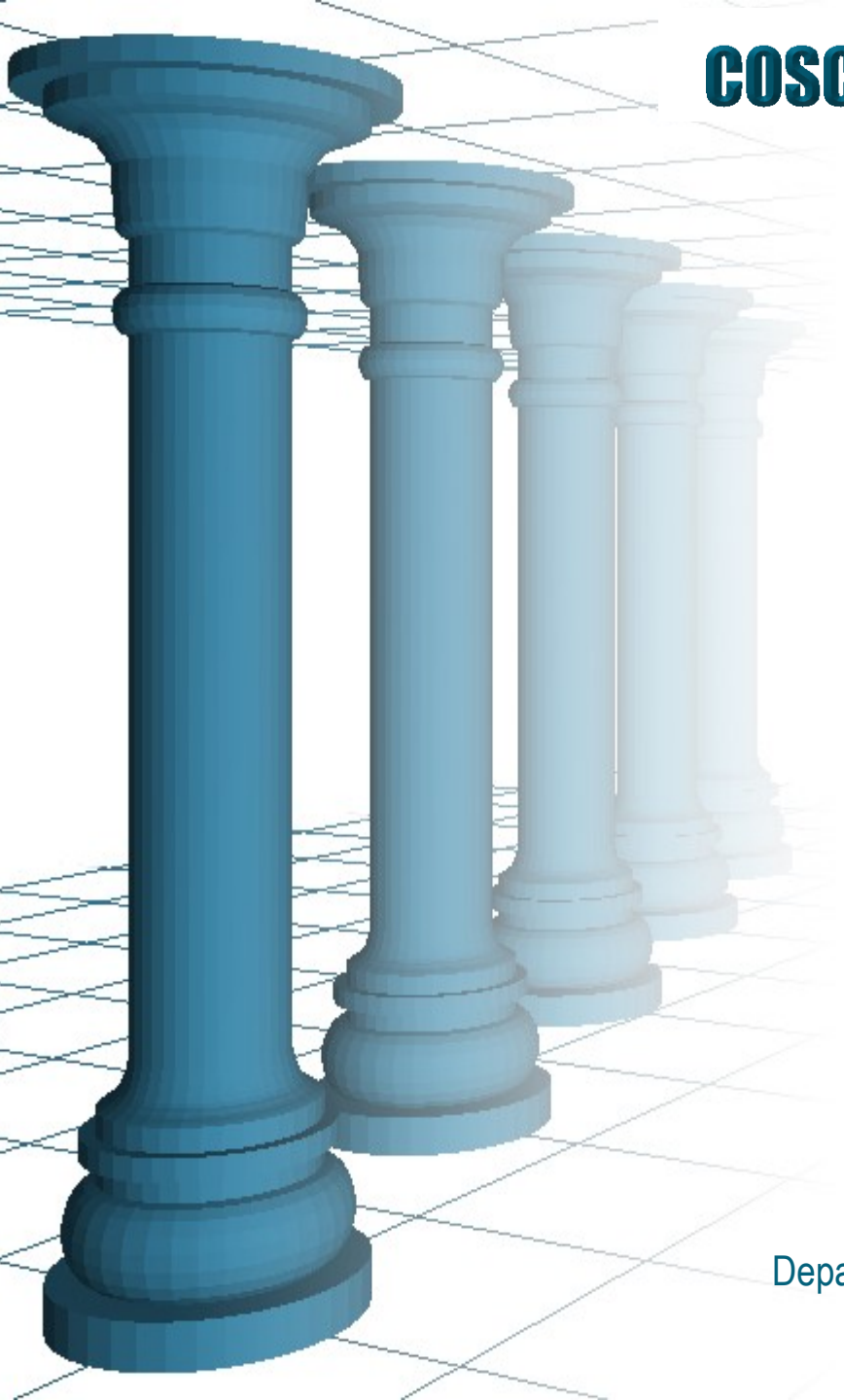


COSC363 Computer Graphics

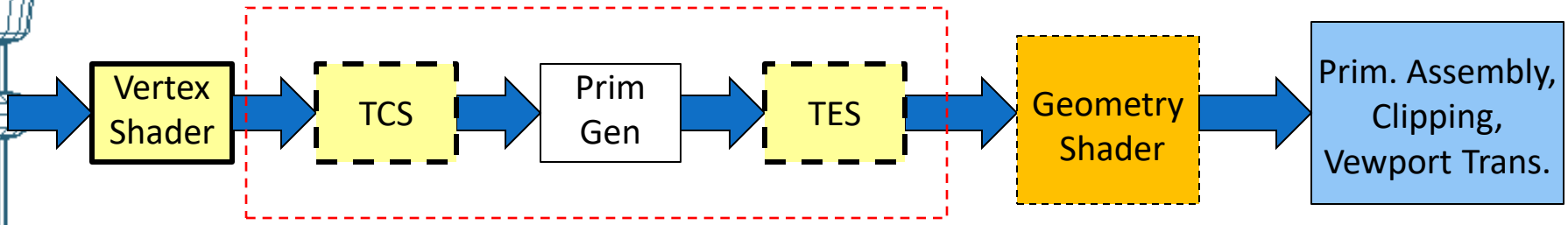
12

Geometry Shader

A Primitive Production Machine

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

OpenGL Geometry Shader



- The geometry shader receives inputs from either TES or the vertex shader (if tessellation is not active).
- The geometry shader receives inputs in arrays with values corresponding to one whole primitive.
- **A geometry shader can thus process an entire primitive**.

```
out vec3 normal;
out  vec4 colour;
 ...
 ...
 ...
gl_Position = ...;
```

Myshader.vert

```
in vec3 normal[];
in  vec4 colour[];
int nvert = gl_in[].length();
for(int i=0; i < nvert; i++) {
  vec4 p = gl_in[].gl_Position;
  ...
```

Myshader.geom

Geometry Shader

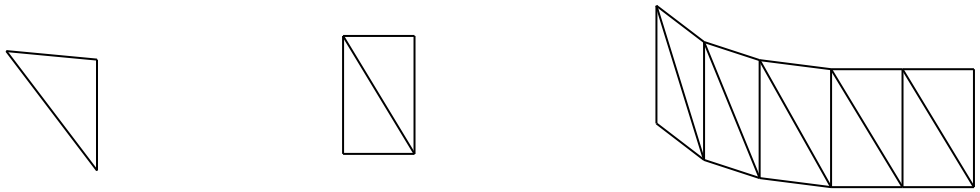
- A geometry shader runs once per input primitive. If the input is a triangle strip, then the geometry shader will be invoked for each individual triangle in the strip (as if the triangle strip were expanded out into a triangle list).
- A geometry shader can change the type of the input primitive and the number of vertices. It can also be used to discard primitives.
- The power of the geometry shader lies in its ability to get information about each primitive, and also to output values into multiple streams.
- The geometric shader is the last shader stage before clipping and rasterization. It must emit each vertex in the clip coordinate space using the built-in variable gl_Position.

Geometry Shader

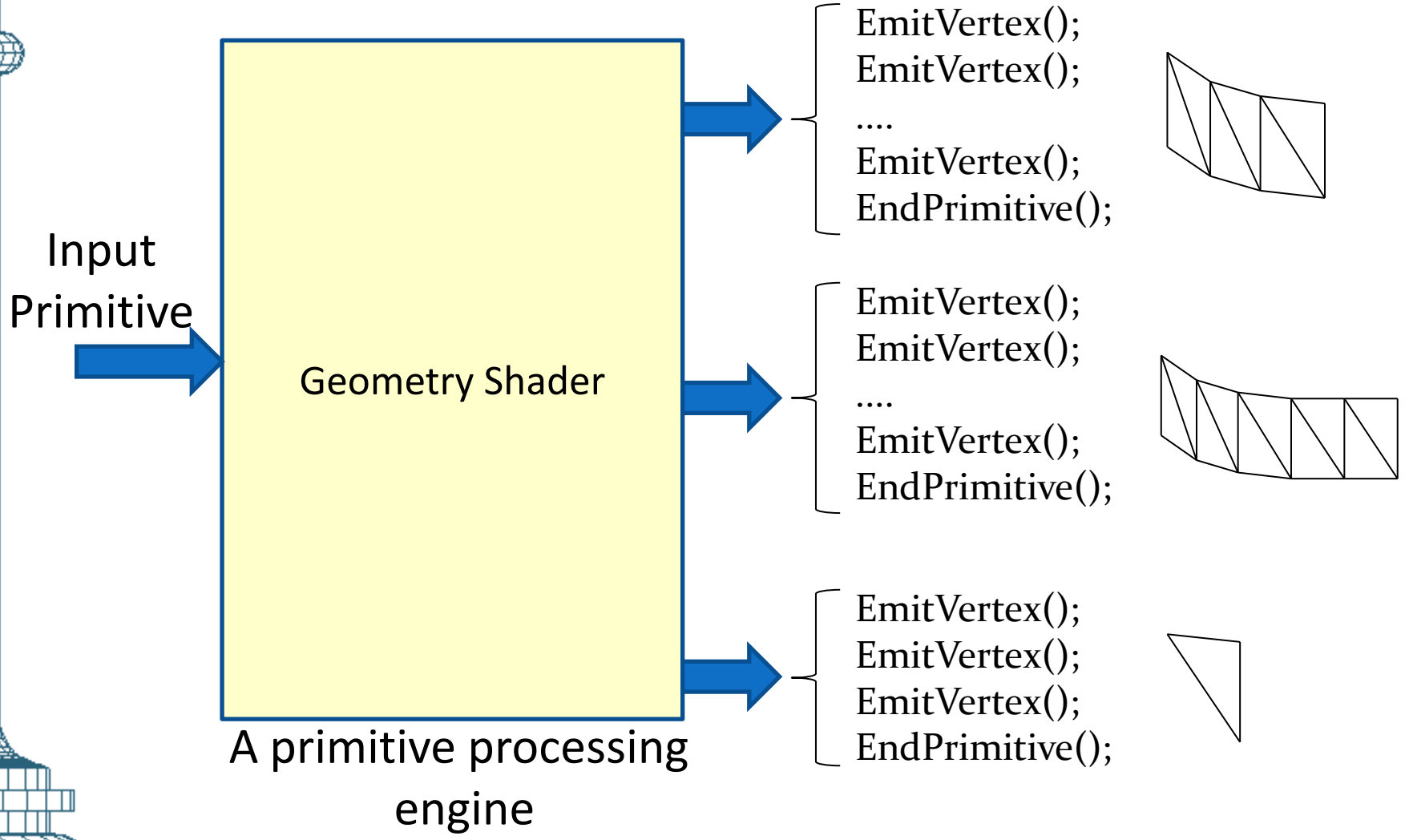
- The input primitive type is specified by the application through glDrawArrays() or glDrawElements(). E.g:

```
glDrawArrays(GL_TRIANGLES, 0, 300);
```

- The input primitve can be of any type: points, lines, line strip, triangles, quads etc.
- The geometry shader can also accept a special primitive type **GL_TRIANGLES_ADJACENCY**
- The only output primitive types that can be used in a geometry shader are points, line strip and triangle strip
- Triangles and quads can be generated as special cases of a triangle strip



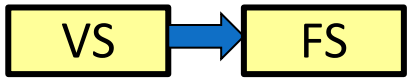
Geometry Shader



Geometry Shader Outputs

- A geometry shader has two built-in functions **`EmitVertex()`** and **`EndPrimitive()`**.
- In one execution of the geometry shader, it can produce multiple vertices and multiple primitives (multiple triangles as part of a triangle strip).
- Each call to `EmitVertex()` appends one vertex at the end of the current primitive. Its position is defined by **`gl_Position`**, and other attributes using out variables.
- If the geometry shader exits without calling `EmitVertex()`, then it does not produce any output primitive.

A Simple Application **Without** GS



Application

Defines a VBO for 4 Quads (16 Vertices)

Vertex Shader

Converts Vertices to Clip Coordinates

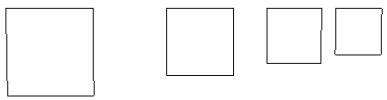
Fragment Shader

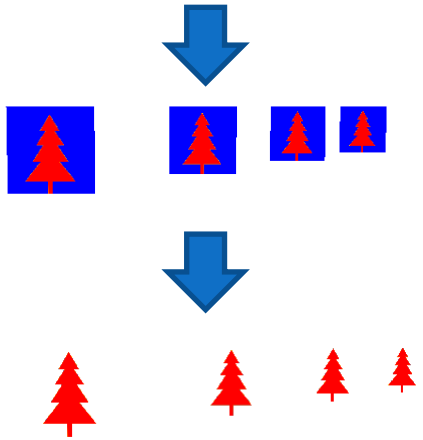
Accesses a Texture and Discards Fragments

```
glDrawArrays
  (GL_QUADS, 0, 16);
```

```
void main()
{
 gl_Position =
  mvpMatrix * position;
 oTexCoord = texcoord;
}
```

```
void main()
{
  col=texture(tree, oTexcoord);
  if (col.b > 0.5) discard;
  gl_FragColor = col;
}
```





A Simple Application **With** GS

Application

Defines a VBO for 4 Points
(4 Vertices)

```
glDrawArrays
  (GL_POINTS, 0, 4);
```

Vertex Shader

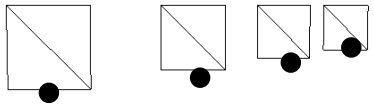
A Pass-through Shader

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

Geometry Shader

Generates quads and texture coordinates

See next slide



A Simple Application **With** GS

Geometry Shader

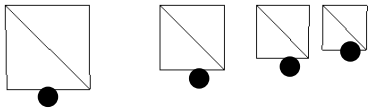
Generates quads and texture coordinates

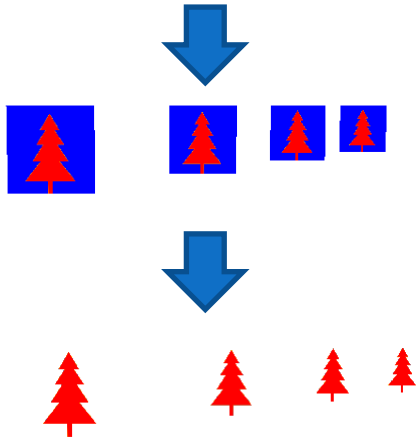
```
#version 400
layout (points) in;
layout (triangle_strip, max_vertices = 4) out;
uniform mat4 mvpMatrix;
out vec2 oTexcoord;
void main()
{
   vec4 pos;
   vec4 p = gl_in[0].gl_Position;

   pos = vec4(p.x-0.5, p.y, p.z, 1);
   gl_Position = mvpMatrix * pos;
   oTexcoord = vec2(0, 0);
   EmitVertex();

   pos = vec4(p.x+0.5, p.y, p.z, 1);
   gl_Position = mvpMatrix * pos;
   oTexcoord = vec2(1, 0);
   EmitVertex();

   pos = vec4(p.x-0.5, p.y+1, p.z, 1);
   gl_Position = mvpMatrix * pos;
   oTexcoord = vec2(0, 1);
   EmitVertex();

   pos = vec4(p.x+0.5, p.y+1, p.z, 1);
   gl_Position = mvpMatrix * pos;
   oTexcoord = vec2(1, 1);
   EmitVertex();

   EndPrimitive();
}
```

Fragment Shader

Accesses a Texture and Discards Fragments

No change in Fragment shader (Slide 7)

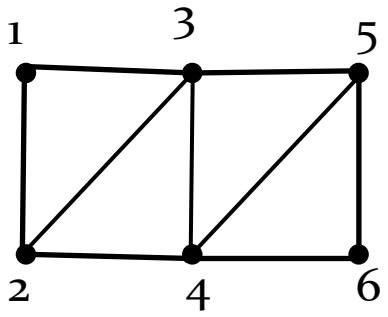




Geometry Shader Output Example 1

Producing a triangle strip:

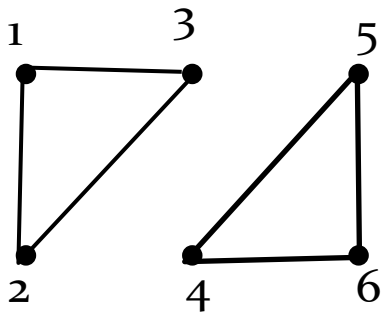
```
gl_Position = ...;  //Point 1
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 2
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 3
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 4
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 5
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 6
oColor = ...;
EmitVertex();
EndPrimitive();
```



Geometry Shader Output Example 2

Producing triangles:

```
gl_Position = ...;  //Point 1
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 2
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 3
oColor = ...;
EmitVertex();
EndPrimitive();
gl_Position = ...;  //Point 4
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 5
oColor = ...;
EmitVertex();
gl_Position = ...;  //Point 6
oColor = ...;
EmitVertex();
EndPrimitive();
```

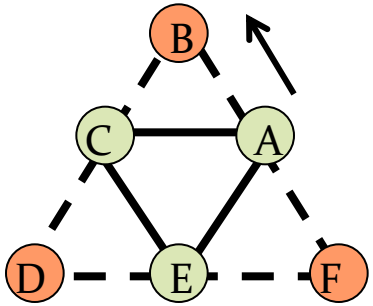


Adjacency Primitives

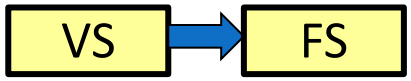
OpenGL provides new types of adjacency primitives that contains information about vertices adjacent to the current primitive

- GL_LINES_ADJACENCY: 4 vertices per primitive. The second and third vertices form the current line. The first and the fourth vertices are treated as adjacent to it.

- GL_TRIANGLES_ADJACENCY: 6 vertices per primitive. First, third and fifth vertices form the current triangle.



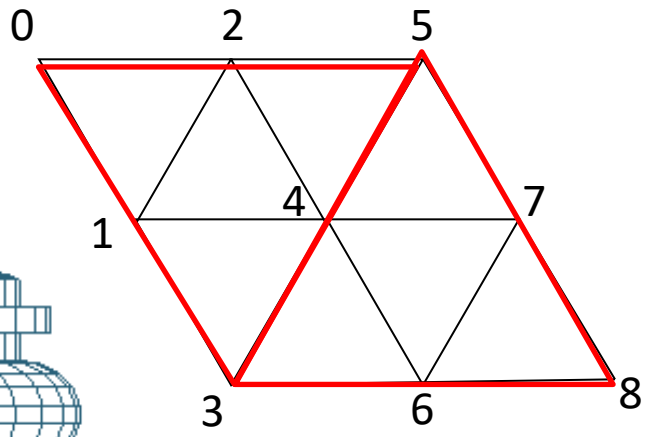
Adjacency Primitives

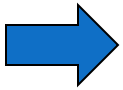


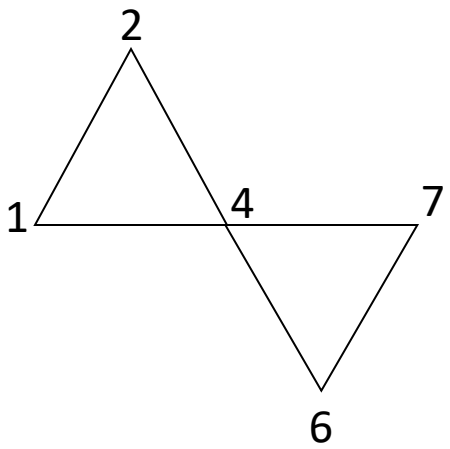
When an adjacency primitive is drawn without a geometry shader, only vertices corresponding to the current primitive (line or triangle) are drawn. The adjacent vertices are discarded.

Example: 1, 3, 4, 5, 2, 0, 6, 8, 7, 5, 4, 3

```
glDrawElements
(GL_TRIANGLES_ADJACENCY,12,GL_UNSIGNED_INT,NULL);
```

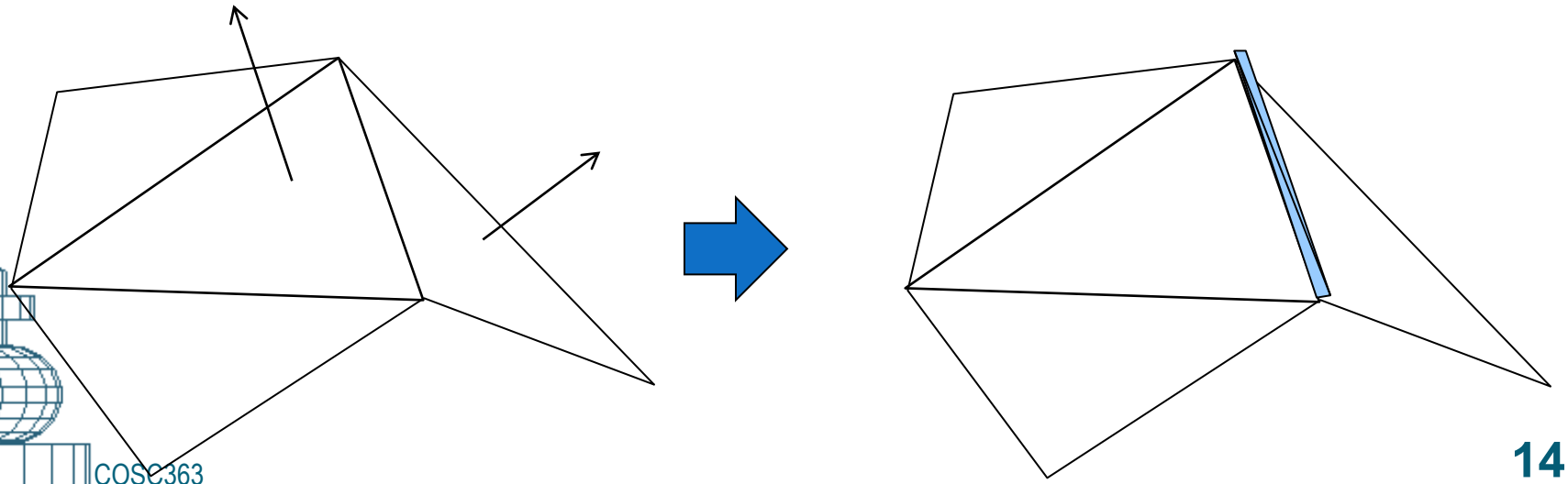






Adjacency Primitives

- If a geometry shader is active, it will receive all vertices of the adjacency primitive in `gl_in[].gl_Position` array.
- We can use information about adjacent triangles to modify the attributes of the current triangle or to generate additional primitives.
- Example: Crease edges can be detected by measuring the angle between surface normal vectors of adjacent triangles.



Geometry Culling

A geometry shader can discard a primitive as a whole:

```
#version 400
layout (triangles) in;
layout (triangle_strip, max_vertices = 3) out;
void main()
{
    int i;
    if(condition) return;
    for(i=0; i<gl_in.length(); i++)
    {
        gl_Position = gl_in[i].gl_Position;
        EmitVertex();
    }
    EndPrimitive();
}
```

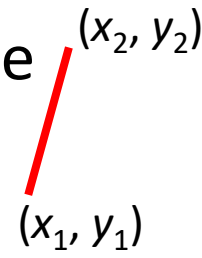
Some user-defined condition
Eg. n.v < 0

Geometry Amplification

Line strip

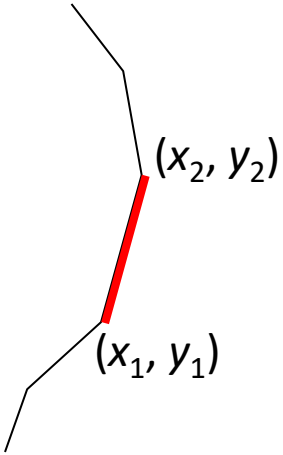
Example:

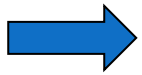
- The application generates a 2D line strip.
- The geometry shader receives individual lines of the line strip: $(x_1, y_1), (x_2, y_2)$.
- The geometry shader outputs a triangle strip for each line received.

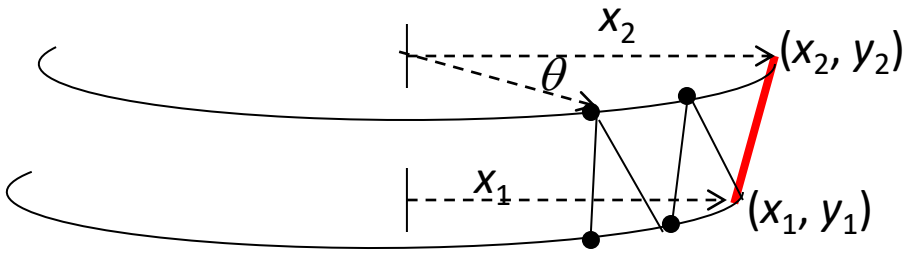



```
glDrawArrays(GL_LINE_STRIP,...);
```

```
layout(triangle_strip)out;
```

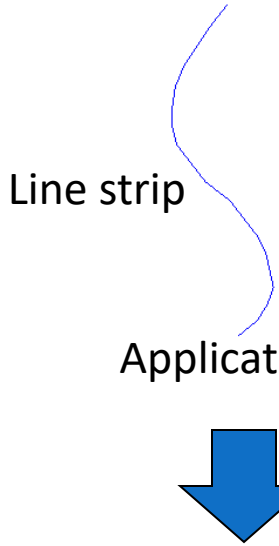


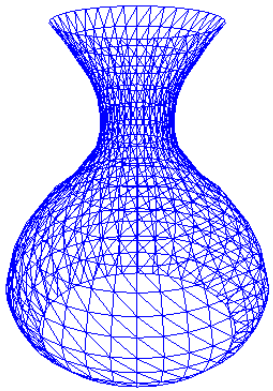




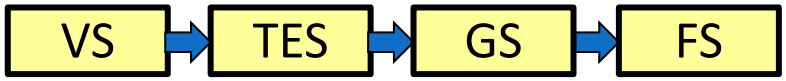
Geometry Amplification

```
layout (lines) in;
layout (triangle_strip, max_vertices = 80) out;
void main()
{
    float x1 = gl_in[0].gl_Position.x;
    float y1 = gl_in[0].gl_Position.y;
    float x2 = gl_in[1].gl_Position.x;
    float y2 = gl_in[1].gl_Position.y;
    float xnew, znew;
    float theta, ctheta, stheta;
   int i;
   for(i = 0;  i <= 36; i++)
   {
       theta = i * 0.17453;   //step size = 10 degs
       ctheta = cos(theta);
       stheta = sin(theta);
       xnew =  x1 * ctheta;
       znew = -x1 * stheta;
       gl_Position = mvpMatrix * vec4(xnew, y1, znew, 1);
       EmitVertex();
       xnew =  x2 * ctheta;
       znew = -x2 * stheta;
       gl_Position = mvpMatrix * vec4(xnew, y2, znew, 1);
       EmitVertex();
    }
    EndPrimitive();
}
```



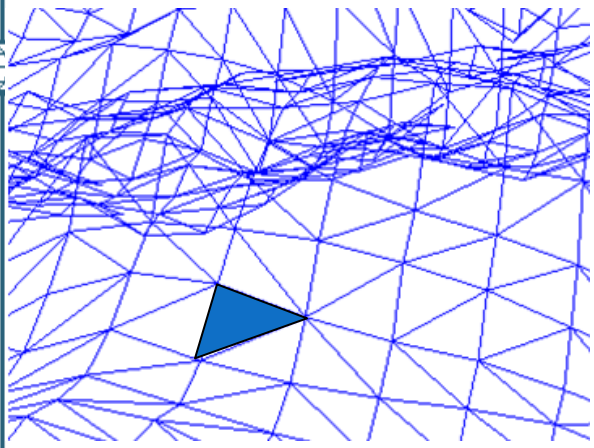


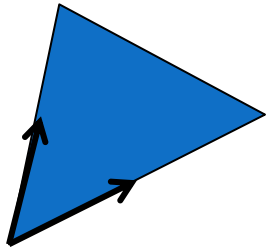
Tessellation + GS

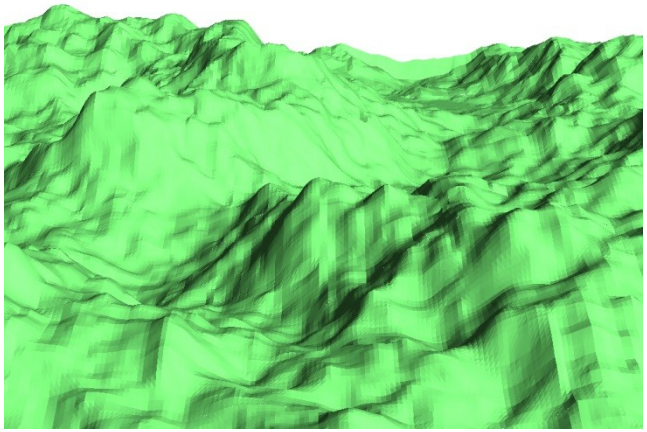


- The vertex shader can access only one vertex of a primitive
- If the tessellation stage is active, the tessellation evaluation shader also has access to only one vertex (u, v) generated by the primitive generator
- The tessellation evaluation shader repositions each vertex on a surface. To perform lighting calculations on the repositioned mesh triangle, we require surface normal vectors at the vertices.
- Since the geometry shader receives all vertices of each triangle of the mesh, we can perform the lighting calculation inside the geometry shader. Useful for rendering terrains and Bezier surfaces.

Lighting Calculations in a GS







Geometry Shader: (input: triangle)

- Receives all three vertices of each triangle.
- Computes the unit normal vector $\boldsymbol{n}$ of the triangle.
- Performs lighting calculation
- Outputs the colour values and the vertices in clip coordinates.

Lighting Calculations in a GS

```
layout (triangles) in;
layout (triangle_strip, max_vertices = 3) out;
...
out vec4 oColor;

  vec3 vector1 = gl_in[0].gl_Position.xyz
               - gl_in[2].gl_Position.xyz;
  vec3 vector2 = gl_in[1].gl_Position.xyz
               - gl_in[2].gl_Position.xyz;
  vec3 normal = normalize(cross(vector1, vector2)
  //Perform lighting calculation
  color = ...
  for(i = 0; i < gl_in.length(); i++)
  {
       gl_Position = mvpMatrix * gl_in[i].gl_Position;
       oColour = color;
       EmitVertex();
  }
  EndPrimitive();
```