COSC363 Computer Graphics

12

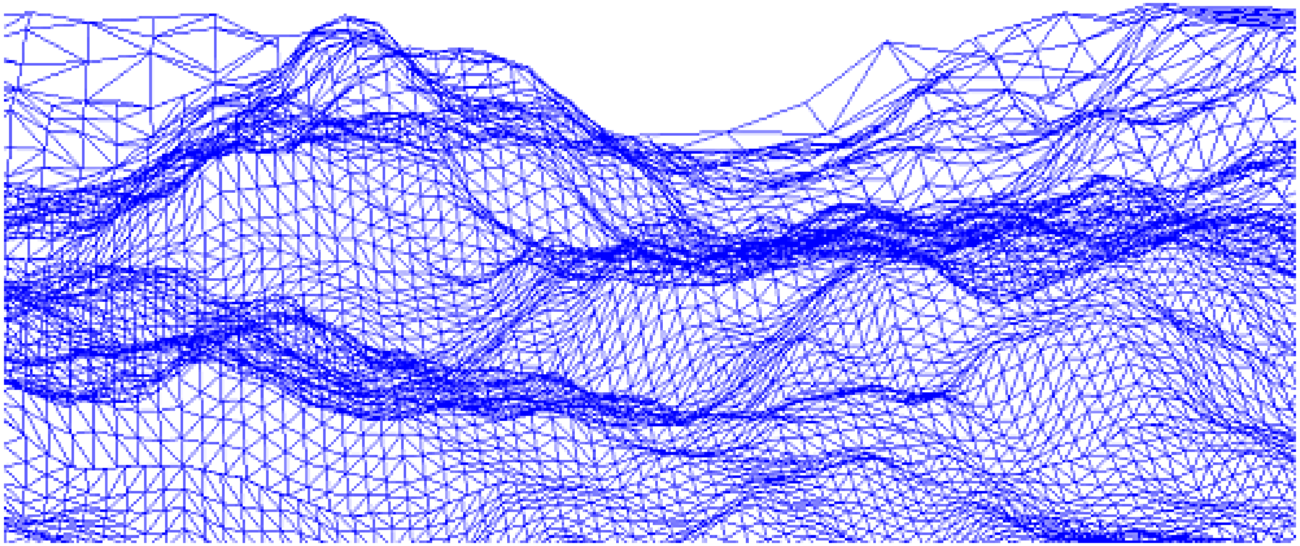
Tessellation Shaders

The birth place of terrains

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

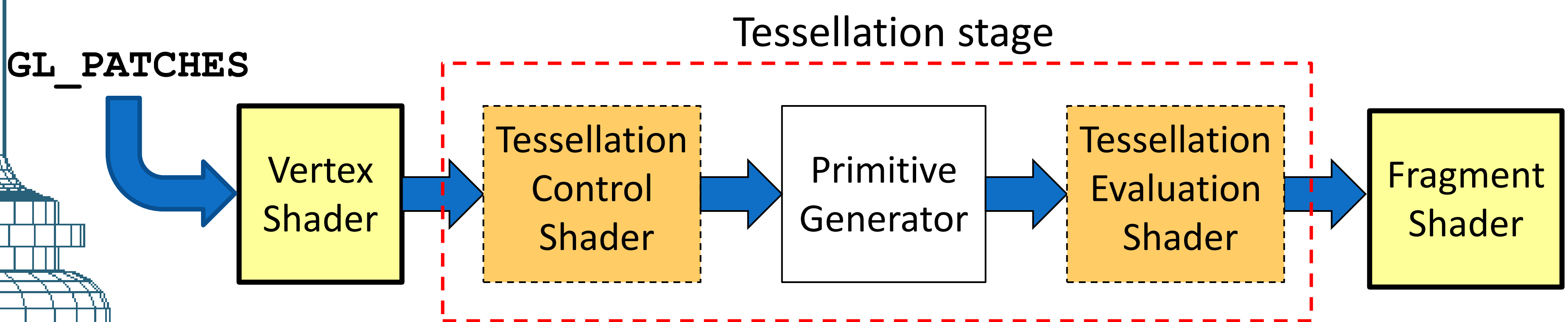
Applications of Tessellation

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



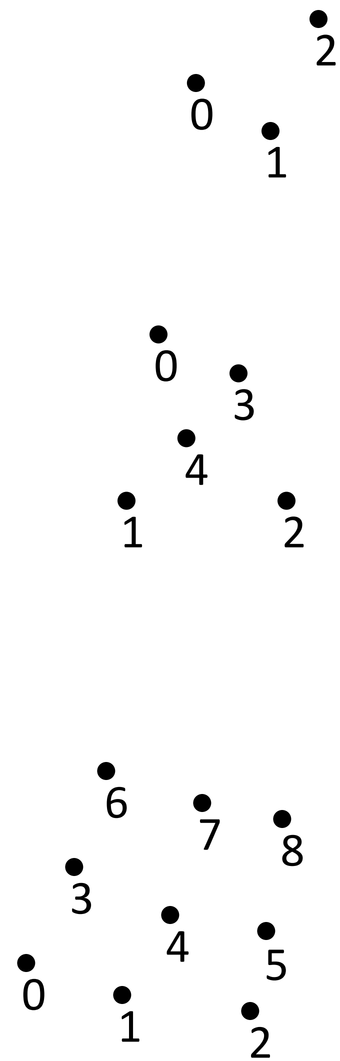
Tessellation of Patches

- 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.



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 without using the tessellation stage.
- 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);
```

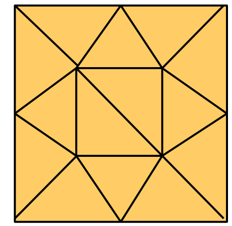
From Patches to Triangles

Patch

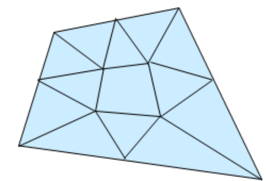
Modify/add/del patch vertices

Specify tessellation levels

Tessellation Coordinates



Vertices of Triangles



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.
- The vertices of every triangle in the tessellation will have normalized coordinates.

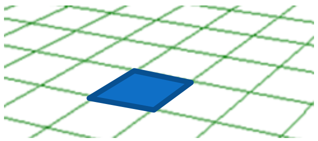
From Patches to Triangles

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. These parameters can be defined in the control shader. Alternatively, the default values to be used for all patches can be specified in the application.
- The pattern of tessellation also depends on the type of the domain (quad or triangle).
- The normalized coordinates of a tessellation are repositioned on a surface using the patch coordinates inside the tessellation evaluation shader. The output primitive at this stage is always triangles.

Example: Terrain Generation

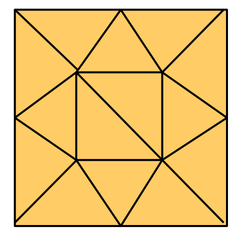
Dynamic Level of Detail (LOD)



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

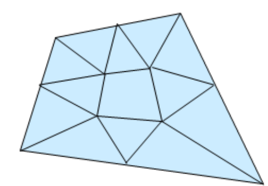
The tessellation levels are computed based on the distance of the patch from the camera

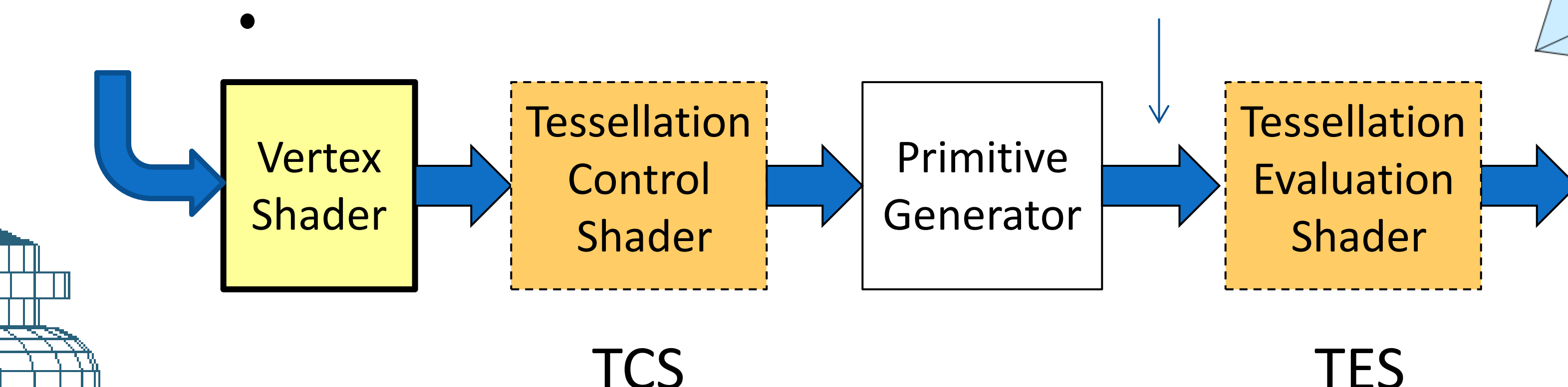
Tessellation Coordinates



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

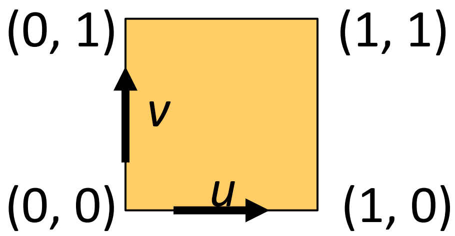
Vertices of Triangles





Normalized Parametric Domains

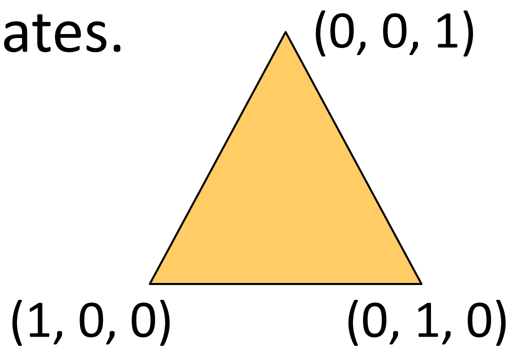
- The primitive generator always outputs vertices with coordinates in the range [0, 1] (Tessellation Coordinates). These coordinates are often used as parameters in blending functions to compute the point's final position.
- There are two types of normalized parametric domains:
  - **Quadrilateral Domain**: This is a unit square. Coordinate representation: (u, v)



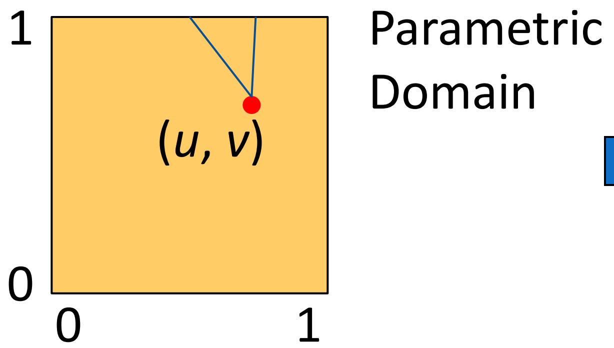
  - **Triangular domain**: This is an equilateral triangle, with vertices defined using **barycentric** coordinates.

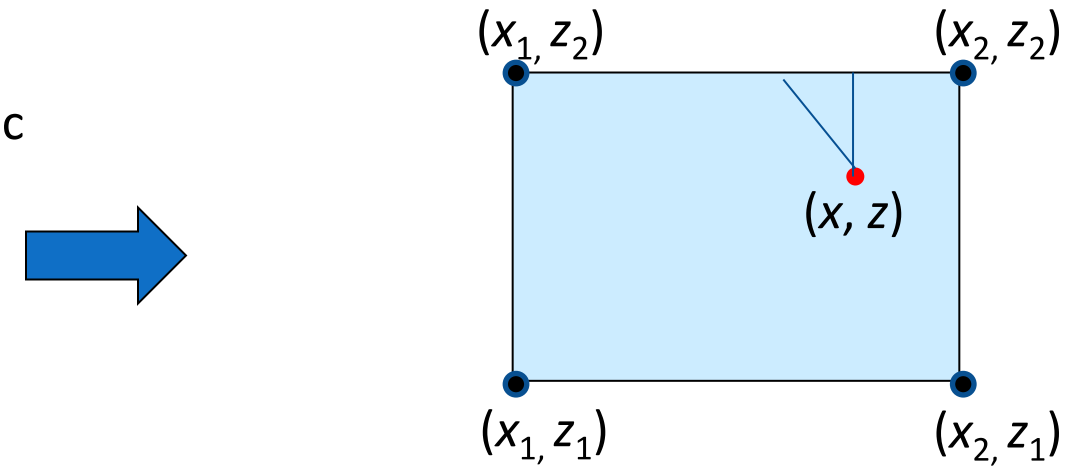
    Coordinate representation: (u, v, w)

    $u + v + w = 1.$



From Quad Domain to Surface





Assume that the patch vertices represent a rectangular segment (eg. a part of a terrain grid)

Given (u, v), what are the values of (x, y, z) ?

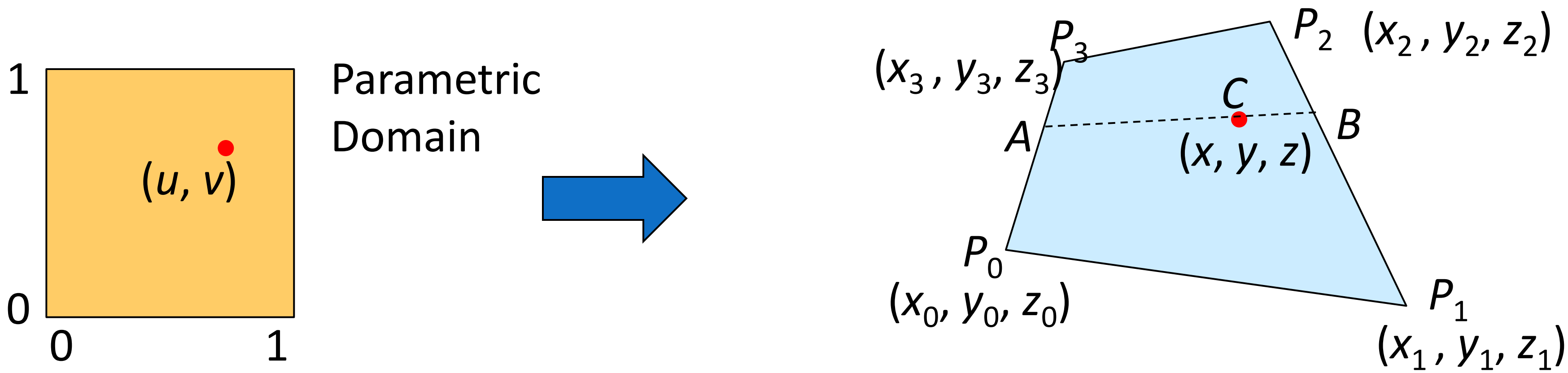
$$x = (1-u)\, x_1 + u\, x_2 \qquad 0 \le u \le 1.$$

$$y = 0$$

$$z = (1-v)\, z_1 + v\, z_2 \qquad 0 \le v \le 1.$$

Limitation: The patch must be rectangular in shape, with axis aligned edges.

Quad Domain: General Mapping



Given (u, v), what are the values of (x, y, z) ?

$$A = (1-v)\, P_0 + v\, P_3$$

$$B = (1-v)\, P_1 + v\, P_2 \qquad\qquad 0 \le v \le 1.$$

$$C = (1-u)\, A + u\, B \qquad\qquad 0 \le u \le 1.$$

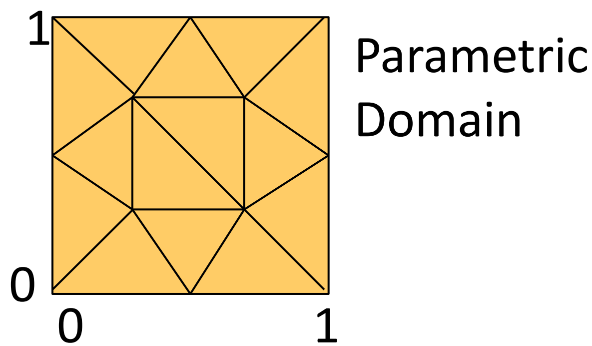
Hence,

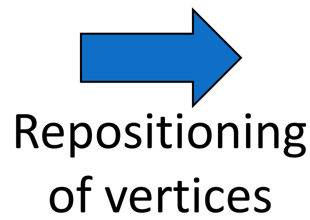
$$x = (1-u)\, \{\, (1-v)\, x_0 + v\, x_3 \,\} + u\, \{\, (1-v)\, x_1 + v\, x_2 \,\}$$

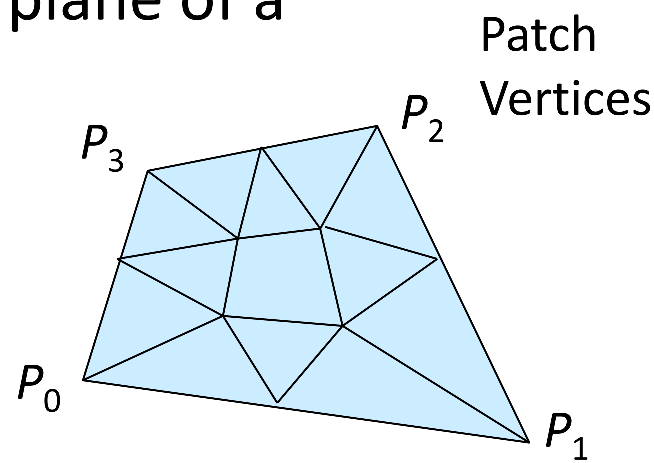
Similar equations for y, z.

Quad Domain: General Mapping

- The previous mappings tessellate the plane of a quadrilateral patch.

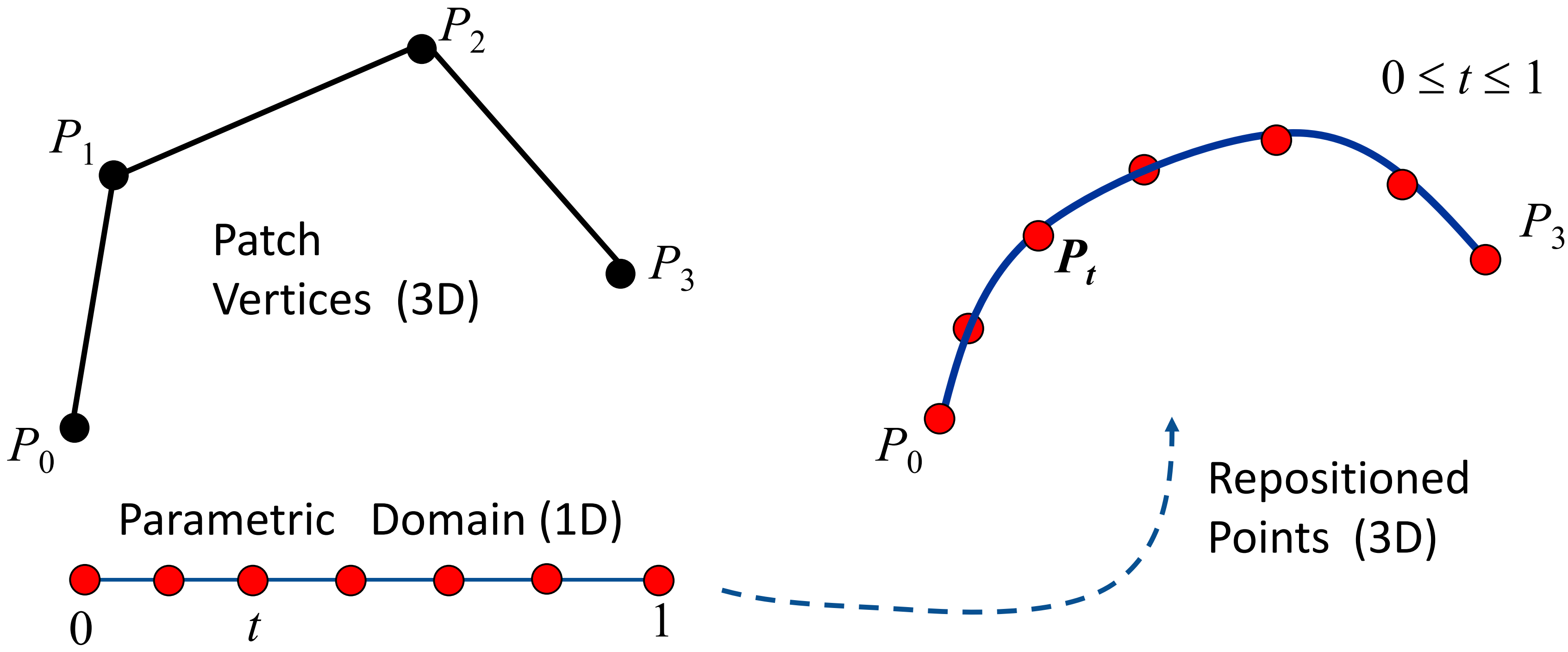






- However, tessellating a plane is not very useful!
  - Terrain applications could assign height values to the vertices using a height map texture (accessible from TES)
  - Surface design applications could use patch vertices as a control mesh to generate vertices of a Bezier patch (or similar approximating surfaces).

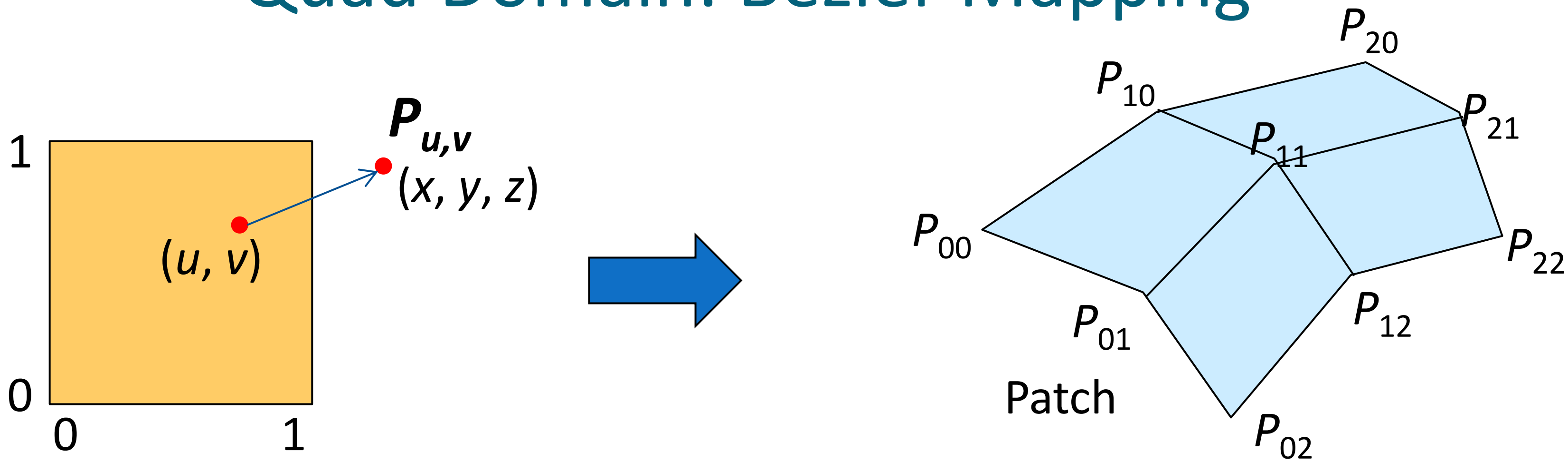
Bezier Curves Revisited



$$\boldsymbol{P_t} = (1-t)^3\, P_0 + 3(1-t)^2 t\, P_1 + 3(1-t)t^2 P_2 + t^3 P_3$$

- The range of the parameter *t* can be treated as a normalized parametric domain in 1D.
- We use a tessellation of this domain and the input patch vertices to generate a cubic Bezier curve.

Quad Domain: Bezier Mapping

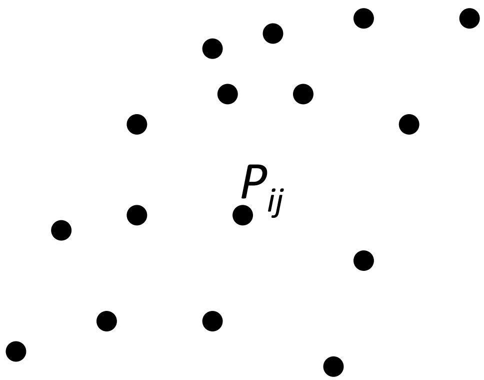


We can use a set of patch vertices $P_{00}...P_{22}$ to create a mapping $(u, v) \rightarrow (x, y, z)$ using Bezier surface equations!

$$
\begin{aligned}
\boldsymbol{P}_{u,v} = \quad & (1-u)^2 \ \{ (1-v)^2 P_{00} + 2v(1-v)\, P_{10} + v^2 P_{20} \} \\
& + 2(1-u)u \ \{ (1-v)^2 P_{01} + 2v(1-v)\, P_{11} + v^2 P_{21} \} \\
& + u^2 \ \{ (1-v)^2 P_{02} + 2v(1-v)\, P_{12} + v^2 P_{22} \}
\end{aligned}
$$

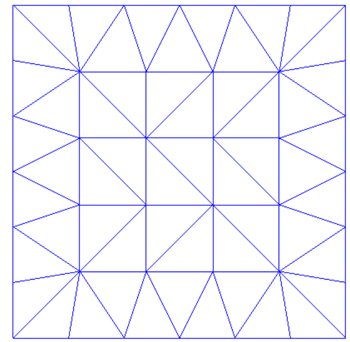
The above is a bi-quadratic Bezier surface.

Quad Domain: Bezier Mapping



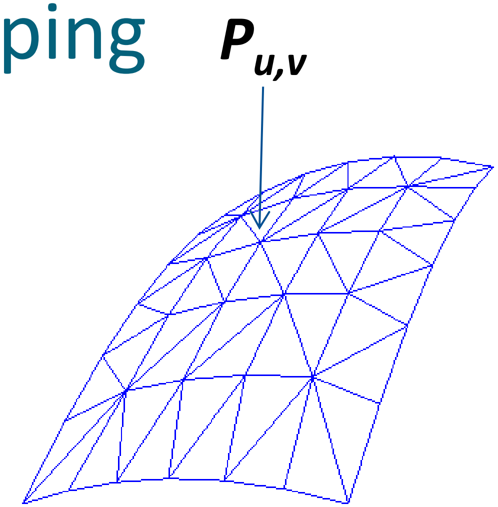
A 4x4 patch

+



Tess. Coords (u, v)

=



Repositioned Verts

- For generating a bi-cubic Bezier surface, we require a 4x4 patch as input.
- The Bezier surface is given by

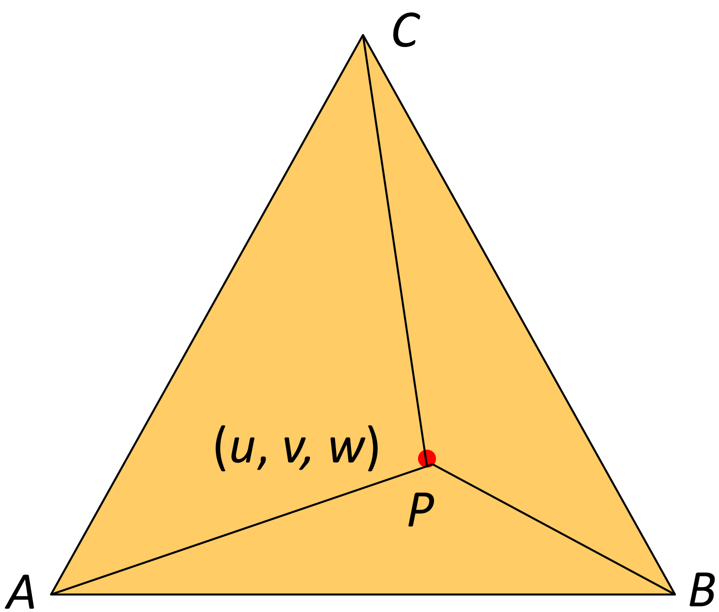
$$\boldsymbol{P_{u,v}} = \sum_{i=0}^{3}\sum_{j=0}^{3} B_i(u)B_j(v)P_{ij}$$

where

B_i () is the i^{th} **Bernstein polynomial** of degree 3

Barycentric Coordinates

The barycentric coordinates (u, v, w) of any point P that belongs to a triangle ABC are defined as follows:



$$u = \frac{\Delta PBC}{\Delta ABC}$$

$$v = \frac{\Delta APC}{\Delta ABC}$$

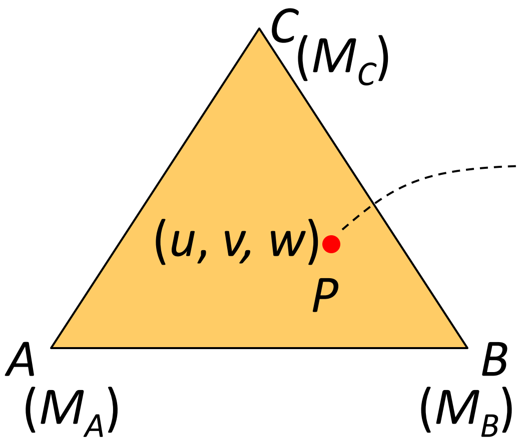
$$w = \frac{\Delta ABP}{\Delta ABC}$$

Properties: For any point, $u + v + w = 1$.

Vertices: $A = (1, 0, 0)$, $B = (0, 1, 0)$, $C = (0, 0, 1)$.

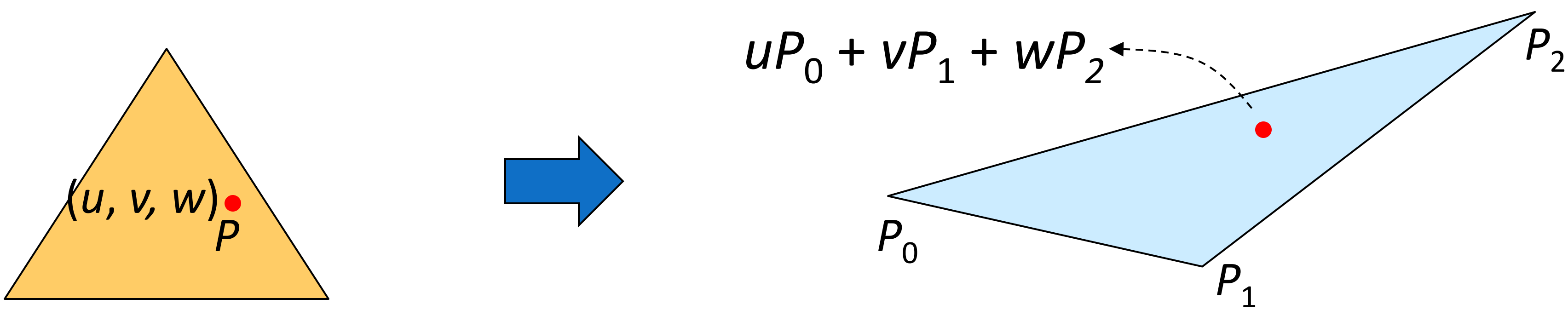
Barycentric Coordinates: Applications

Interpolation: Using barycentric coordinates, we can find the interpolated value of any vertex attribute (*M*)

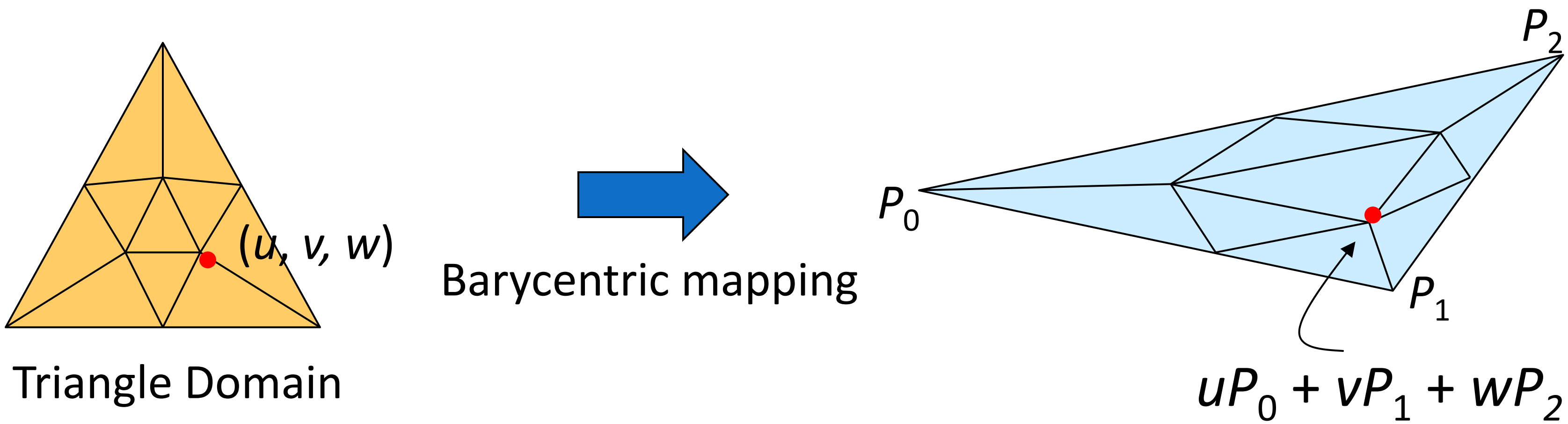


$$M_P = uM_A + vM_B + wM_C$$

Barycentric Mapping:



Triangle Domain: General Mapping



- We had many options in quadrilateral domain, but the only mapping possible here is through barycentric coordinates.
- For mesh subdivision, the coordinates of a mesh vertex is usually further modified using other information (eg. height map value, surface eqn. etc)

Tessellation Levels

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

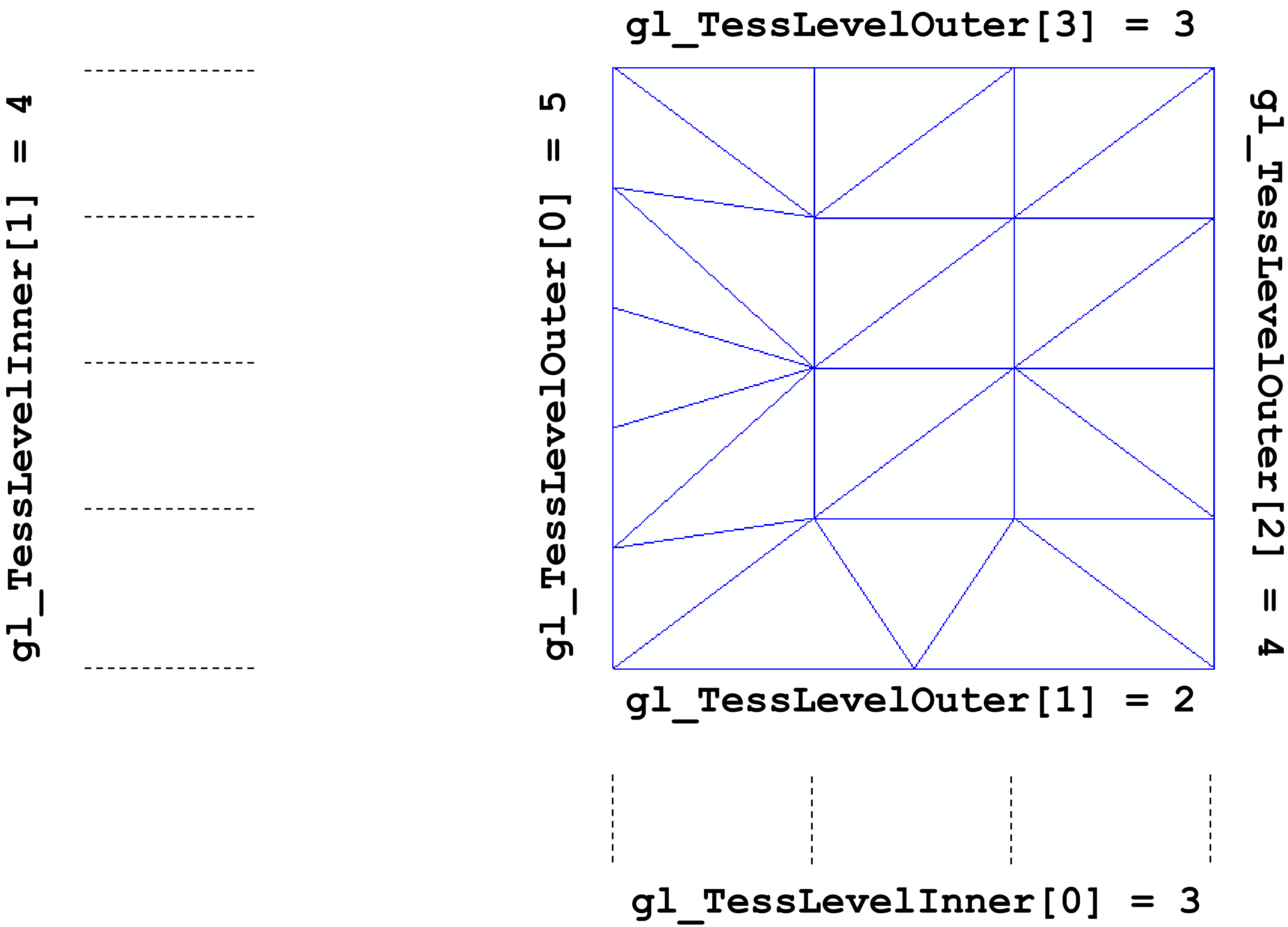
- Outer tessellation level:
  - 4 values (one for each side of the quad; for a triangle the last value is 0) stored in arrays

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

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

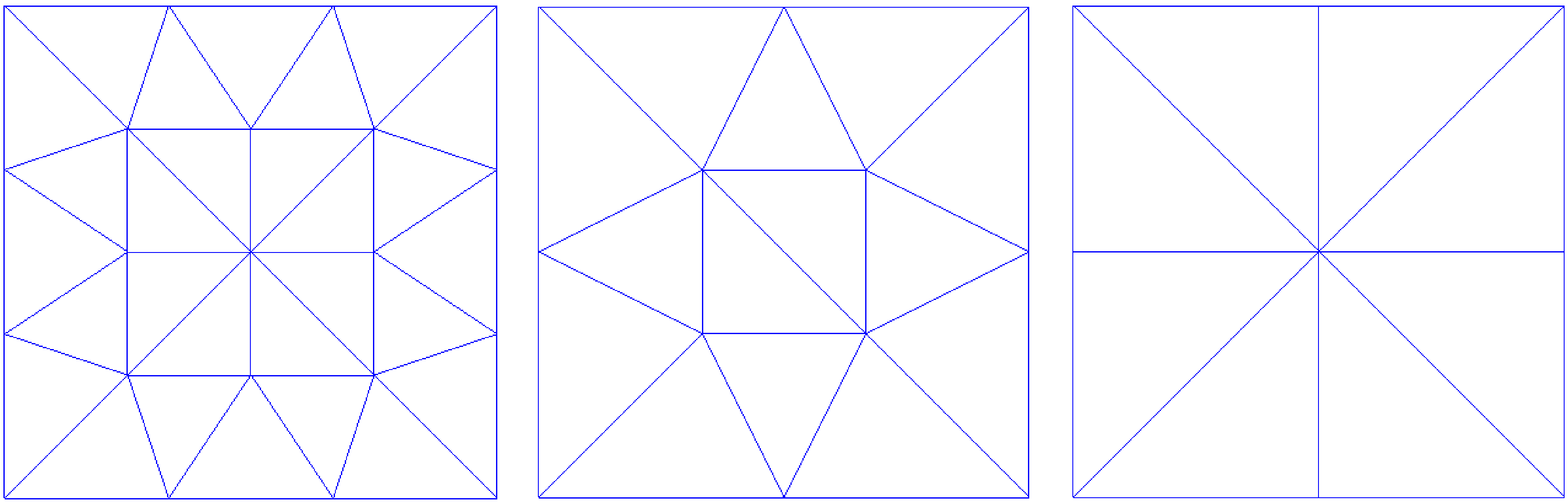
```
gl_TesslevelInner[0], gl_TesslevelInner[1]
```

Tessellation Levels: Quad



Tessellation Levels: Quad

QuadTessn.cpp

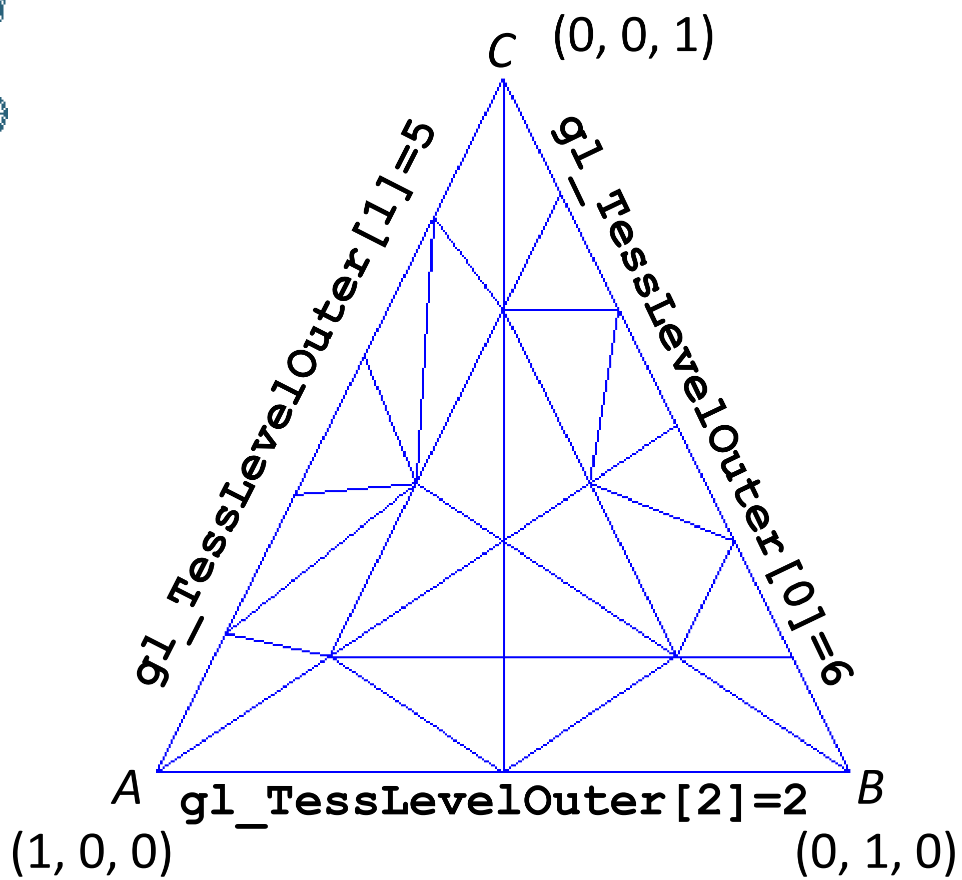


3333; 44   2222; 33   2222; 22

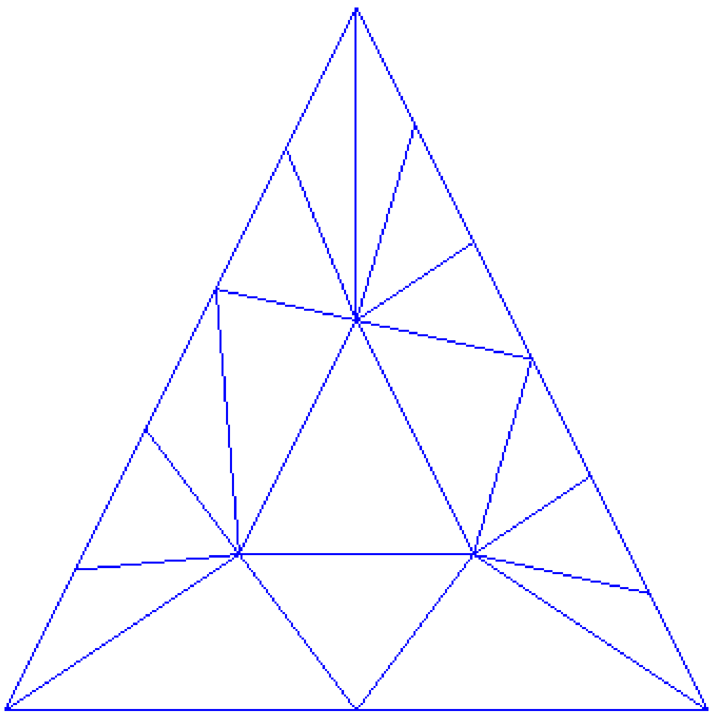
Outer   Inner

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

Tessellation Levels: Triangle



`gl_TessLevelInner[0] = 4`

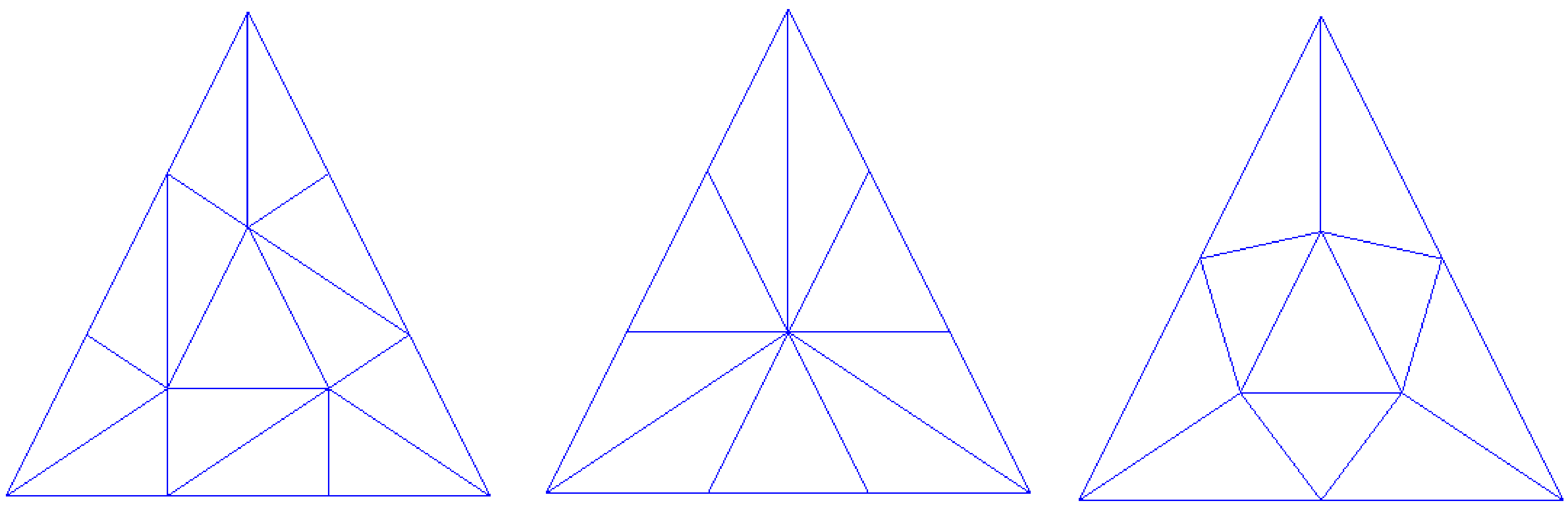


`gl_TessLevelInner[0] = 3`

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

Tessellation Levels: Triangle

`TriTessn.cpp`

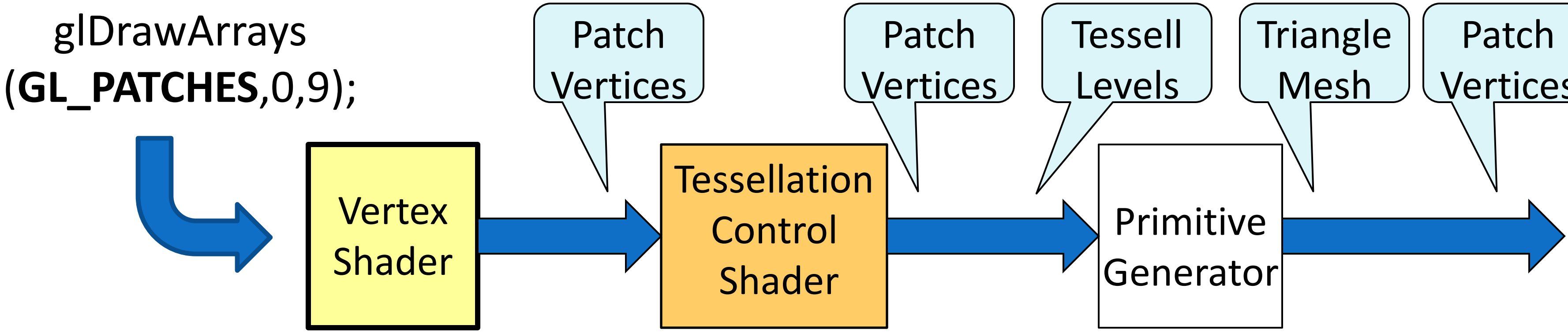


$\underbrace{3330;}_{\text{Outer}}\ \underbrace{30}_{\text{Inner}}$

3330; 20

2220; 30

Tessellation Control Shader



- 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

output patch vertices
`gl_PatchVerticesOut`

Index of the current out vertex

```
#version 400

layout(vertices = 3) out;

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

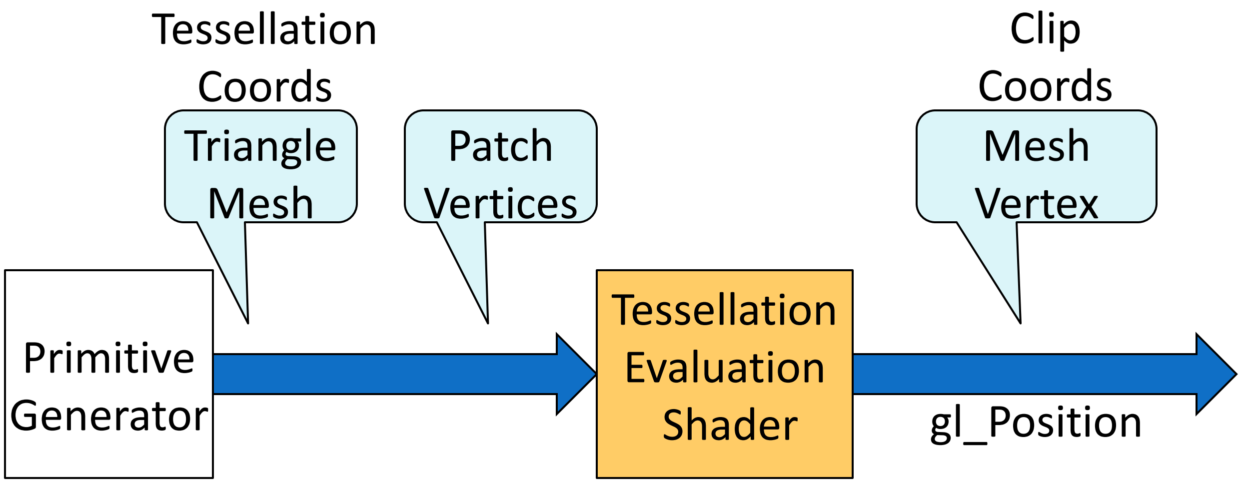
Tessellation Control Shader

- The tessellation control shader on the previous slide is a simple pass-through shader.
- Pass-through control shaders may be omitted (bypassed). The default tessellation levels for all patches can be specified in the OpenGL application as follows:

```
glPatchParameterfv(GL_PATCH_OUTER_LEVEL, olvl_arr);
glPatchParameterfv(GL_PATCH_INNER_LEVEL, ilvl_arr);
```

GLfloat arrays

Tessellation Evaluation Shader



- The primitive generator emits a triangle mesh with vertices defined in a normalized domain. These coordinates are referred to as tessellation coords.
- The tessellation evaluator repositions each mesh vertex using patch vertices, and outputs them in clip coordinates.
- The evaluation shader executes once for each mesh vertex.

Tessellation Evaluation Shader

```
#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 coords

Patch vertices

See slide 10

Clip Coords

Tessellation Evaluation Shader

```
#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)

See slide 17

Improper Tessellation: Cracking

An edge shared by two patches must be tessellated by the same amount for both patches.