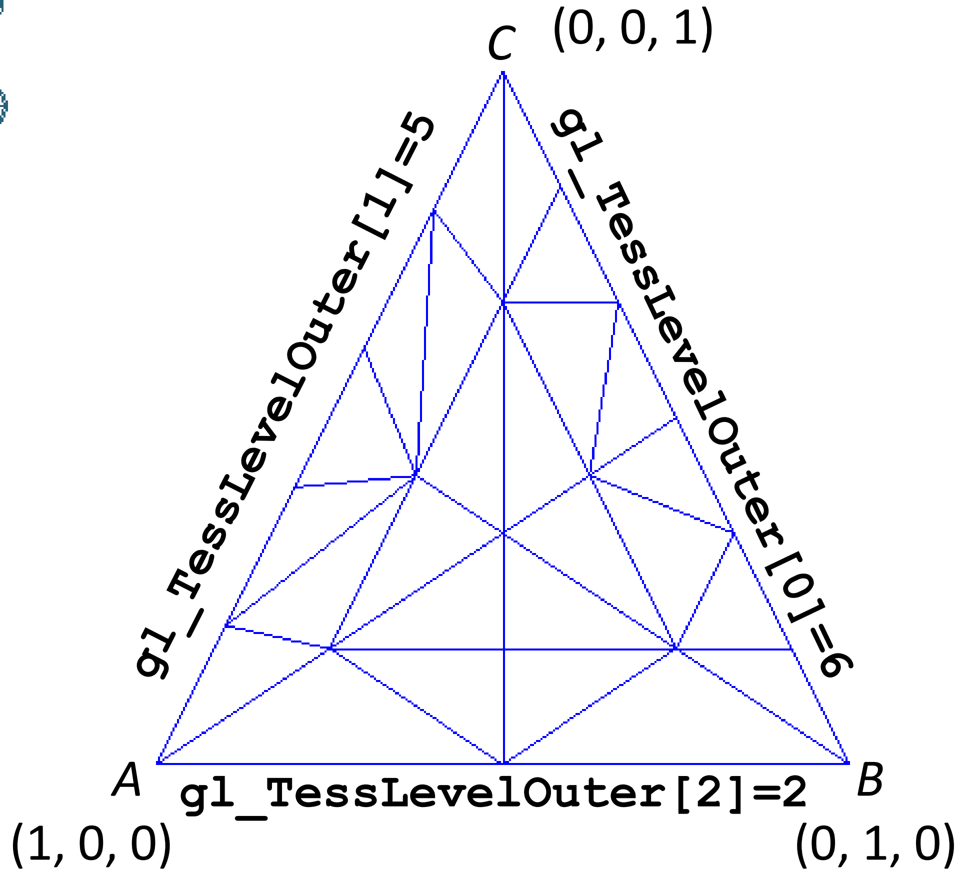
2222; 33



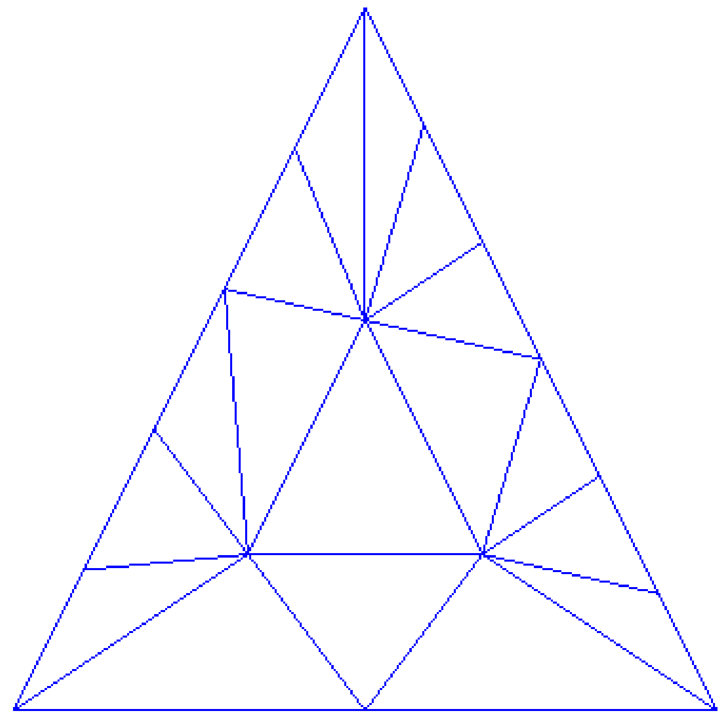
2222; 22

A quad domain has 4 outer tessellation levels and 2 inner levels

Tessellation Levels: Triangle



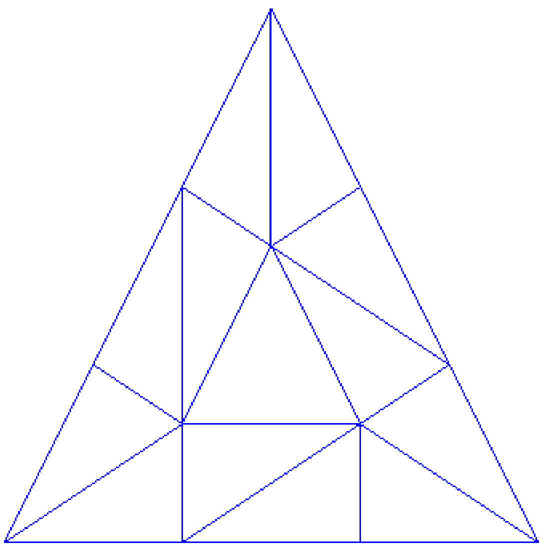
`gl_TessLevelInner[0] = 4`



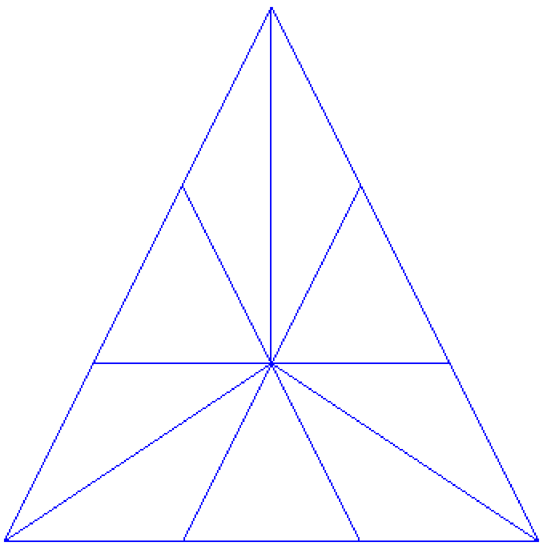
`gl_TessLevelInner[0] = 3`

A triangle domain has 3 outer tessellation levels and 1 inner level

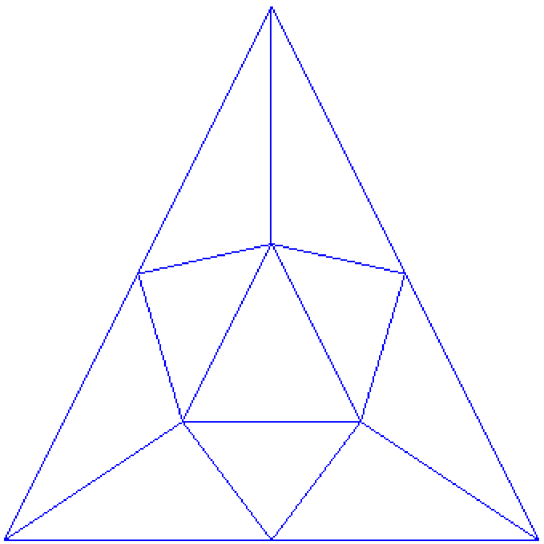
Tessellation Levels: Triangle



$\underbrace{3330;}_{\text{Outer}}\ \underbrace{30}_{\text{Inner}}$



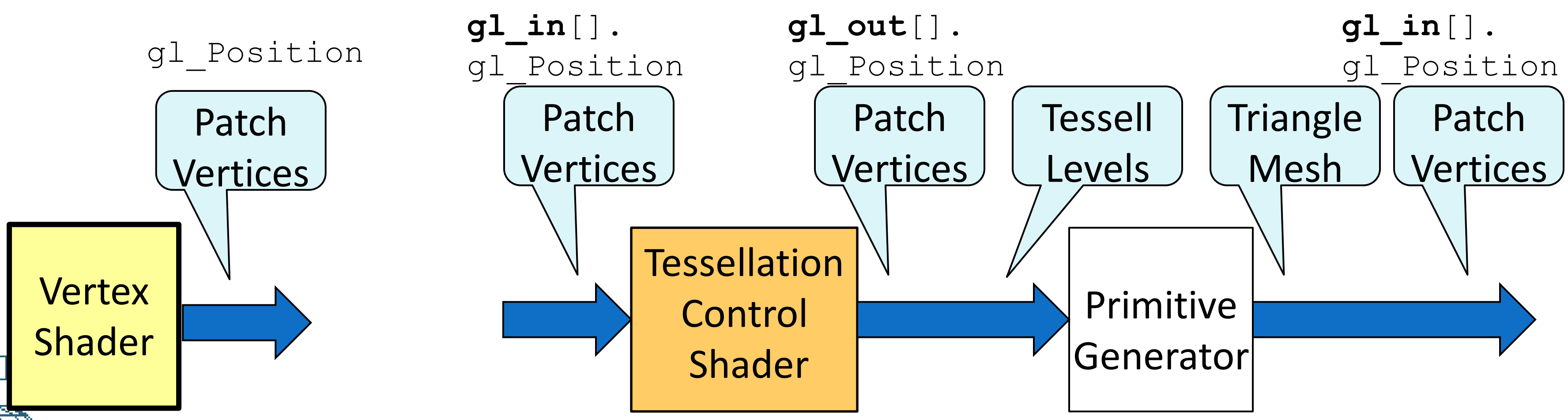
3330; 20



2220; 30

Tessellation Control Shader (TCS)

- The tessellation control shader is commonly used to set the inner and outer tessellation levels.
- Optionally, the shader can also create new or remove existing patch vertices. *All* patch vertices are available inside the shader in an array.
- The tessellation control shader will execute once for each output patch vertex.



Tessellation Control Shader: Example

output patch vertices

Index of the current out vertex

```
#version 400
layout(vertices = 9) out;

void main()
{
    gl_out[gl_InvocationID].gl_Position
        = gl_in[gl_InvocationID].gl_Position;
    gl_TessLevelOuter[0] = 6;
    gl_TessLevelOuter[1] = 6;
    gl_TessLevelOuter[2] = 6;
    gl_TessLevelOuter[3] = 6;
    gl_TessLevelInner[0] = 5;
    gl_TessLevelInner[1] = 5;
}
```

Tessellation Control Shader

- The tessellation control shader on the previous slide is a simple pass-through shader, and does not modify tessellation levels.
- Such pass-through control shaders may be omitted (bypassed). If the tessellation levels remain constant for all patches, they can be specified in the OpenGL application as follows:

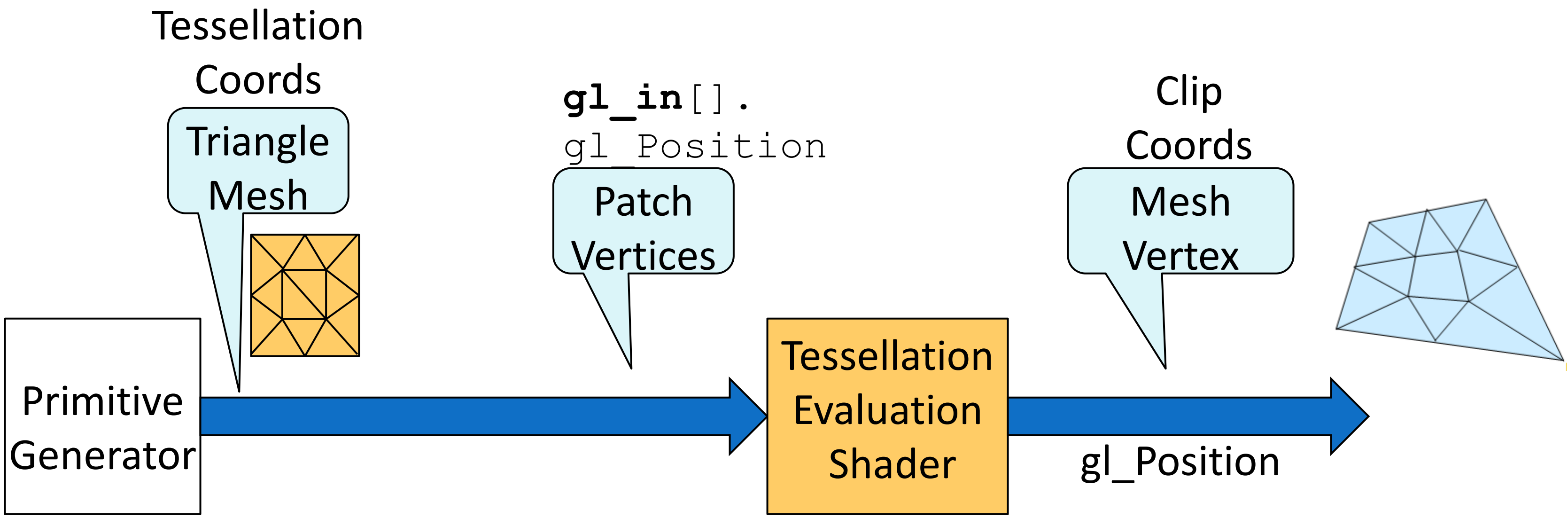
```
float outer[4] = {6, 6, 6, 6};
float inner[2] = {5, 5};
glPatchParameterfv(GL_PATCH_DEFAULT_OUTER_LEVEL, outer);
glPatchParameterfv(GL_PATCH_DEFAULT_INNER_LEVEL, inner);
```

From Patches to Triangles

The primitive generator outputs a **mesh of triangles** having the following characteristics:

- The pattern of tessellation is based on the “inner” and “outer” tessellation levels specified by the user.
- The pattern of tessellation also depends on the type of the domain (quad or triangle).
- The output primitive at this stage is always triangles.
- The vertex coordinates of the generated triangles will have values in the range [0, 1].
- The Tessellation Evaluation Shader operates on *one* mesh vertex (*u*, *v*) for a quad domain or (*u*, *v*, *w*) for a triangle domain. The evaluation shader acts like a vertex shader for the vertices emitted by the primitive generator.

Tessellation Evaluation Shader



- The tessellation evaluator repositions each mesh vertex (*u*, *v*) using patch vertices, and outputs them in clip coordinates.
- The evaluation shader executes once for each input mesh vertex.

Tessellation Evaluation Shader (Quad Domain)

```
#version 400
layout(quads, equal_spacing, ccw) in;
uniform mat4 mvpMatrix;
vec4 posn;

void main()
{
    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    posn = (1-u)* (1-v) * gl_in[0].gl_Position
                + u * (1-v) * gl_in[1].gl_Position
                + u *  v    * gl_in[2].gl_Position
             + (1-u) * v    * gl_in[3].gl_Position;
    gl_Position = mvpMatrix * posn;
}
```

Domain

Tessellation coordinates of the tessellated mesh vertices

Patch vertices

See slide 4

Clip Coords

Tessellation Evaluation Shader (Triangle Domain)

```
#version 400
layout(triangles, equal_spacing, ccw) in;
uniform mat4 mvpMatrix;
vec4 posn;

void main()
{
    posn = gl_TessCoord.x * gl_in[0].gl_Position
         + gl_TessCoord.y * gl_in[1].gl_Position
         + gl_TessCoord.z * gl_in[2].gl_Position;

    gl_Position = mvpMatrix * posn;
}
```

Domain

Tessellation coords (barycentric)

Fragment Shader

- The tessellation evaluation shader receives only the tessellation coordinates for each vertex of the triangle mesh generated by the primitive generator. It does not receive any other vertex attributes.
- If surface normal vectors are not available, the primitives cannot be rendered using lighting equations.
- Therefore the fragment shader generally has a simple form:

```
#version 330

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

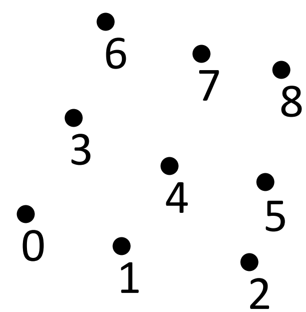
Applications of Tessellation Shaders

Bi-quadratic Bezier Surfaces

TCS

```
#version 400
layout(vertices = 9) out;

void main()
{
   gl_out[gl_InvocationID].gl_Position =
       gl_in[gl_InvocationID].gl_Position;
   for(int i = 0; i < 4; i++)
       gl_TessLevelOuter[i] = 8;
   for(int i = 0; i < 2; i++)
       gl_TessLevelInner[i] = 6;
}
```

Patch Vertices



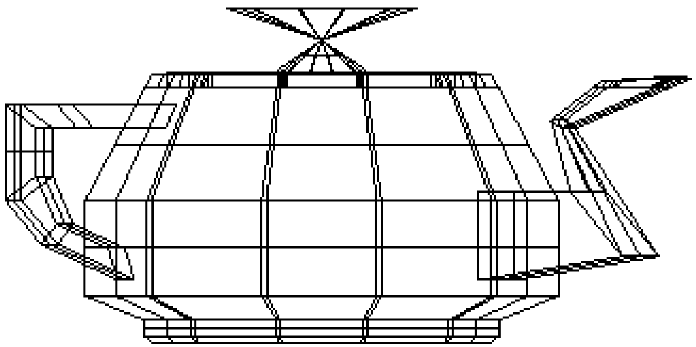
Bi-quadratic Bezier Surfaces

TES

```
#version 400
layout(quads, equal_spacing, ccw) in;
uniform mat4 mvpMatrix;

float Bez(int i, float u) {
    if(i == 0)      return (1-u)*(1-u);
    else if(i == 1) return 2*u*(1-u);
    else if(i == 2) return u*u;
    else return 0;
}
void main(){
    vec4 posn = vec4(0);
    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;
    for(int j = 0; j < 3; j++)
      for(int i = 0; i < 3; i++)
         posn += Bez(i, u) * Bez(j, v) *
                    gl_in[3*j+i].gl_Position;
    gl_Position = mvpMatrix * posn;
}
```

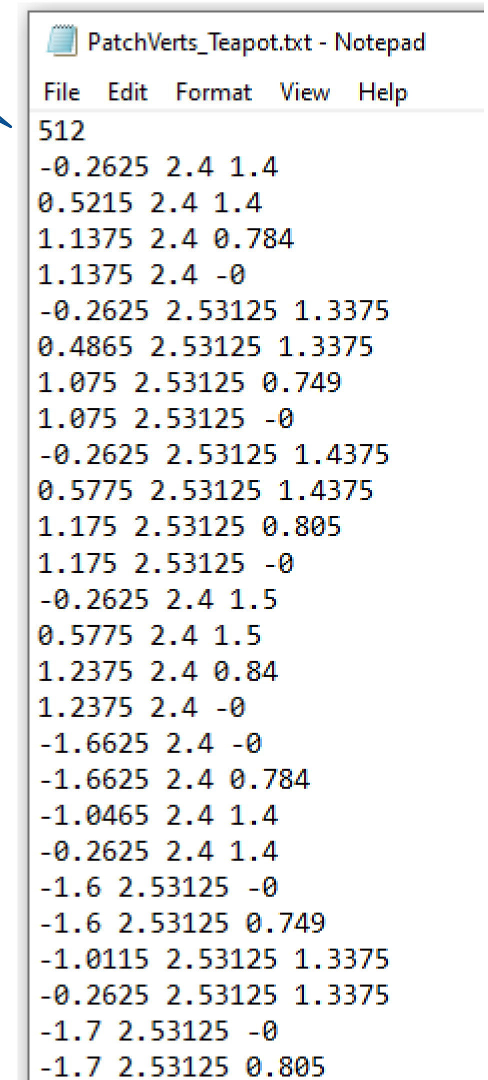
See slide 5

Bi-Cubic Bezier Surfaces



32 x 16

Patch Vertex Data

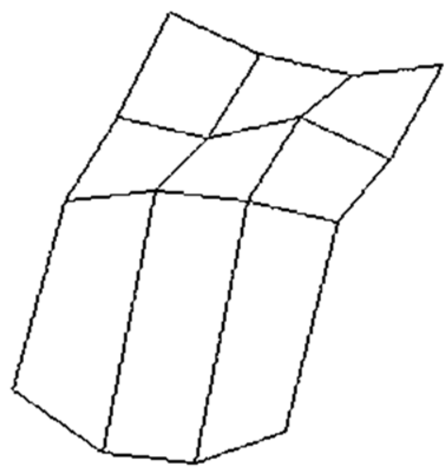


PatchVerts_Teapot.txt - Notepad

File Edit Format View Help

```
512
-0.2625 2.4 1.4
0.5215 2.4 1.4
1.1375 2.4 0.784
1.1375 2.4 -0
-0.2625 2.53125 1.3375
0.4865 2.53125 1.3375
1.075 2.53125 0.749
1.075 2.53125 -0
-0.2625 2.53125 1.4375
0.5775 2.53125 1.4375
1.175 2.53125 0.805
1.175 2.53125 -0
-0.2625 2.4 1.5
0.5775 2.4 1.5
1.2375 2.4 0.84
1.2375 2.4 -0
-1.6625 2.4 -0
-1.6625 2.4 0.784
-1.0465 2.4 1.4
-0.2625 2.4 1.4
-1.6 2.53125 -0
-1.6 2.53125 0.749
-1.0115 2.53125 1.3375
-0.2625 2.53125 1.3375
-1.7 2.53125 -0
-1.7 2.53125 0.805
```

- 32 Patches, each patch containing 16 vertices (4x4 control polygonal grid)



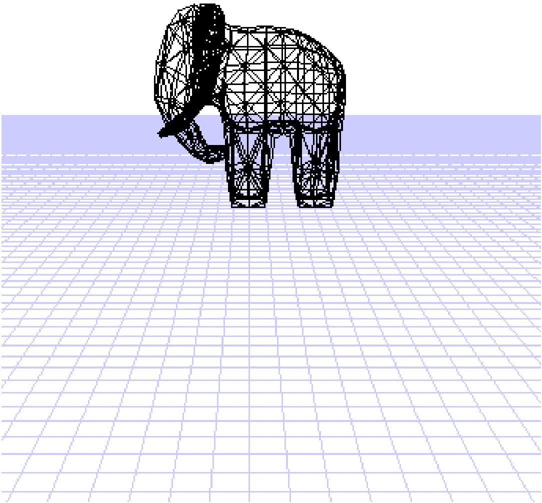
TES

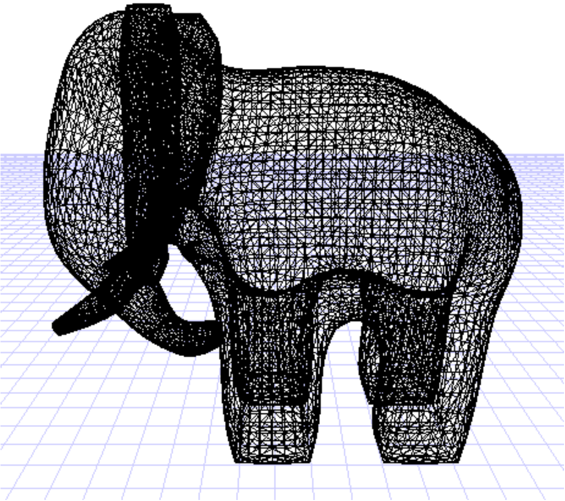
Bi-Cubic Bezier Surfaces

```
#version 400

layout(quads, equal_spacing, ccw) in;
uniform mat4 mvpMatrix;

void main()
{
    float u = gl_TessCoord.x;
    float v = gl_TessCoord.y;

    float Au = (1-u) * (1-u) * (1-u);
    float Bu = 3 * u * (1-u) * (1-u);
    float Cu = 3 * u * u * (1-u);
    float Du = u * u * u;
    float Av = (1-v) * (1-v) * (1-v);
    float Bv = 3 * v * (1-v) * (1-v);
    float Cv = 3 * v * v * (1-v);
    float Dv = v * v * v;

    vec4 posn;
    posn =  Au * (Av * gl_in[0].gl_Position +  Bv * gl_in[1].gl_Position
                + Cv * gl_in[2].gl_Position + Dv * gl_in[3].gl_Position )
          + Bu * (Av * gl_in[4].gl_Position +  Bv * gl_in[5].gl_Position
                 + Cv * gl_in[6].gl_Position + Dv * gl_in[7].gl_Position )
          + Cu * (Av * gl_in[8].gl_Position +  Bv * gl_in[9].gl_Position
                 + Cv * gl_in[10].gl_Position + Dv * gl_in[11].gl_Position )
          + Du * (Av * gl_in[12].gl_Position +  Bv * gl_in[13].gl_Position
                 + Cv * gl_in[14].gl_Position + Dv * gl_in[15].gl_Position ) ;
    gl_Position = mvpMatrix * posn;

}
```

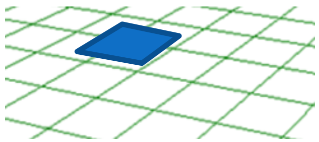
Dynamic Level of Detail

- Objects need not have a high resolution mesh when they are displayed within a small area on the screen.
- Dynamic Level of Detail (LOD) methods adjust the mesh resolution of a surface based on the distance from the camera.
- The tessellation control shader can be used to adjust the tessellation levels based on the distance of the current patch from the camera.





Terrain Generation



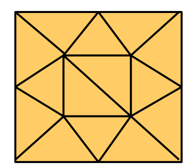
A patch consists of vertices of a small rectangular segment of a terrain grid

Vertex Shader → Tessellation Control Shader → Primitive Generator → Tessellation Evaluation Shader →

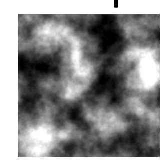
TCS

TES

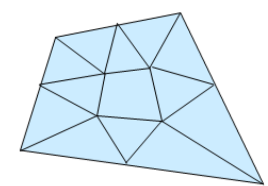
Tessellation Coordinates



Height Map



Vertices of Triangles



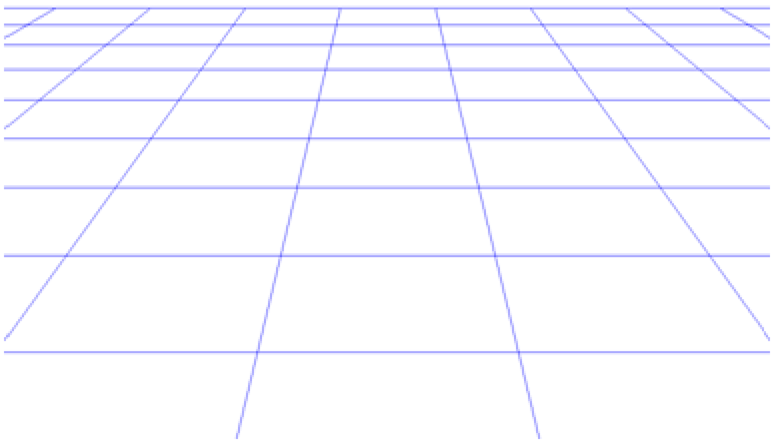
The vertices of the tessellated mesh are assigned height values based on a height map.

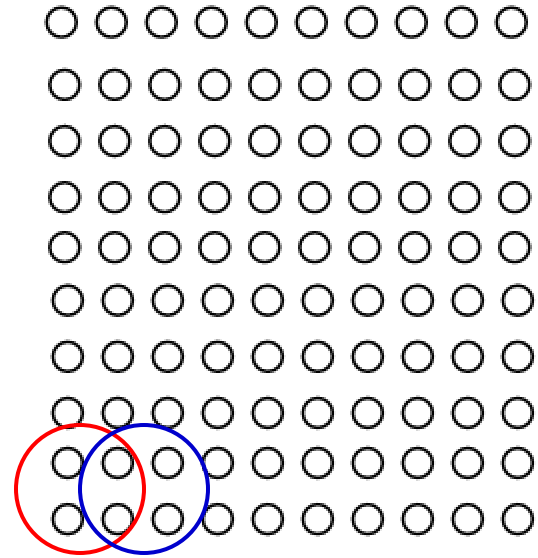
Terrain Construction

- Construct the terrain base. (Application)
- Tessellate the base to the required level. (Control Shader)
- Reposition (*u*, *v*) using patch coordinates. (Evaln Shader)
- Use a height map to modify vertices of the tessellated mesh. (Evaluation Shader)

- Geometry Shader (next topic):
- Perform lighting computation.
- Apply height based texture mapping.

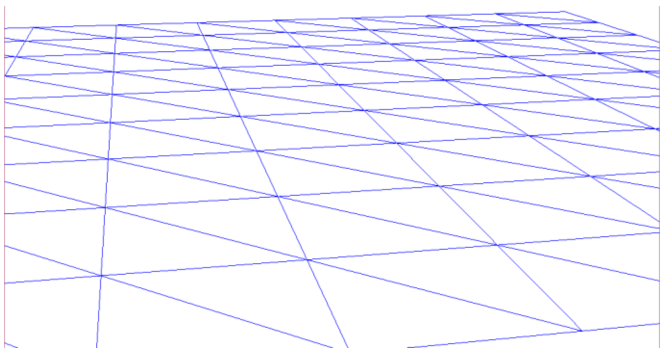
Step 1: Terrain Base



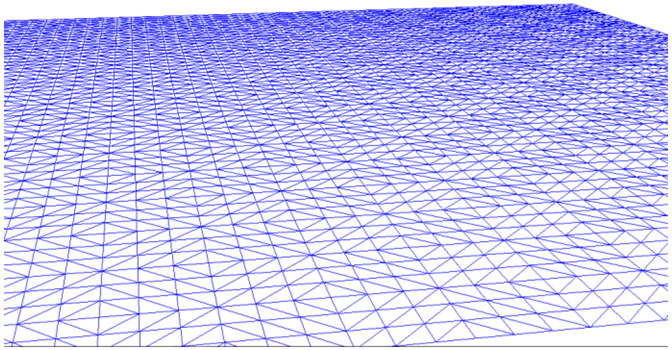


- The terrain base is represented by rectangular array of points on a large quad.
- Each patch has 4 vertices.
- Two VBO: Vertex coordinates, Element array.

Step 2: Tessellation



Tessellation levels: {1, 1, 1, 1; 1, 1}



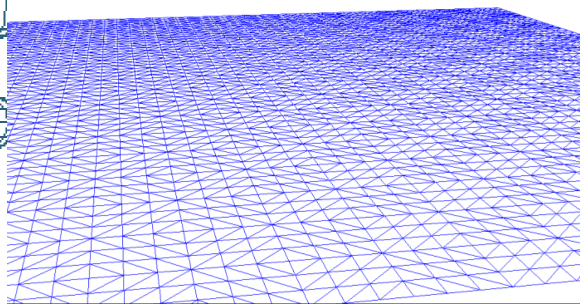
Tessellation levels: {6, 6, 6, 6; 6, 6}

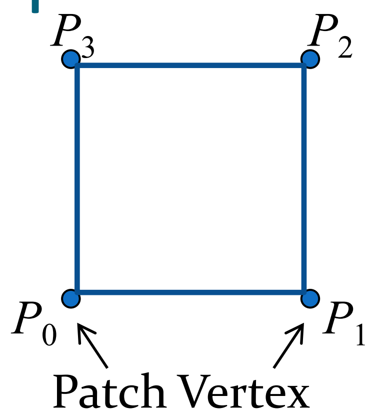
- A simple pass-thru vertex shader !

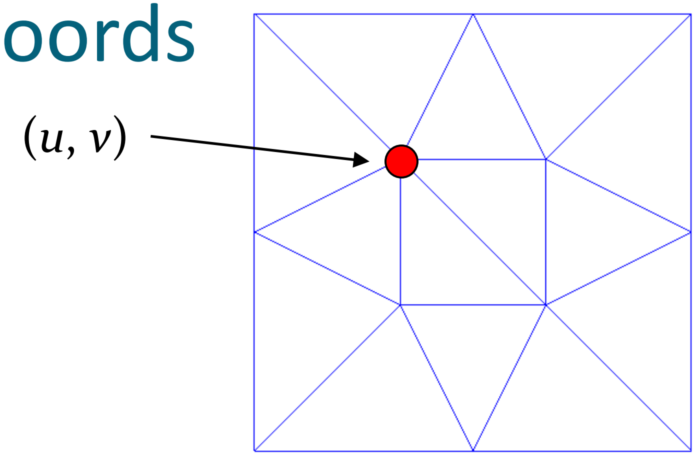
```
layout (location = 0) in vec4 position;
void main()
{
    gl_Position = position;
}
```

- The tessellation control shader receives 4 patch vertices and sets the outer and inner tessellation levels.

Step 3: Tess Coords







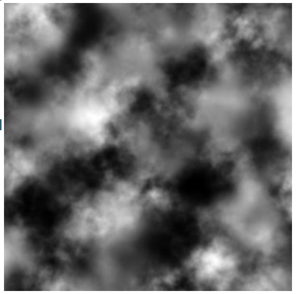
Tessellation Evaluation Shader:

- Receives 4 tessellation coordinates and 4 patch vertices.
- The current mesh vertex (u, v) is repositioned using patch vertices: u = gl_TessCoord.x;
  v = gl_TessCoord.y;

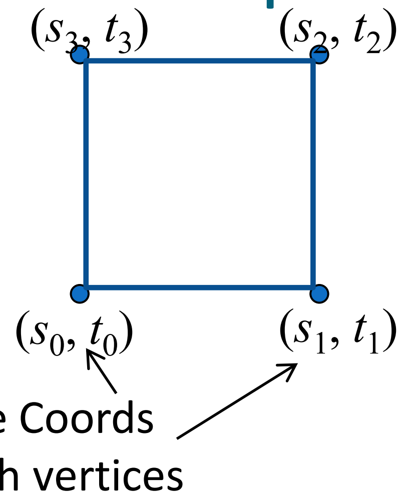
```
posn = (1-u)* (1-v)* gl_in[0].gl_Position
      +   u * (1-v)* gl_in[1].gl_Position
      +   u *  v   * gl_in[2].gl_Position
      +(1-u)*  v   * gl_in[3].gl_Position;
```

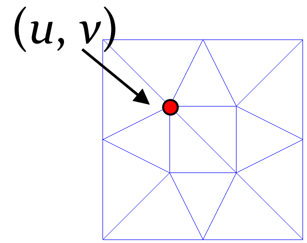
Patch Vertices

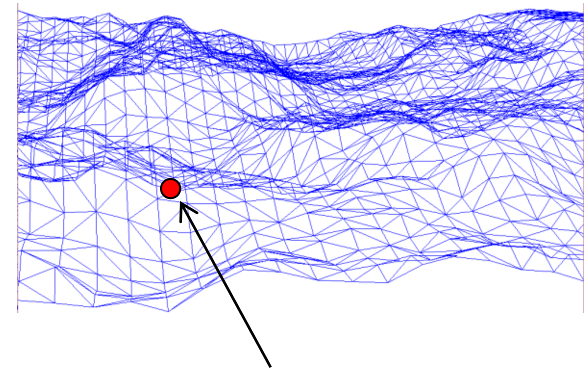
Step 4: Height Map



+







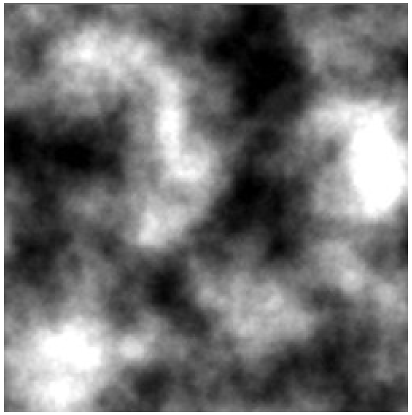
- The texture coordinates for the height map are computed mapping the (u, v) coordinates using the texture coordinates at the patch vertices.

```
tcoord = (1-u)* (1-v) * cont_texCoord[0]
             + u * (1-v) * cont_texCoord[1]
             + u *     v * cont_texCoord[2]
             + (1-u) * v * cont_texCoord[3];
```

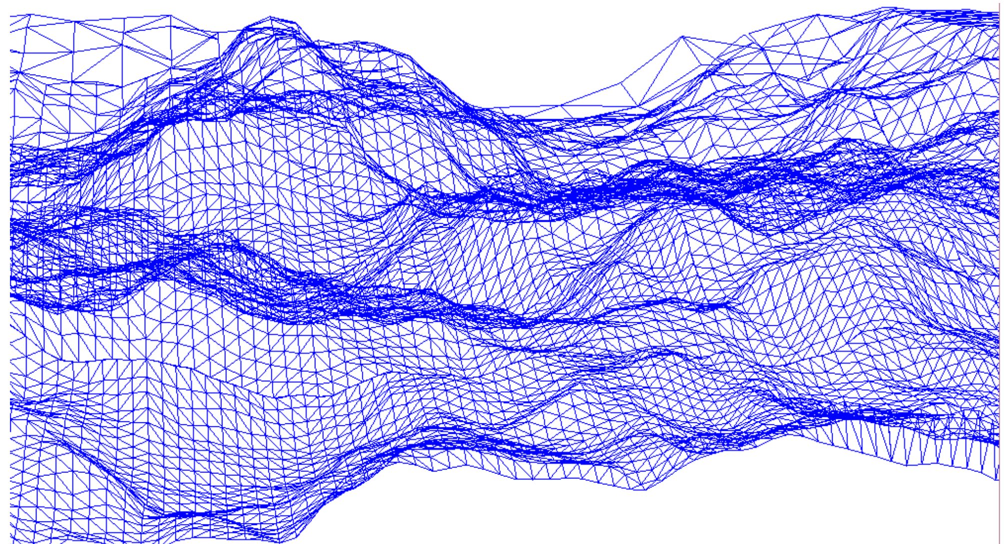
Texture Coordinates

- The height map is sampled using texture coordinates, and the y-value of the vertex (posn.y) is updated.

Terrain Rendering



Height Map



Geometry Shader