Juraj Onderik | onderik@sccg.sk

Lesson

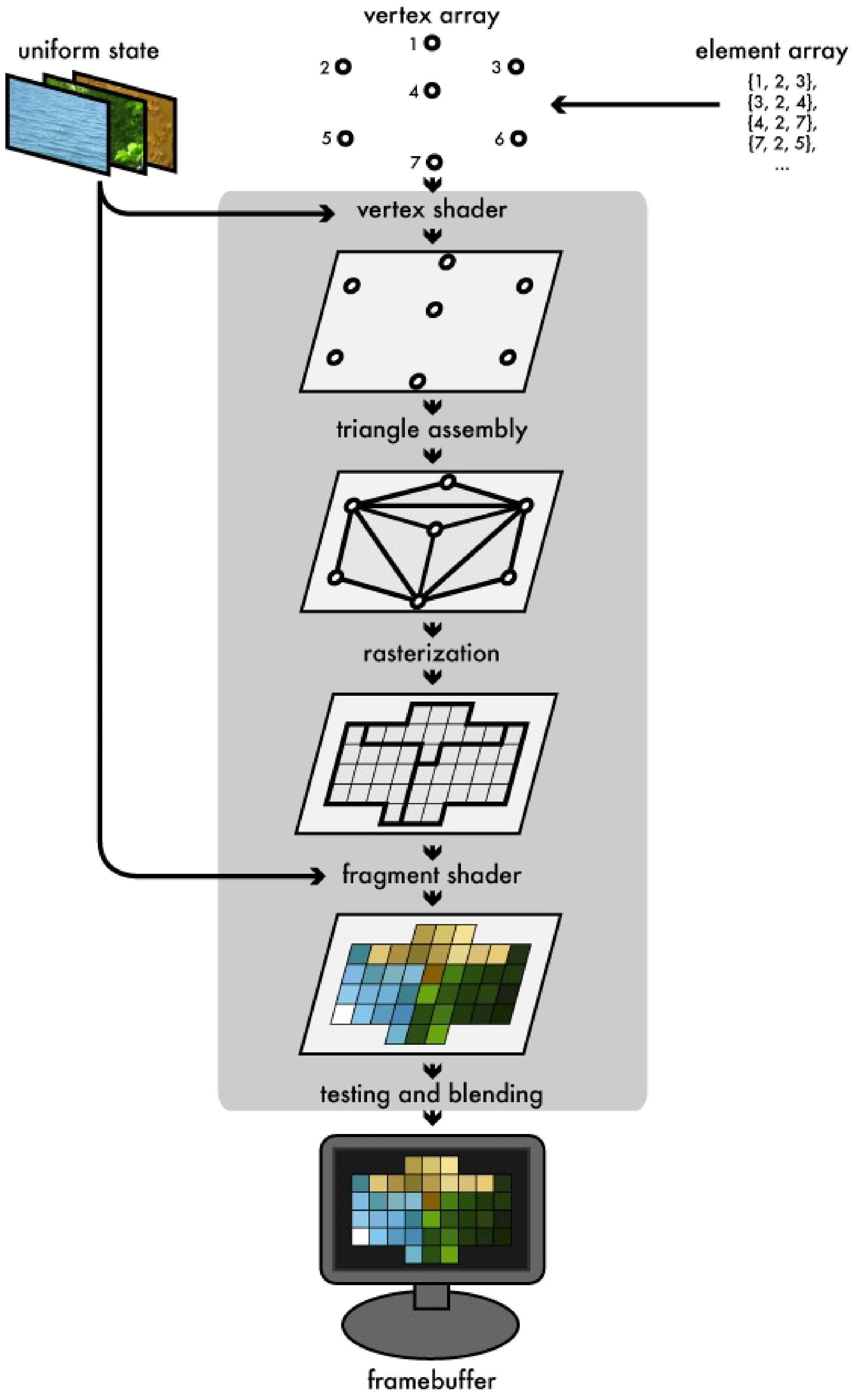
05

The Graphics Pipeline

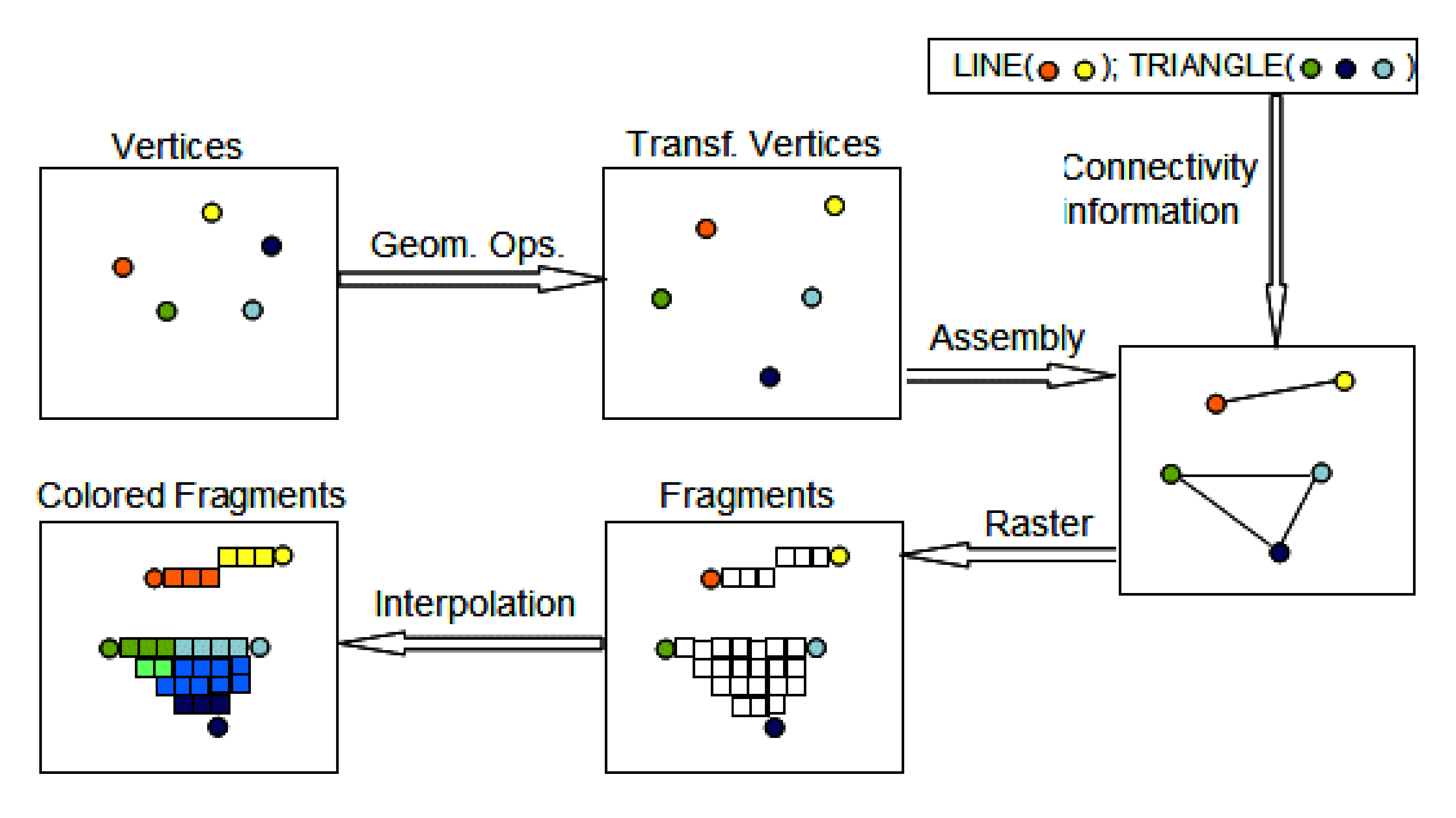
Outline of Lesson 05

- What is The Graphics Pipeline
- Vertex Shader
- Primitive Assembly
- Tessellation Shaders
- Geometry Shader
- Geometry Postprocessing and Rasterization
- Fragment Shader
- Frame Buffer Operations

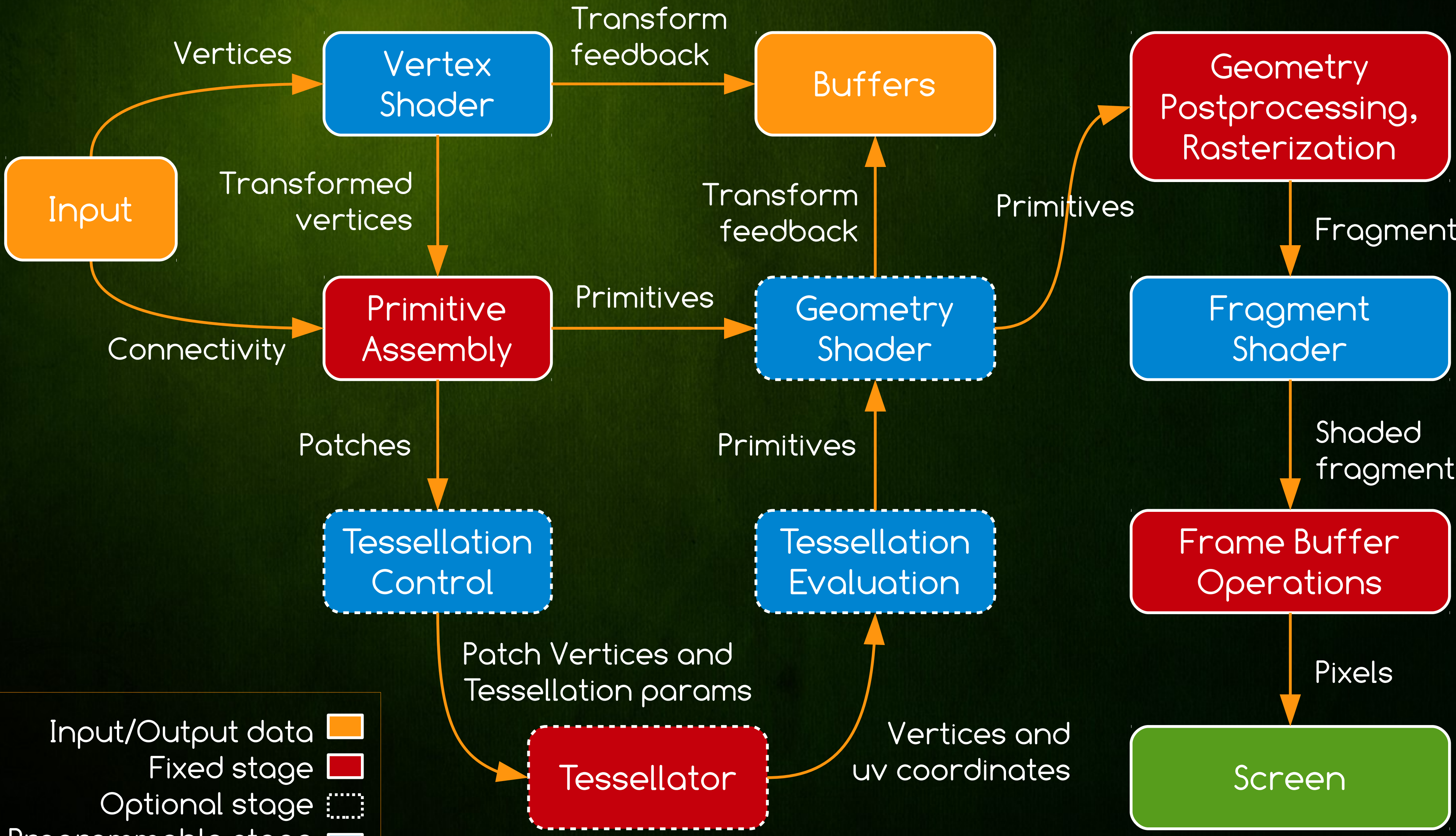
The Graphics Pipeline



Fixed Pipeline Overview

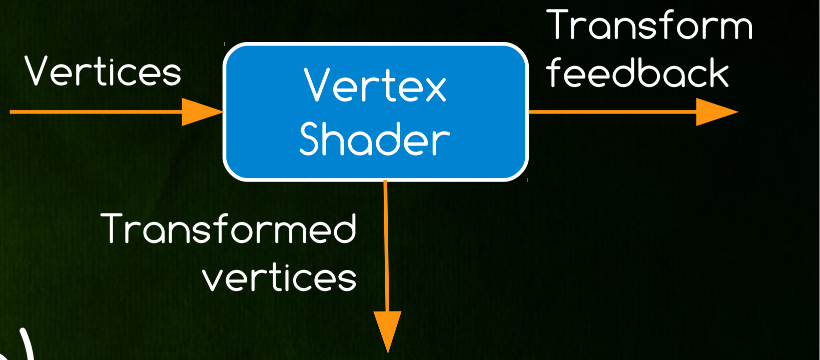


The OpenGL Graphics Pipeline



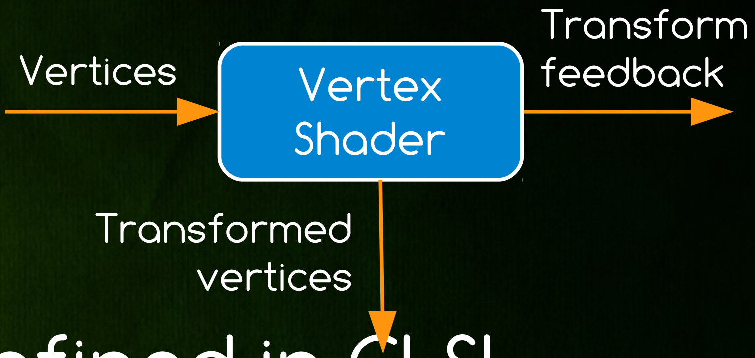
Input/Output data	🟧
Fixed stage	🟥
Optional stage	⬚
Programmable stage	🟦

:: Vertex Shader

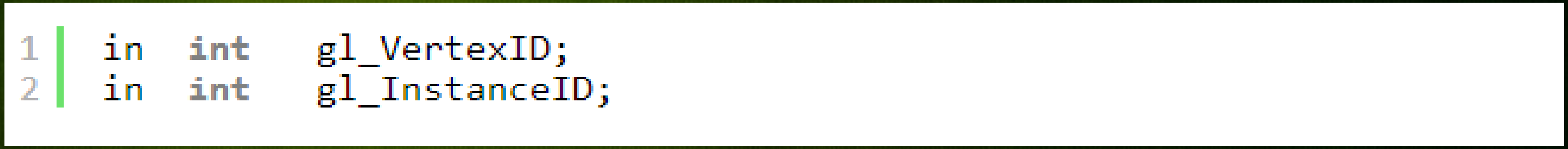


- Specification (Programmable)
  - Operates on vertices, one vertex at a time.
  - Has no knowledge of primitive or its type of the vertex
  - Input: Single vertex
  - Output: Single transformed vertex
- Main Purpose
  - Model-View-Projection transformations
  - Per-vertex Lighting

:: Vertex Shader

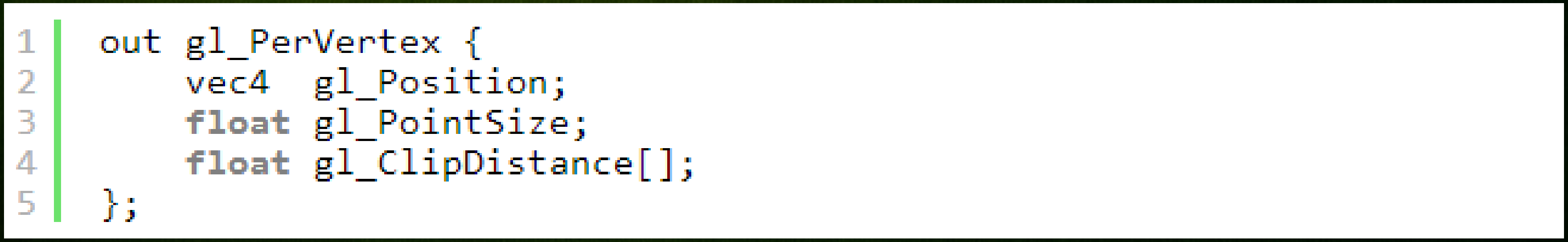


- Input per-vertex variables defined in GLSL



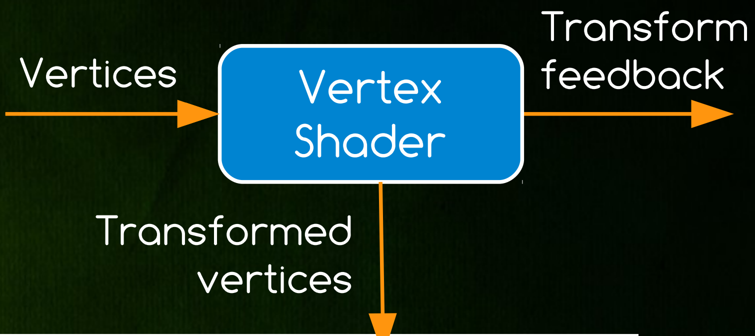
```
in  int   gl_VertexID;
in  int   gl_InstanceID;
```

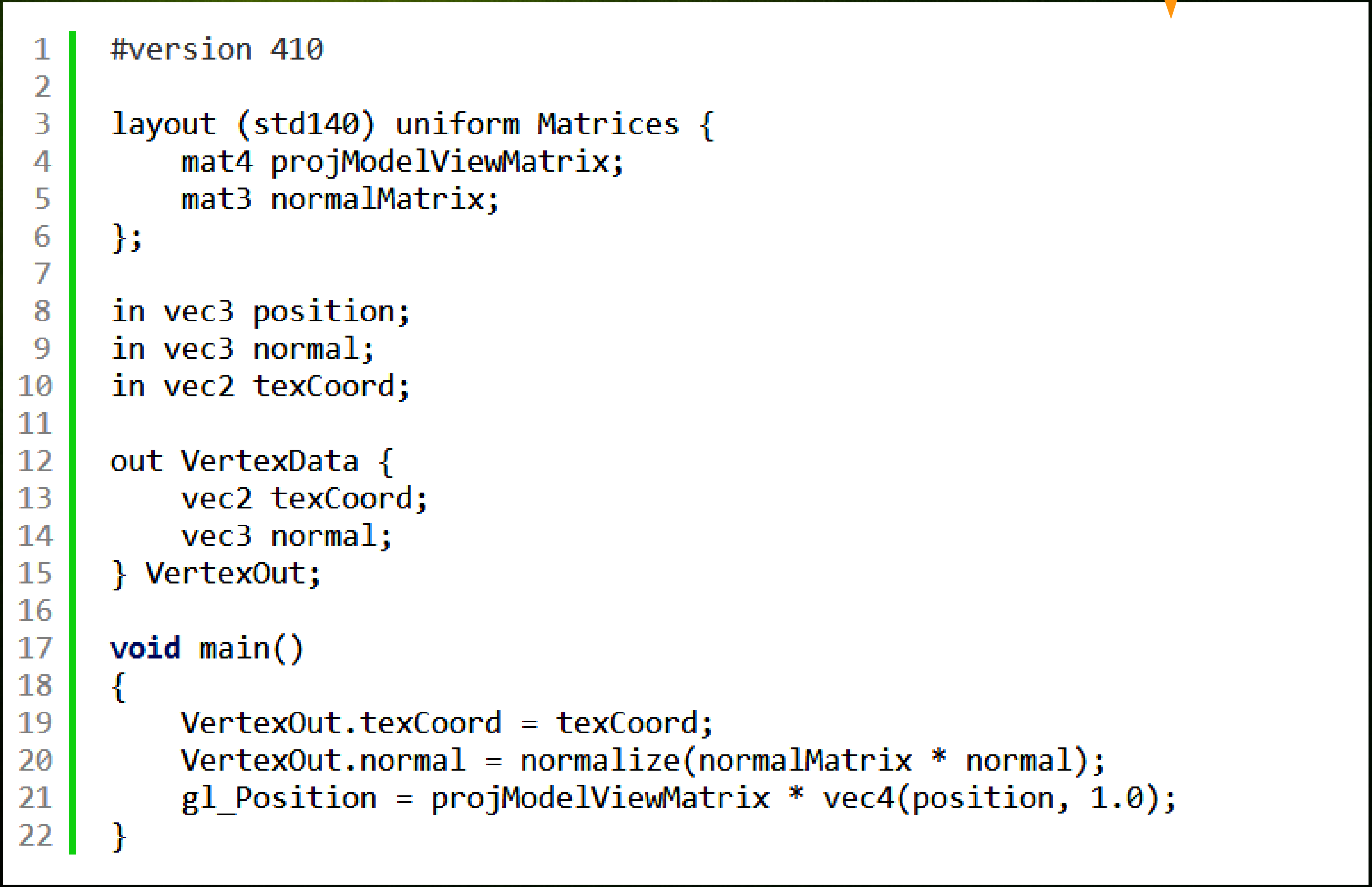
- Output per-vertex variables defined in GLSL



```
out gl_PerVertex {
    vec4  gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
};
```

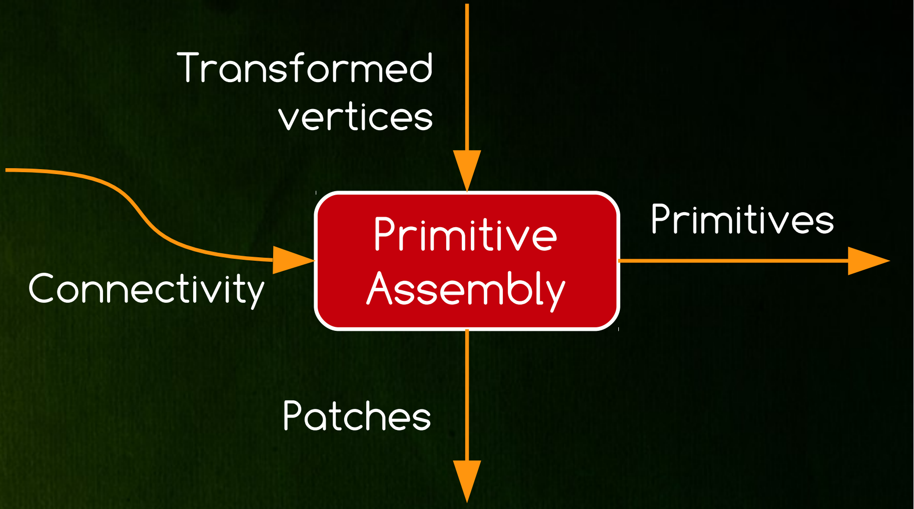
:: Vertex Shader





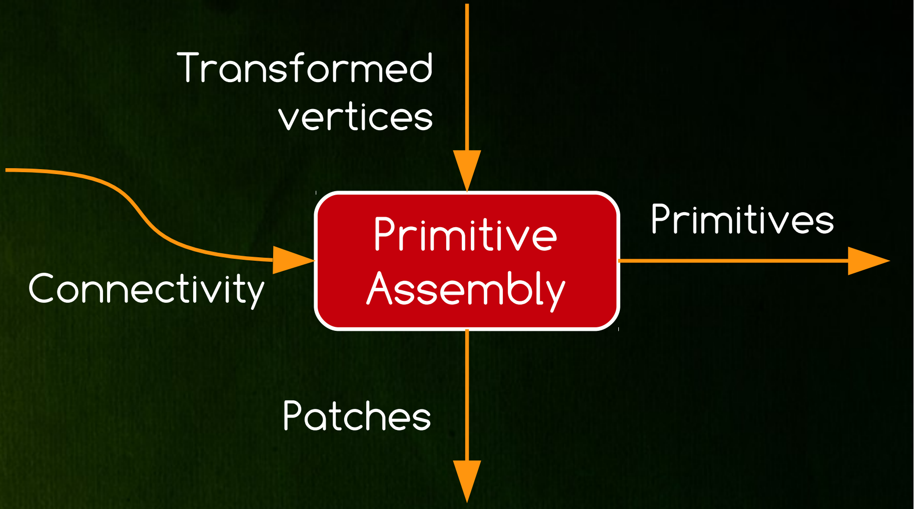
```
#version 410

layout (std140) uniform Matrices {
    mat4 projModelViewMatrix;
    mat3 normalMatrix;
};

in vec3 position;
in vec3 normal;
in vec2 texCoord;

out VertexData {
    vec2 texCoord;
    vec3 normal;
} VertexOut;

void main()
{
    VertexOut.texCoord = texCoord;
    VertexOut.normal = normalize(normalMatrix * normal);
    gl_Position = projModelViewMatrix * vec4(position, 1.0);
}
```

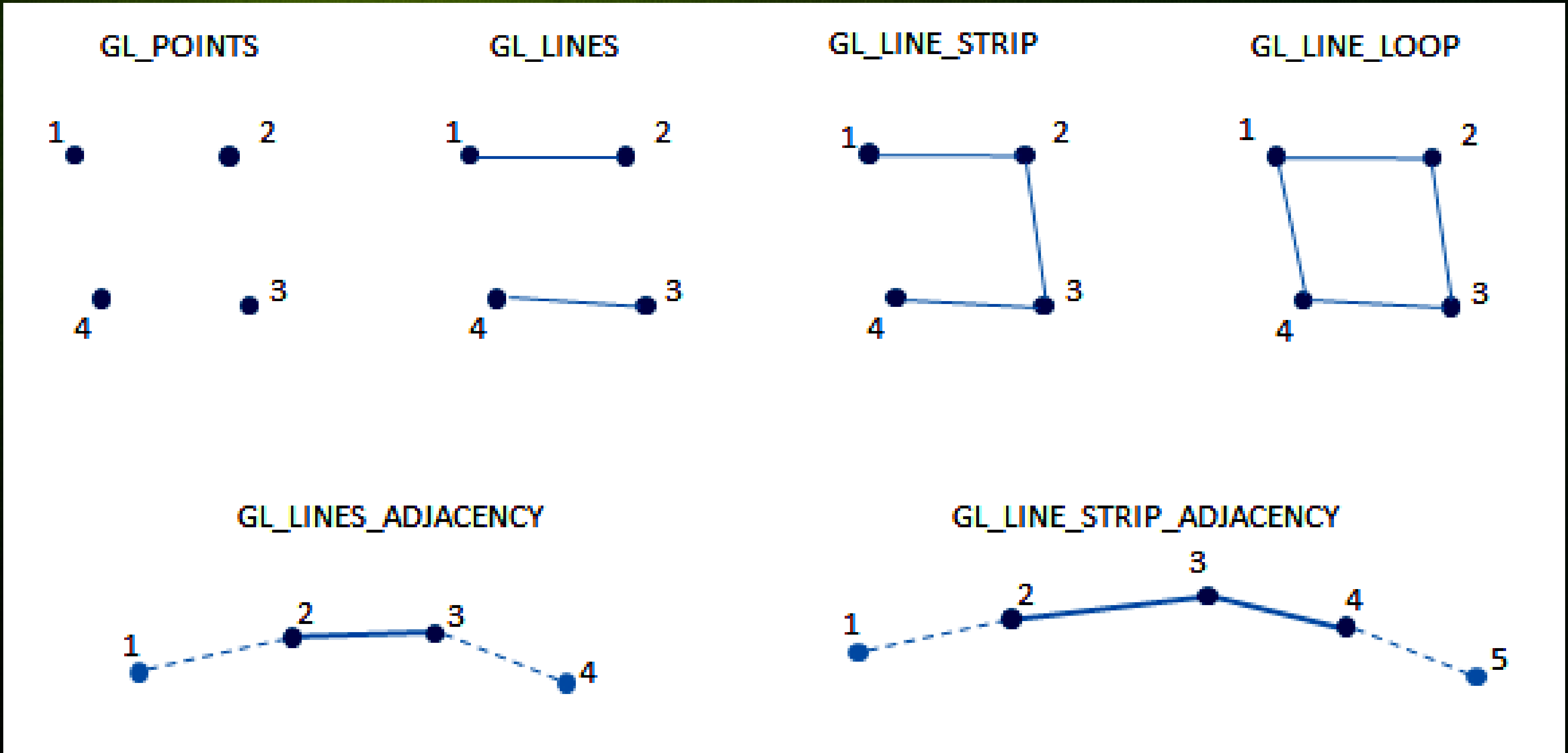
:: Primitive Assembly



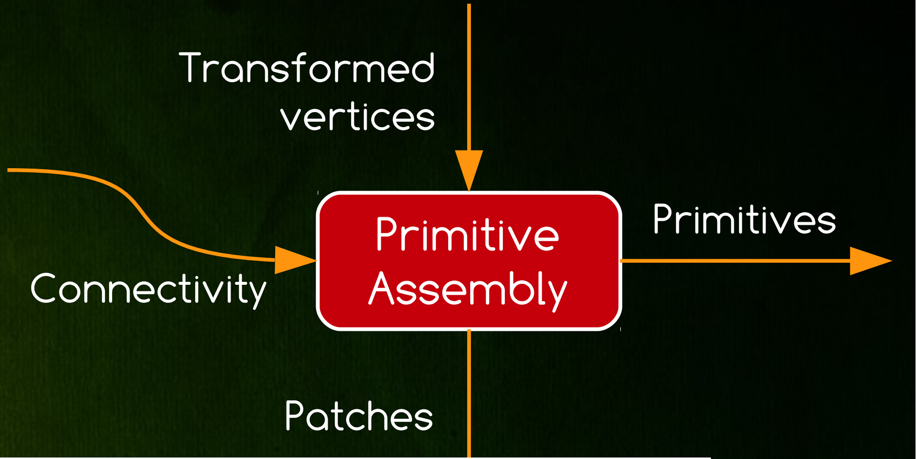
- Specification (Fixed)
  - Constructs list of primitives based on transformed vertices and respective connectivity informations
  - Input: Transformed vertices + connectivity info
  - Output: Ready primitives (lines, triangles...) or patches
- Main Purpose
  - Prepare complete primitive data for next stages (tessellation or geometry shader)

:: Primitive Assembly





:: Primitive Assembly



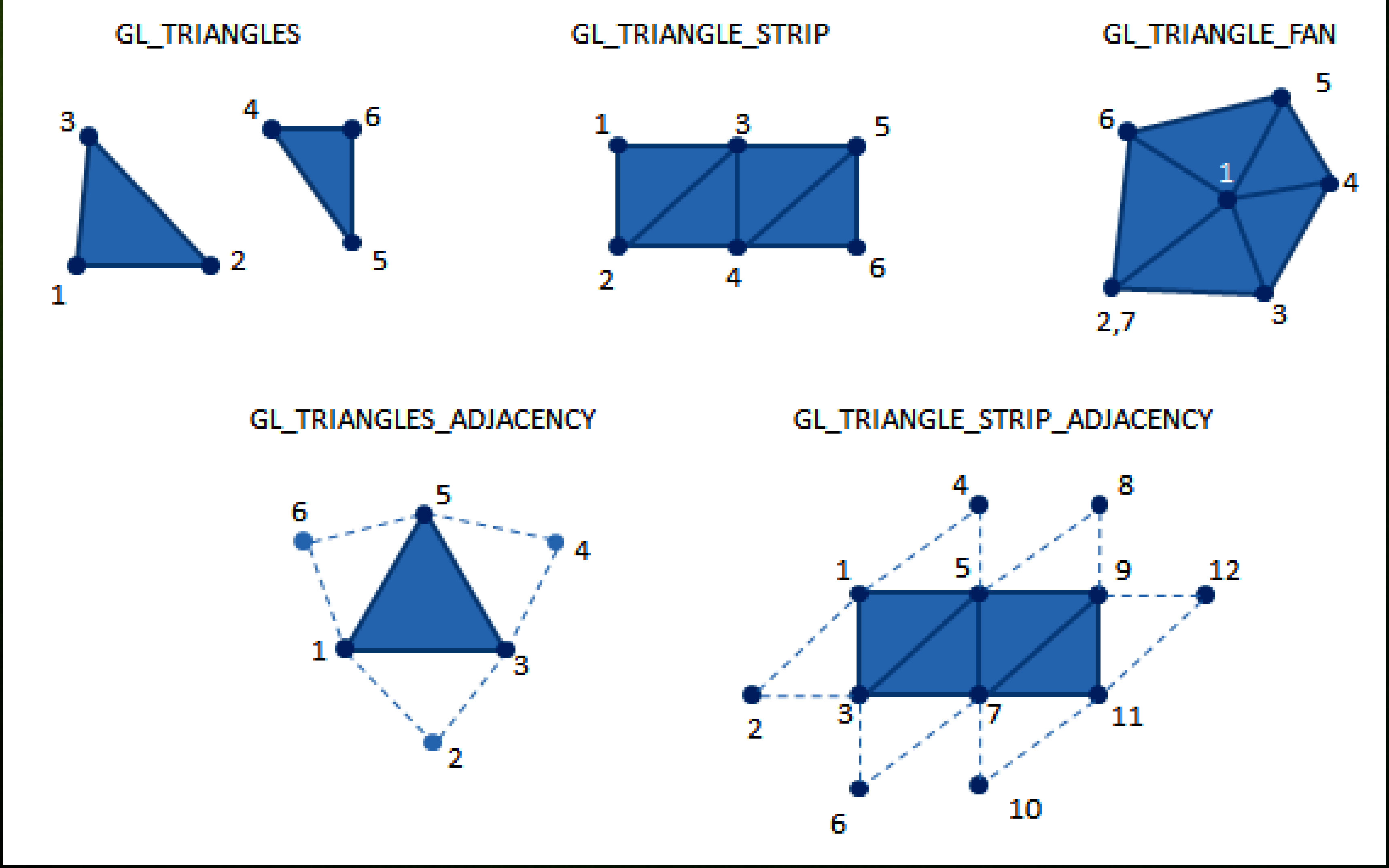
Transformed vertices

Connectivity

Primitive Assembly

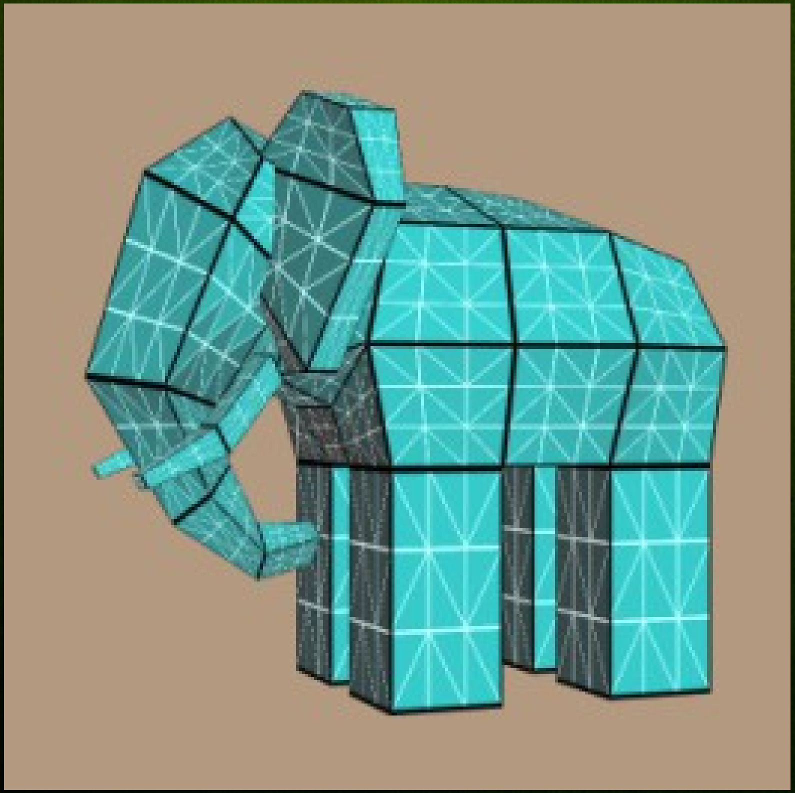
Primitives

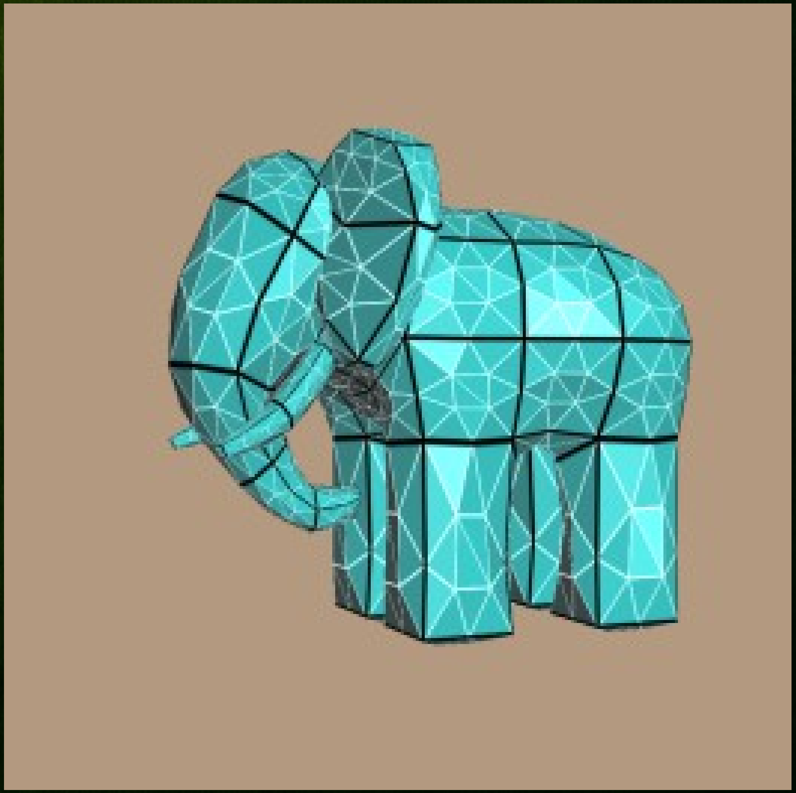
Patches



Tessellation Stages

- Quad subdivision with and without smoothing

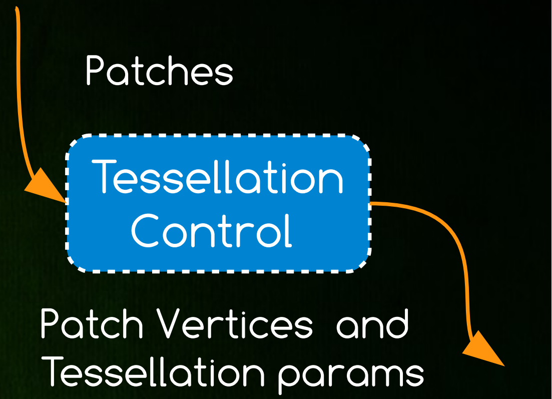




Tessellation Stages

- Specification
  - Three sub-stages (control, tessellator, evaluator)
  - Based on patch data creates new primitives
  - Input: Patches from primitive assembly (or geom)
  - Output: New subdivided primitives based on tessellation scheme
- Main Purpose
  - Dynamic subdivision of geometry
  - Local displacements
  - Level of detail

:: Tessellation Control



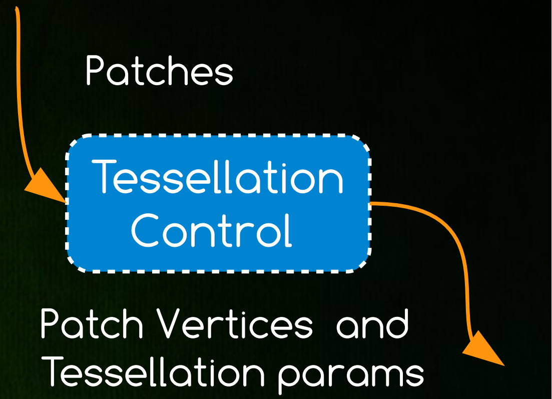
Patches

Tessellation Control

Patch Vertices and Tessellation params

- Specification
  - Set up tessellation levels along edges and faces
  - Input: Patch geometry (vertices + connectivity)
  - Output: Inner and Outer tessellation levels
- Main Purpose
  - Defines subdivision topology
  - Control how much are faces (inner) and edges (outer) subdivided during tessellation

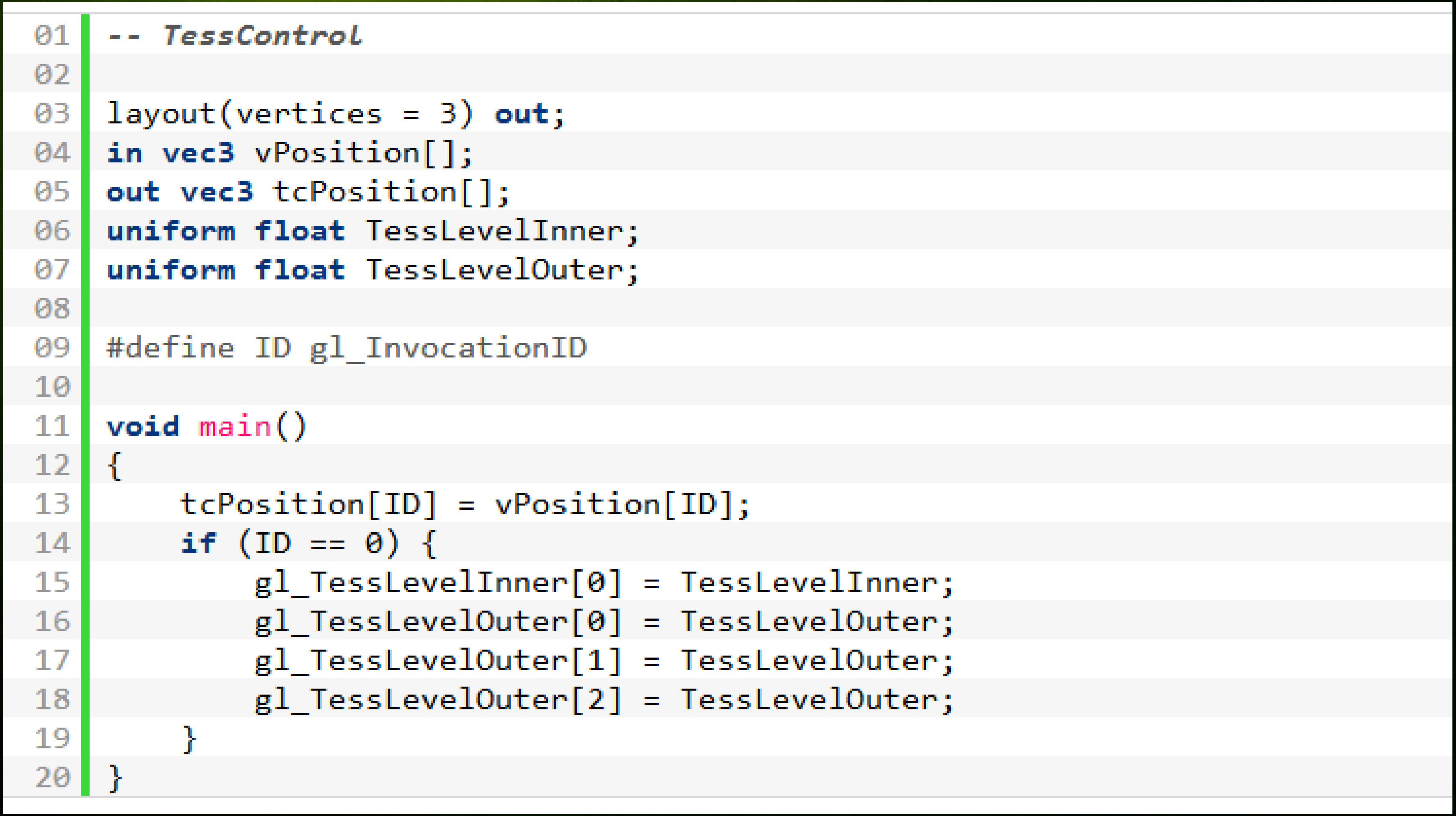
:: Tessellation Control



Patches

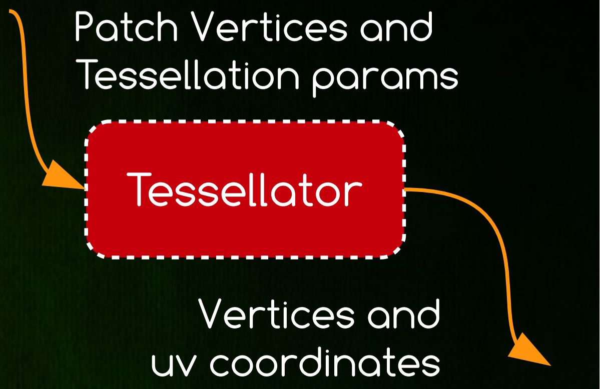
Tessellation Control

Patch Vertices and Tessellation params



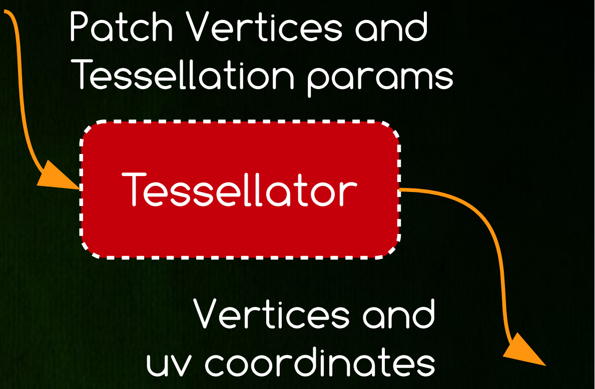
```
-- TessControl

layout(vertices = 3) out;
in vec3 vPosition[];
out vec3 tcPosition[];
uniform float TessLevelInner;
uniform float TessLevelOuter;

#define ID gl_InvocationID

void main()
{
    tcPosition[ID] = vPosition[ID];
    if (ID == 0) {
        gl_TessLevelInner[0] = TessLevelInner;
        gl_TessLevelOuter[0] = TessLevelOuter;
        gl_TessLevelOuter[1] = TessLevelOuter;
        gl_TessLevelOuter[2] = TessLevelOuter;
    }
}
```

:: Tessellator



- Specification (Fixed)
  - Given patch is subdivided on edges and faces based on tessellation levels
  - New sub-patches are created with resp. uv coords
  - Input: Patch vertices and tessellation levels
  - Output: New subdivision vertices and uv coords
- Main Purpose
  - Provides core tessellation functionality
  - Subdivision is fixed

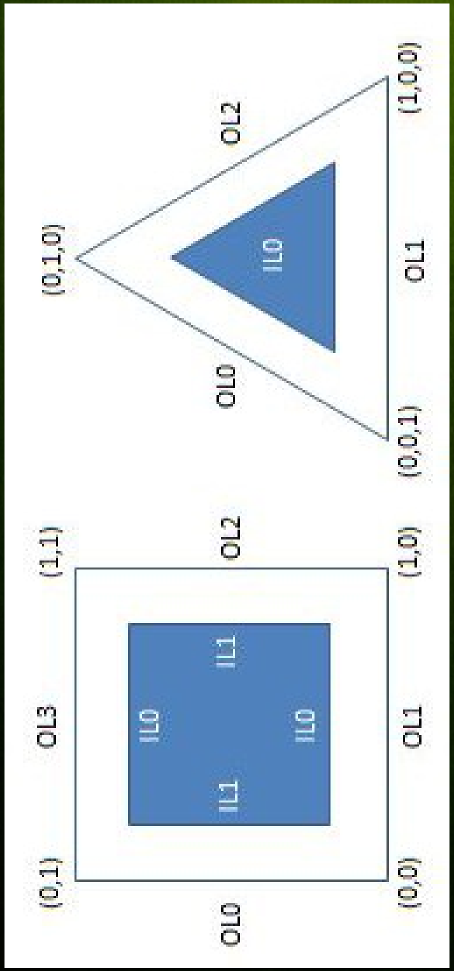
:: Tessellator

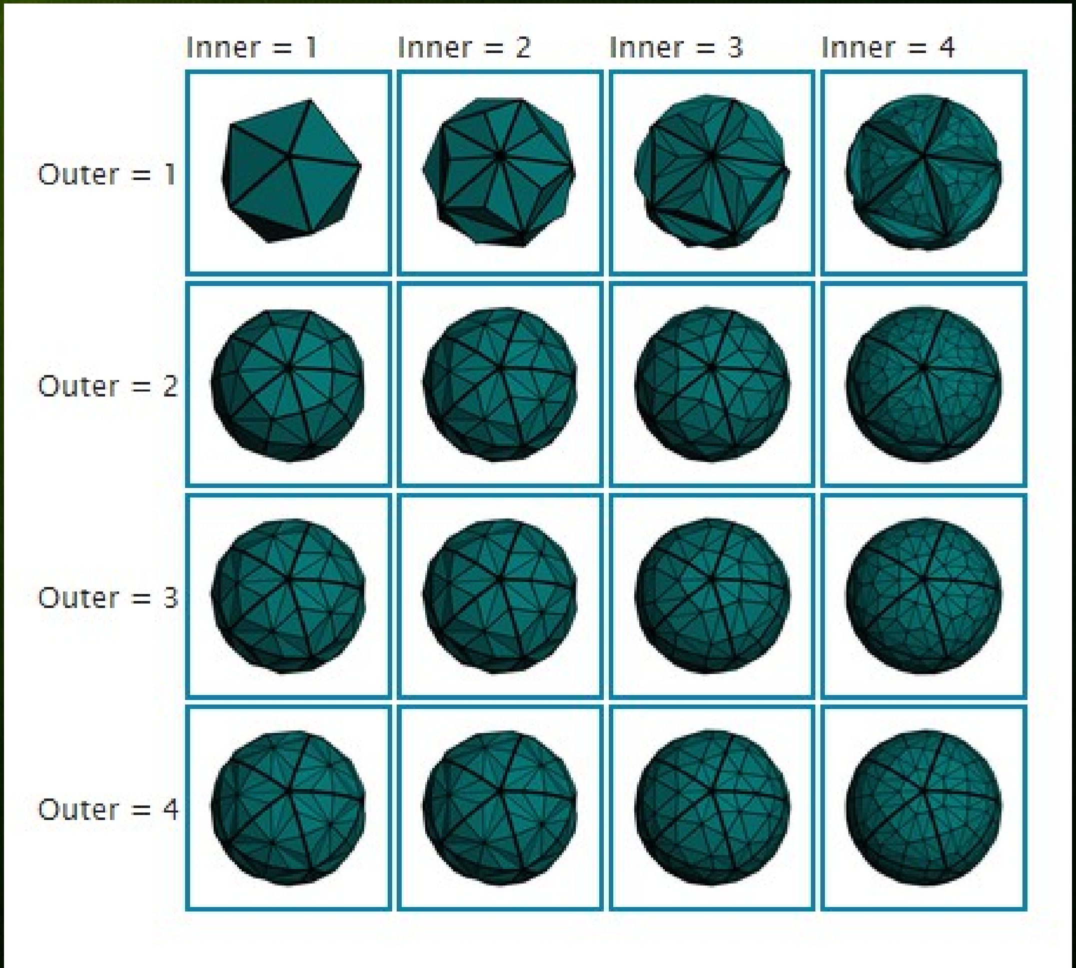


Patch Vertices and Tessellation params

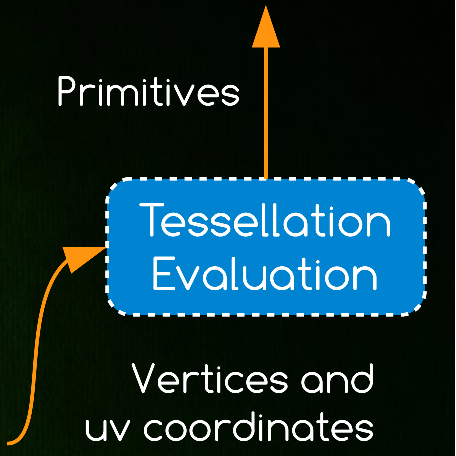
Tessellator

Vertices and uv coordinates



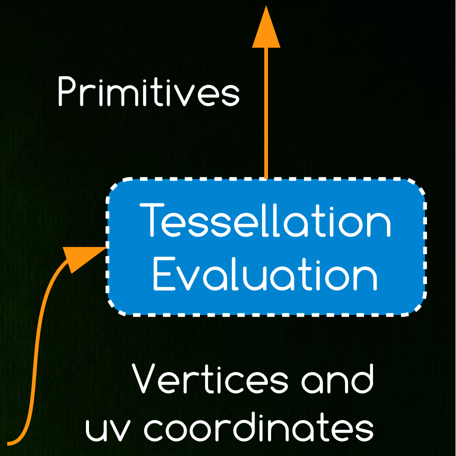


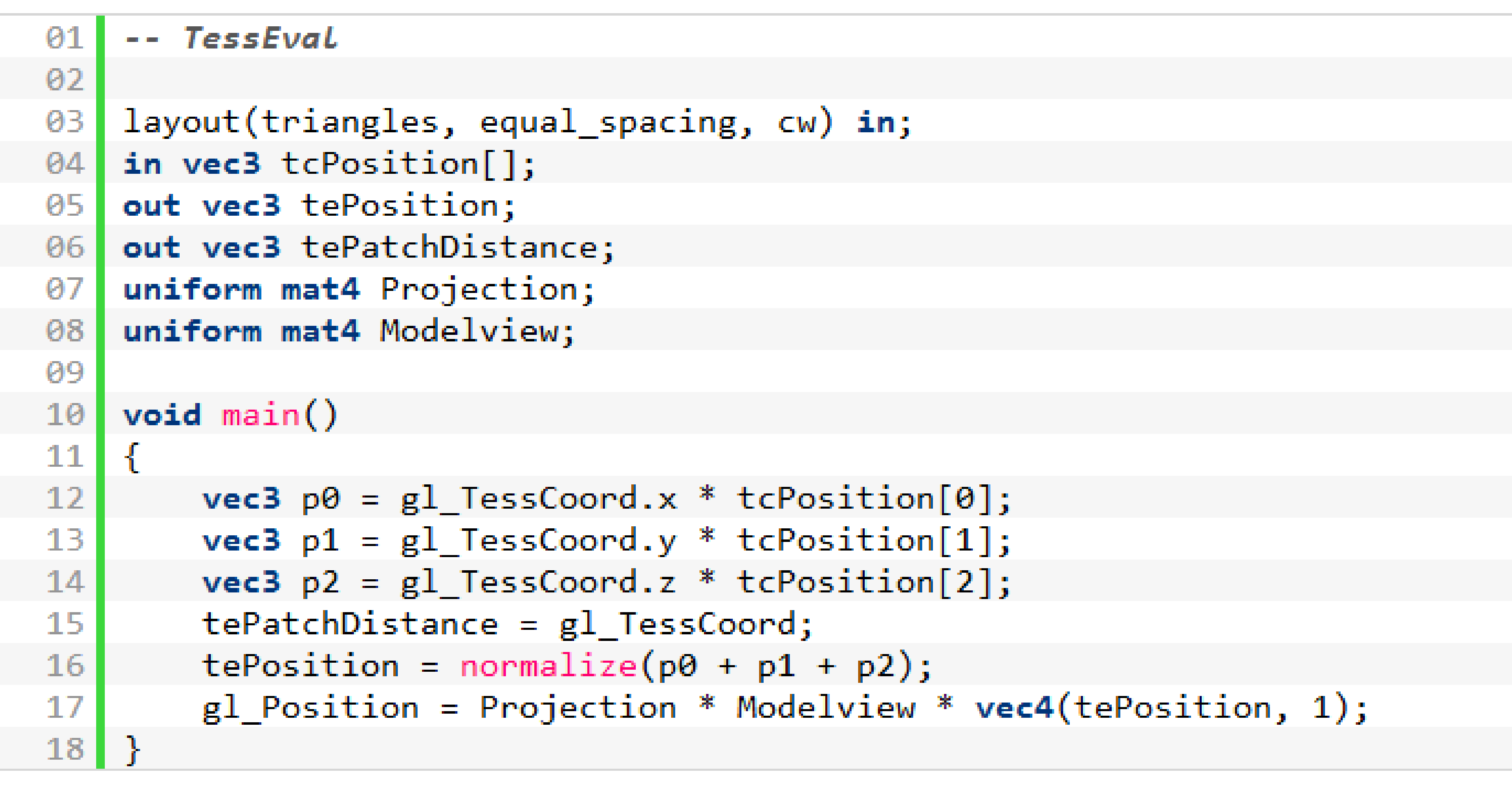
:: Tessellation Evaluation



- Specification (Programmable)
  - Based on uv coords (barycentric coords) evaluates positions of tessellated vertices
  - Input: Patch vertices and uv coordinates
  - Output: New primitives
- Main Purpose
  - Construct new primitives usable for next stages
  - Finalize the tessellation stage

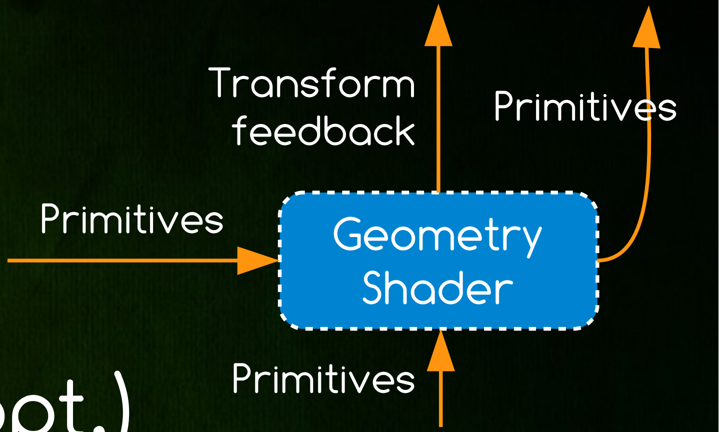
:: Tessellation Evaluation





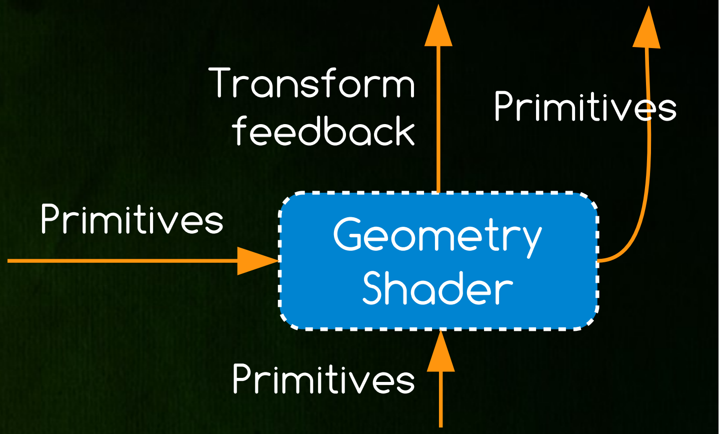
```
-- TessEval

layout(triangles, equal_spacing, cw) in;
in vec3 tcPosition[];
out vec3 tePosition;
out vec3 tePatchDistance;
uniform mat4 Projection;
uniform mat4 Modelview;

void main()
{
    vec3 p0 = gl_TessCoord.x * tcPosition[0];
    vec3 p1 = gl_TessCoord.y * tcPosition[1];
    vec3 p2 = gl_TessCoord.z * tcPosition[2];
    tePatchDistance = gl_TessCoord;
    tePosition = normalize(p0 + p1 + p2);
    gl_Position = Projection * Modelview * vec4(tePosition, 1);
}
```

:: Geometry Shader

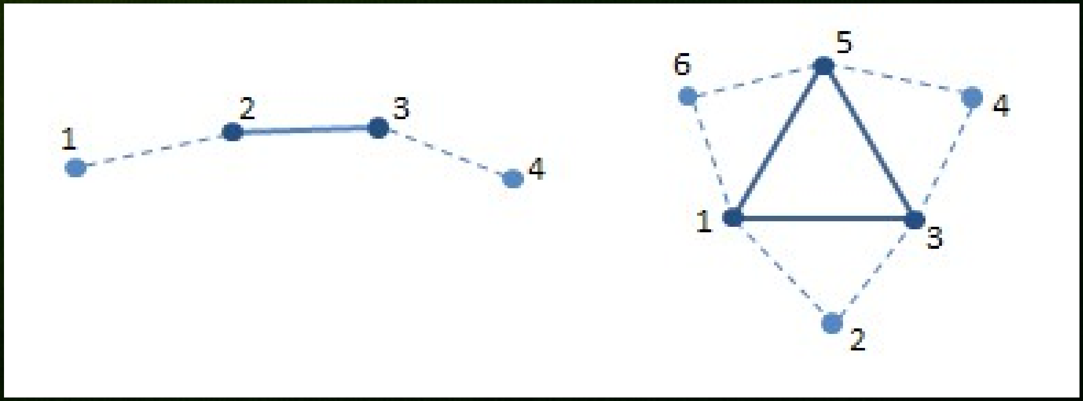


- Specification (Programmable, opt.)
  - Given a primitive Geometry Shader creates zero or more primitives
  - Input: primitives (points, lines, triangles)
  - Output: primitives (points, line-strip, triangle-strip)
- Main Purposes
  - Create new primitives (general tessellation)
  - Layered rendering
  - Transform feedback

:: Geometry Shader

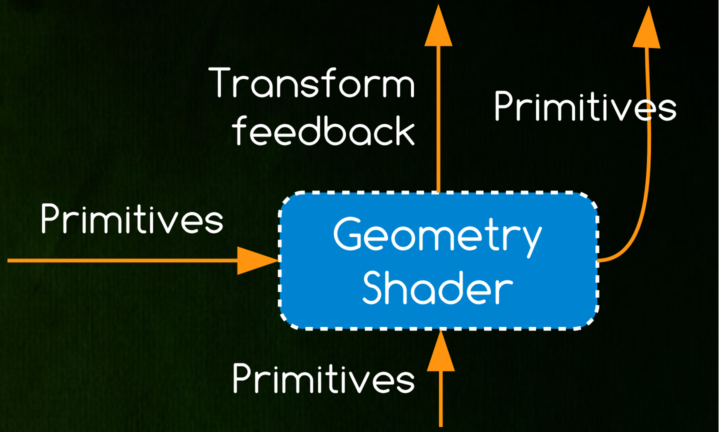


- Input Primitives
  - Points (1 vertex)
  - Lines (2 vertices), lines_adjacency (4 vertices)
  - Triangles (3 vertices), triangles_adjacency (6 ver.)
- Output Primitives
  - Points
  - Line_strip
  - Triangle_strip



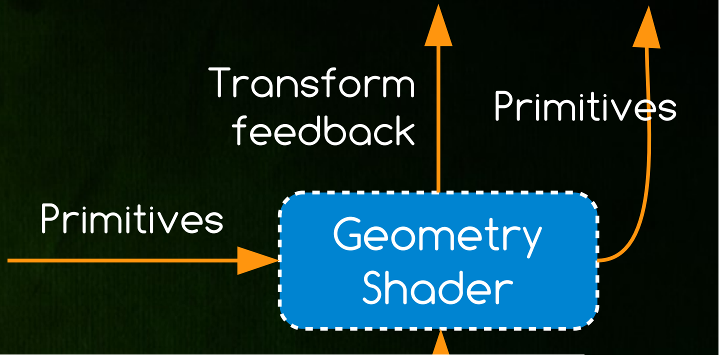
lines_adjacency triangles_adjacency

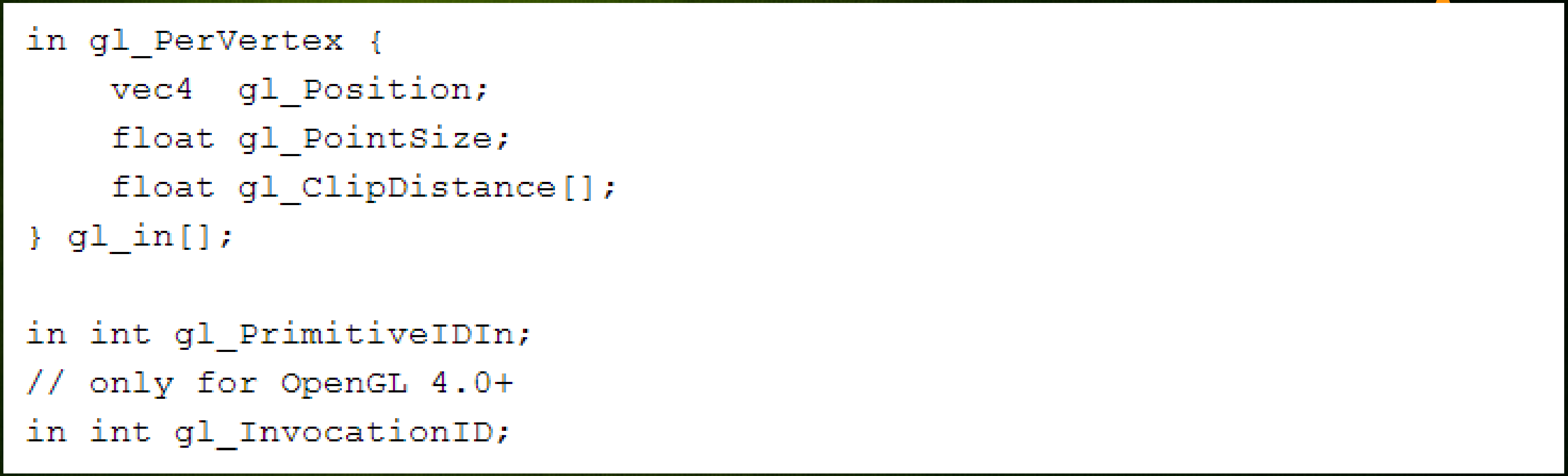
:: Geometry Shader



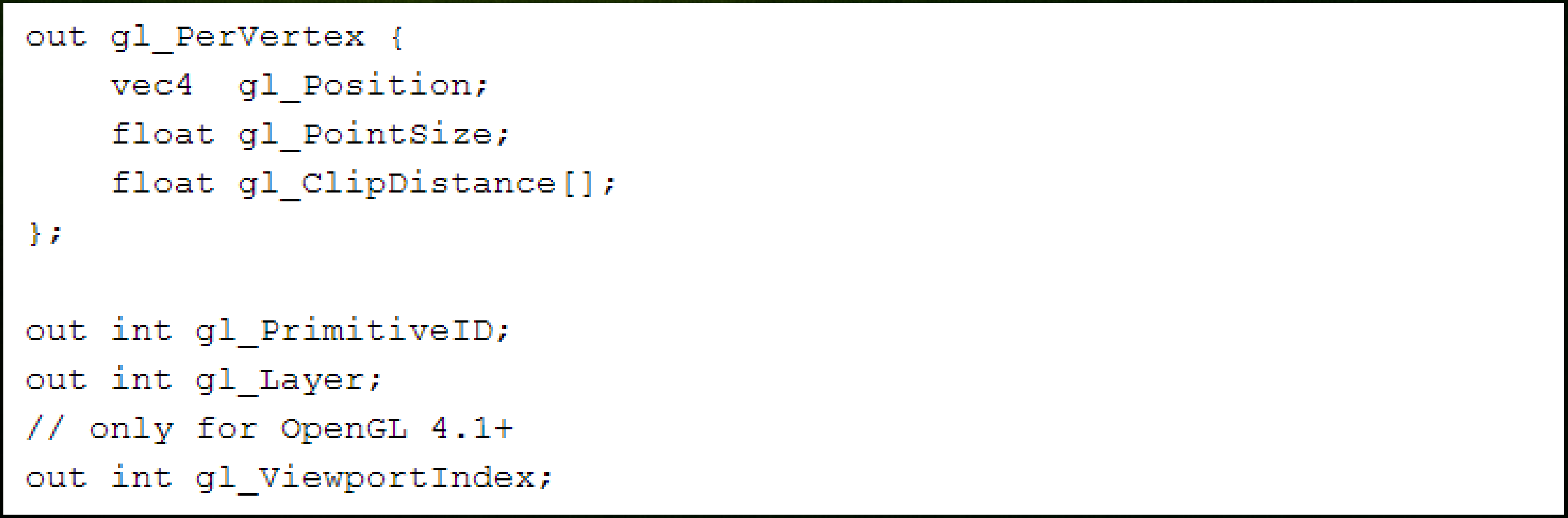
- Layered Rendering
  - Rendering the same geometry into different layers (frame buffers)
  - Eg. rendering into cubemap – 6 different layers
- Transform Feedback
  - We can run the vertex or geometry shader without rasterization and store modified vertices (primitives) into user defined buffers
  - Use user defined (transform feedback) buffers as geometry input for other vertex/geometry shaders

:: Geometry Shader



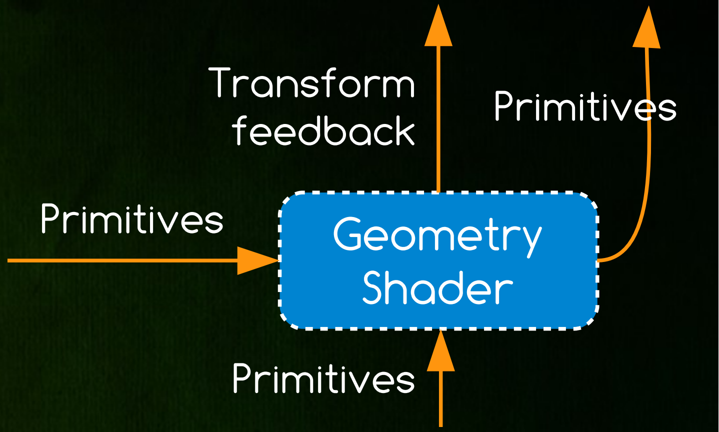


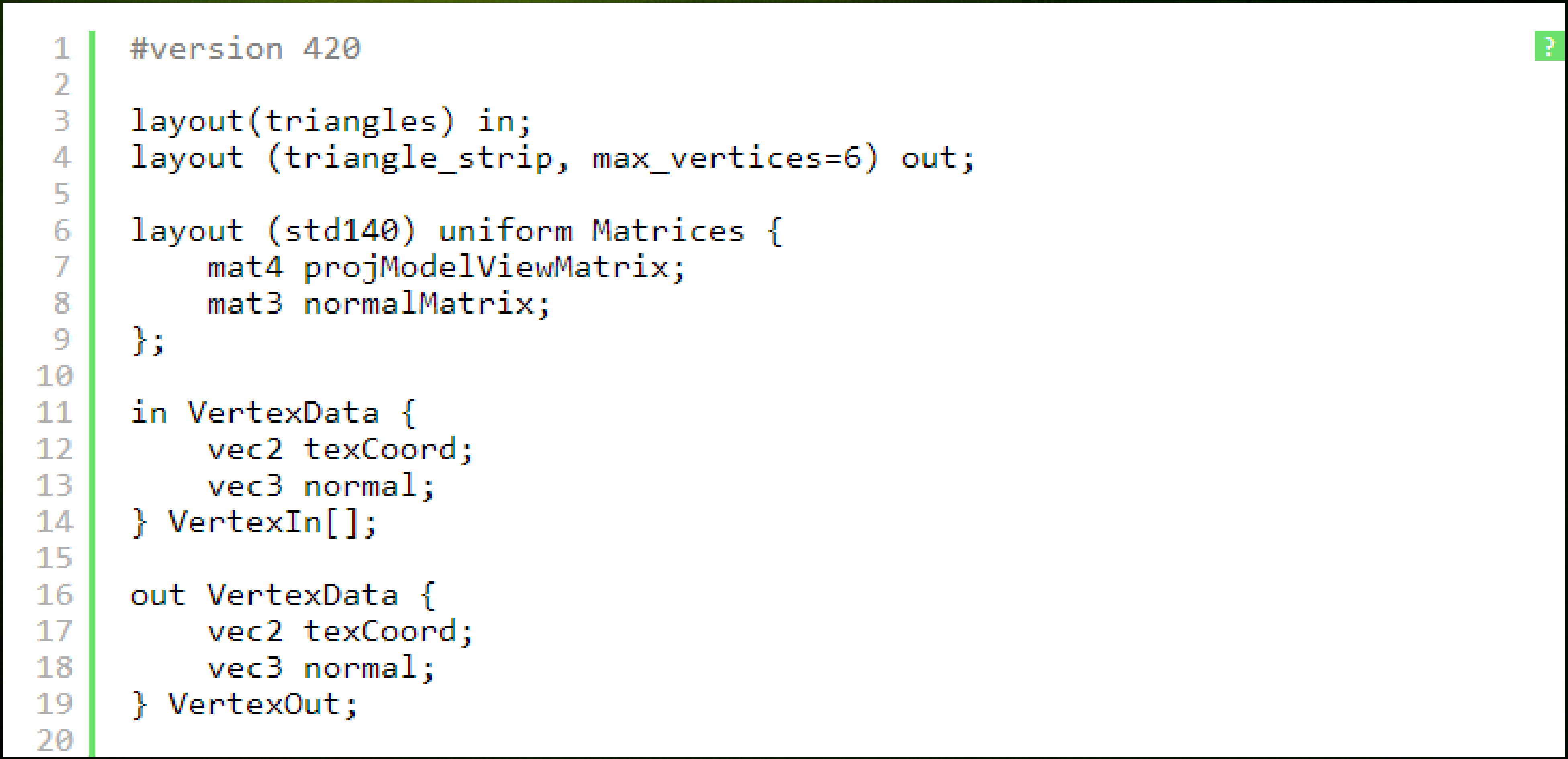
```
in gl_PerVertex {
    vec4  gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
} gl_in[];

in int gl_PrimitiveIDIn;
// only for OpenGL 4.0+
in int gl_InvocationID;
```



```
out gl_PerVertex {
    vec4  gl_Position;
    float gl_PointSize;
    float gl_ClipDistance[];
};

out int gl_PrimitiveID;
out int gl_Layer;
// only for OpenGL 4.1+
out int gl_ViewportIndex;
```

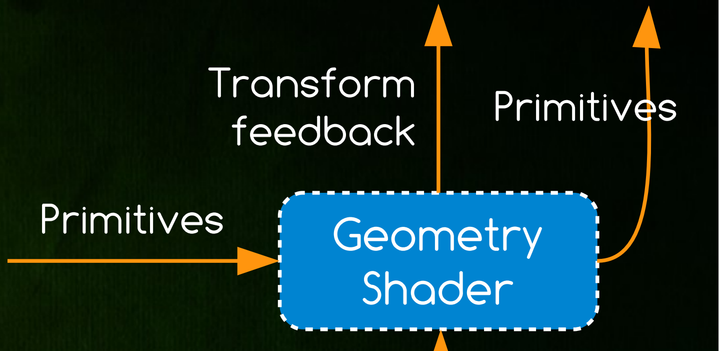
:: Geometry Shader

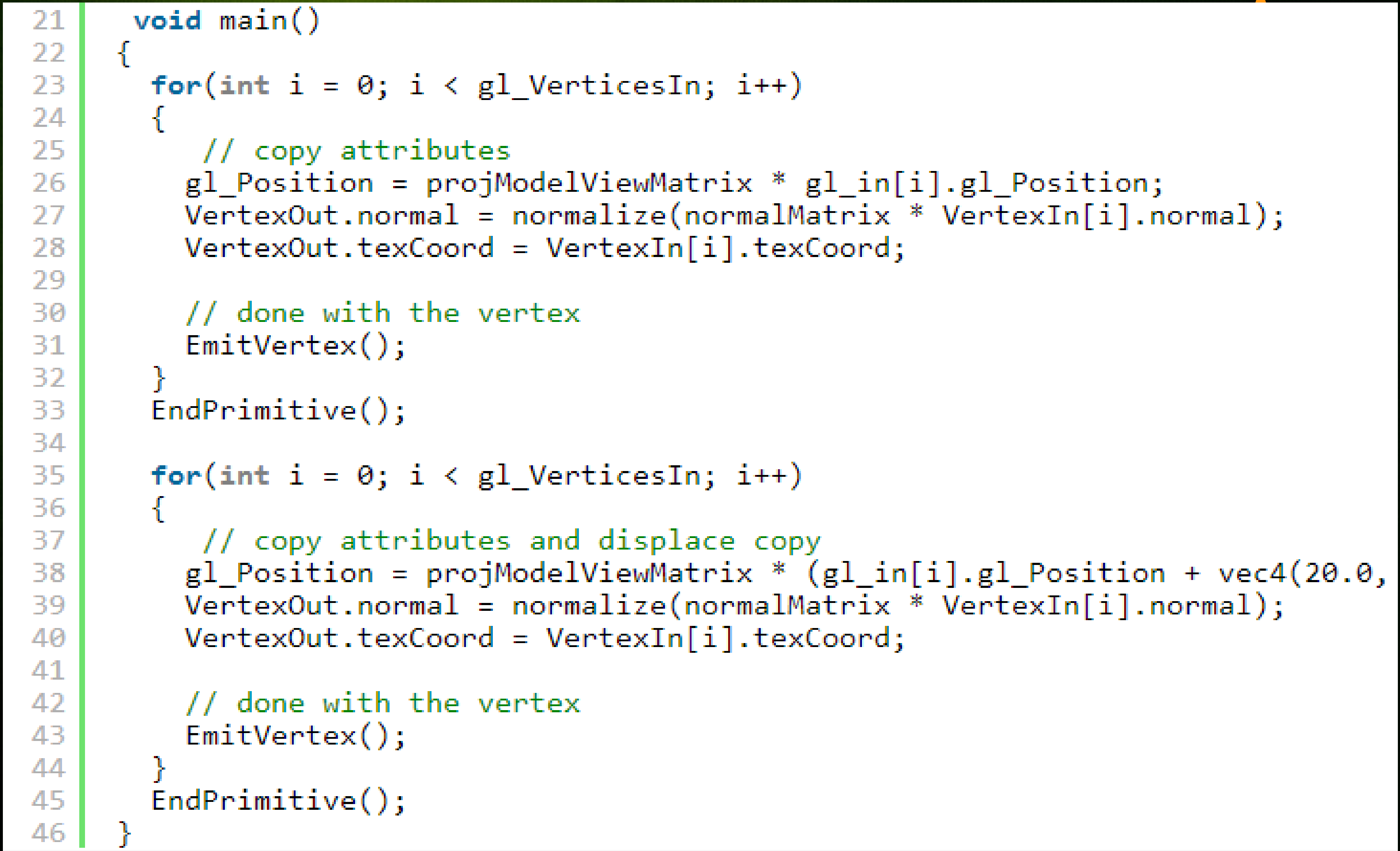




```
#version 420

layout(triangles) in;
layout (triangle_strip, max_vertices=6) out;

layout (std140) uniform Matrices {
    mat4 projModelViewMatrix;
    mat3 normalMatrix;
};

in VertexData {
    vec2 texCoord;
    vec3 normal;
} VertexIn[];

out VertexData {
    vec2 texCoord;
    vec3 normal;
} VertexOut;

```

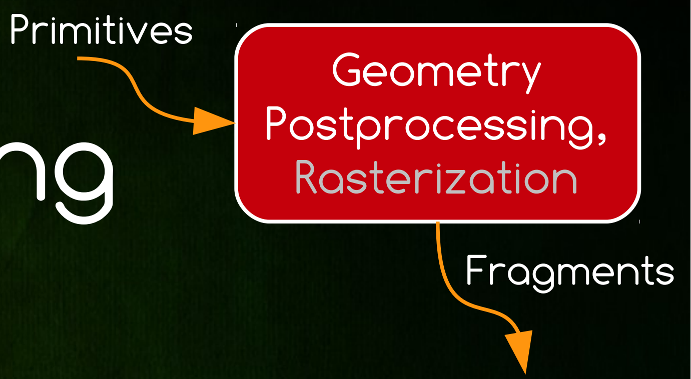
:: Geometry Shader



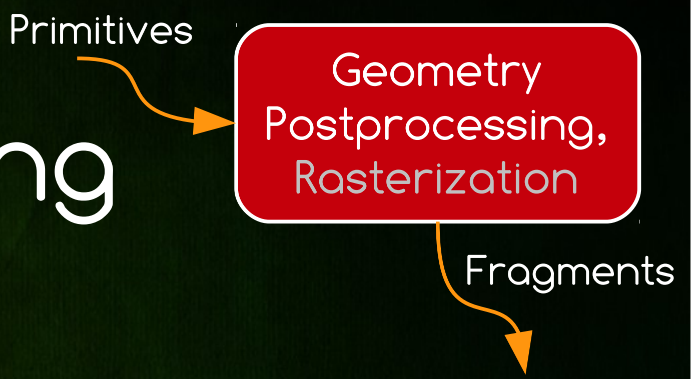


```
 void main()
{
  for(int i = 0; i < gl_VerticesIn; i++)
  {
     // copy attributes
    gl_Position = projModelViewMatrix * gl_in[i].gl_Position;
    VertexOut.normal = normalize(normalMatrix * VertexIn[i].normal);
    VertexOut.texCoord = VertexIn[i].texCoord;

    // done with the vertex
    EmitVertex();
  }
  EndPrimitive();

  for(int i = 0; i < gl_VerticesIn; i++)
  {
     // copy attributes and displace copy
    gl_Position = projModelViewMatrix * (gl_in[i].gl_Position + vec4(20.0,
    VertexOut.normal = normalize(normalMatrix * VertexIn[i].normal);
    VertexOut.texCoord = VertexIn[i].texCoord;

    // done with the vertex
    EmitVertex();
  }
  EndPrimitive();
}
```

:: Geometry Postprocessing



- Specification (Fixed)
  - View Frustum Clipping
  - Perspective Division (homogenous to viewport)
  - Viewport to Window Mapping (coords to pixels)
- Main Purposes
  - Final vertex processing before rasterization
  - Clipping → Perspective → Window



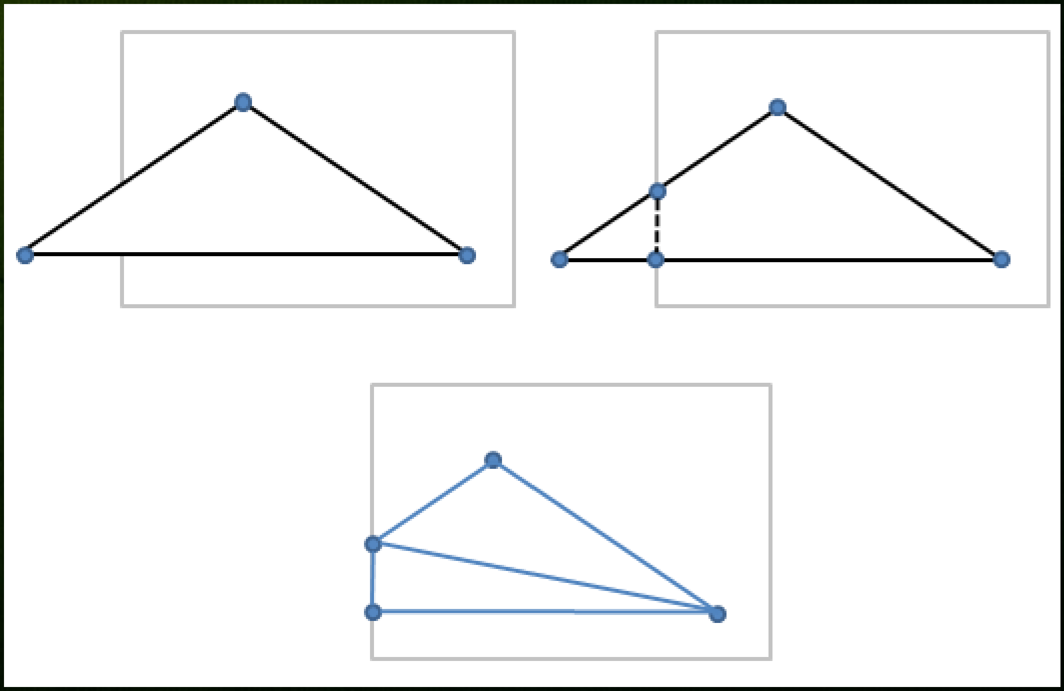
Primitives

Geometry Postprocessing, Rasterization

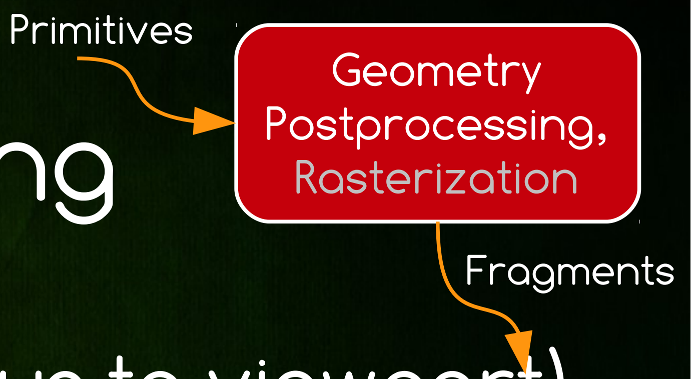
Fragments

:: Geometry Postprocessing

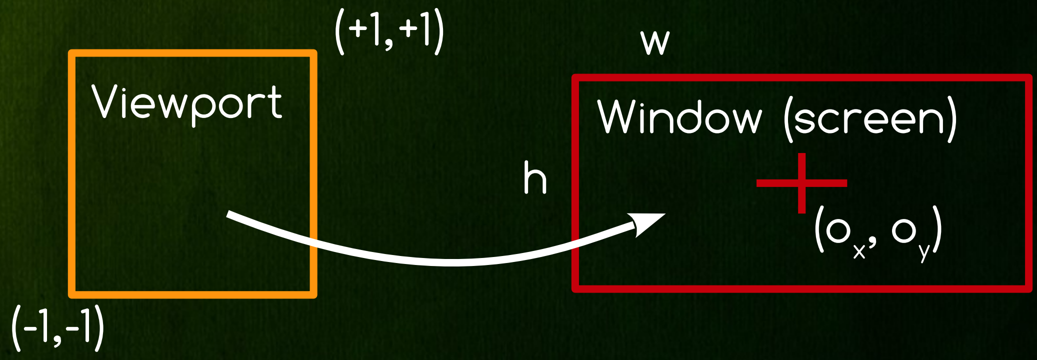
- View Frustum Clipping
  - Reject all geometry outside view frustum (volume)
  - Clip primitives which intersect clipping planes (view volume)
  - Vertex (x_c, y_c, z_c, w_c)
  - Is inside if
  - $-w_c <= x_c <= +w_c$
  - $-w_c <= y_c <= +w_c$
  - $-w_c <= z_c <= +w_c$



:: Geometry Postprocessing

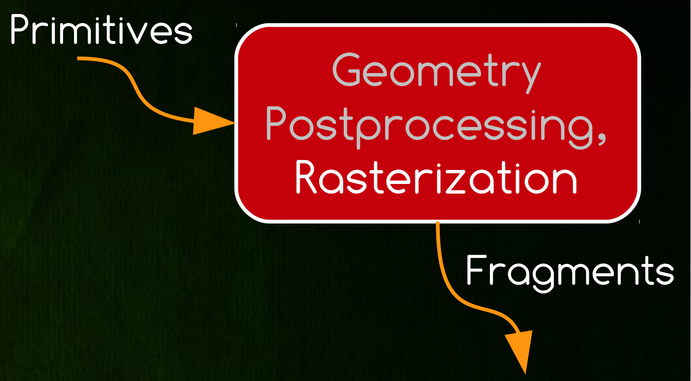


- Perspective Division (homogenous to viewport)
  - $(x_d, y_d, z_d) \rightarrow (x_c/w_c, y_c/w_c, z_c/w_c)$
  - Test if vertex is in clip volume reduces to
  - $-1 <= x_c <= +1$
  - $-1 <= y_c <= +1$
  - $-1 <= z_c <= +1$



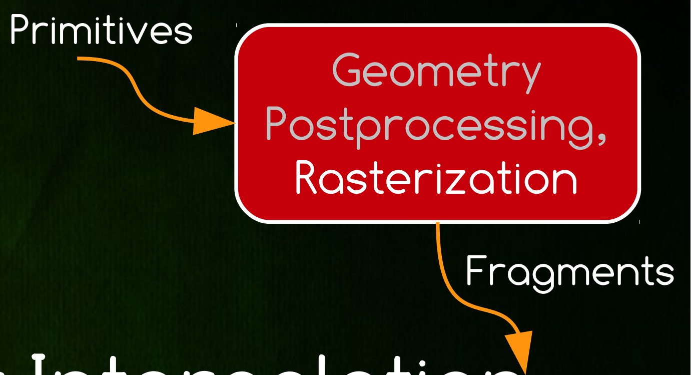
- Viewport to Window Mapping (coords to pixels)
  - $(x_w, y_w, z_w) = (x_d * w/2 + o_x, y_d * h/2 + o_y, (z_d + 1)/2)$

:: Rasterization

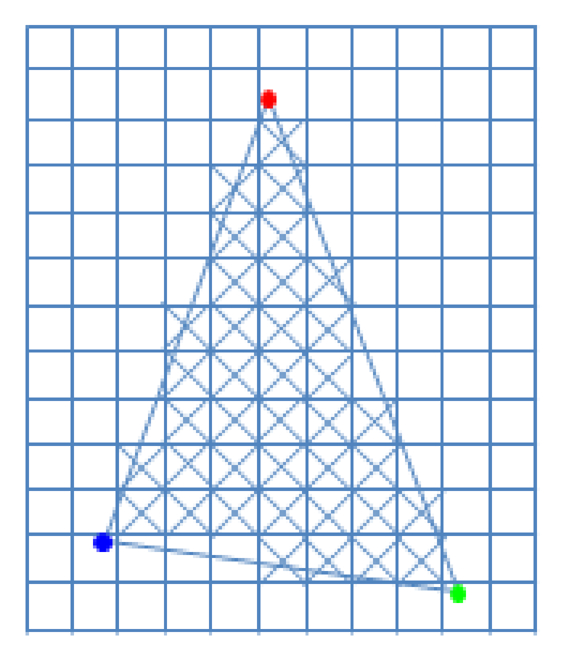


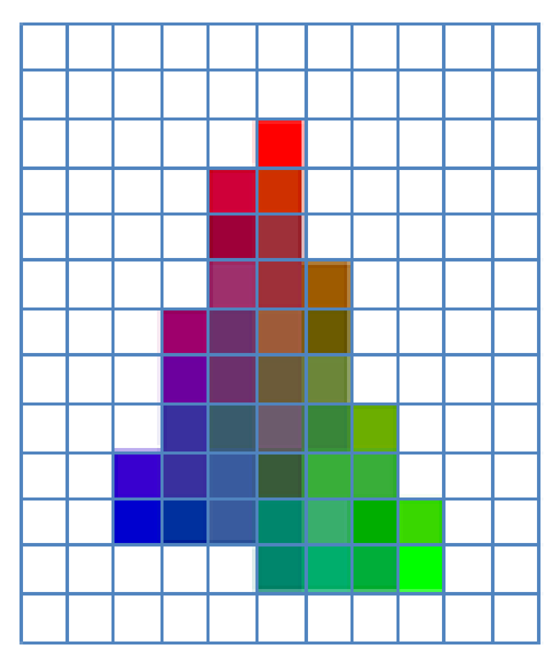
- Specification (Fixed)
  - Rasterization: Determine set of fragments (pixels) representing projected geometry primitives
  - Parameter Interpolation: Compute the attributes for each pixel based on the vertex attributes and the pixel's distance to each vertex screen position (barycentric coordinates)
- Main Purposes
  - Generate image (raster) representation of the given geometry – NOT the final pixel colors !

:: Rasterization

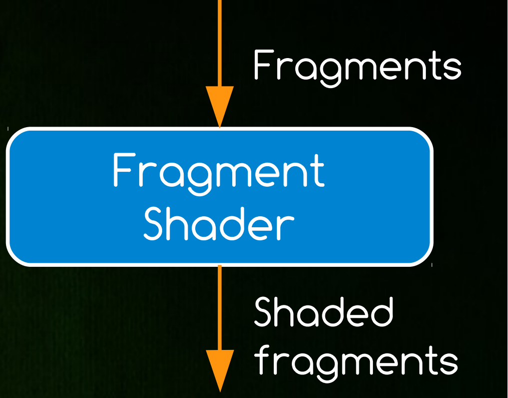


- Rasterization
  - DDA, Bresenham
  - Scanline Algorithm
- Parameter Interpolation
  - $p = a^{*}p_{a} + b^{*}p_{b} + c^{*}p_{c}$
  - $a+b+c = 1 \;|\; 0 <= a,b,c <= 1$



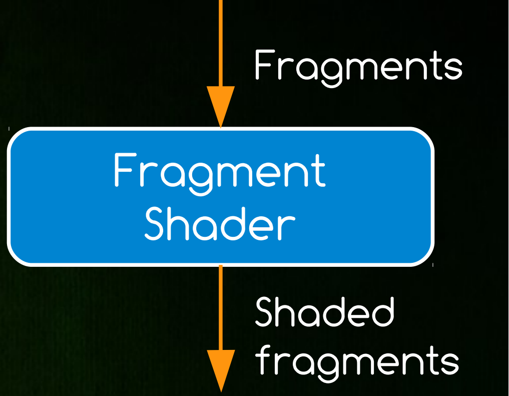


:: Fragment Shader



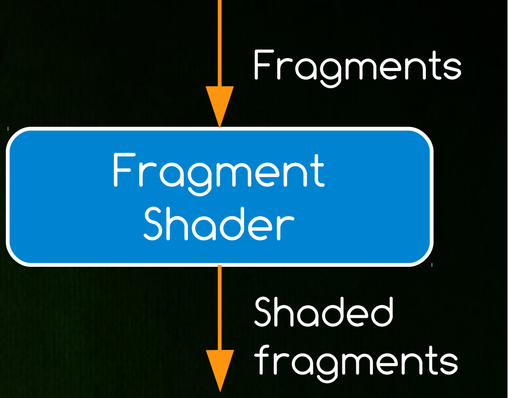
- Specification (Programmable)
  - Final pixel color calculation based on textures and uv coordinates, z-buffer, ...
  - Input: Fragments (frame buffer element) + interpolated data (barycentric coords)
  - Output: Pixels with final color

:: Fragment Shader



- Input parameters
  - **gl_FragCoord**: contains the fragments coordinate (x_f, y_f, z_f, w_f), where (x_f, y_f) is the pixels position on the window, zf is the depth, and w_f is $1/w_c$, where w_c is clip space position
  - **gl_FrontFacing**: tells the orientation of respective primitive. if culling is on all pixels have same value
  - **gl_PrimitiveID**: Index of primitive to which this fragment belongs to

:: Fragment Shader

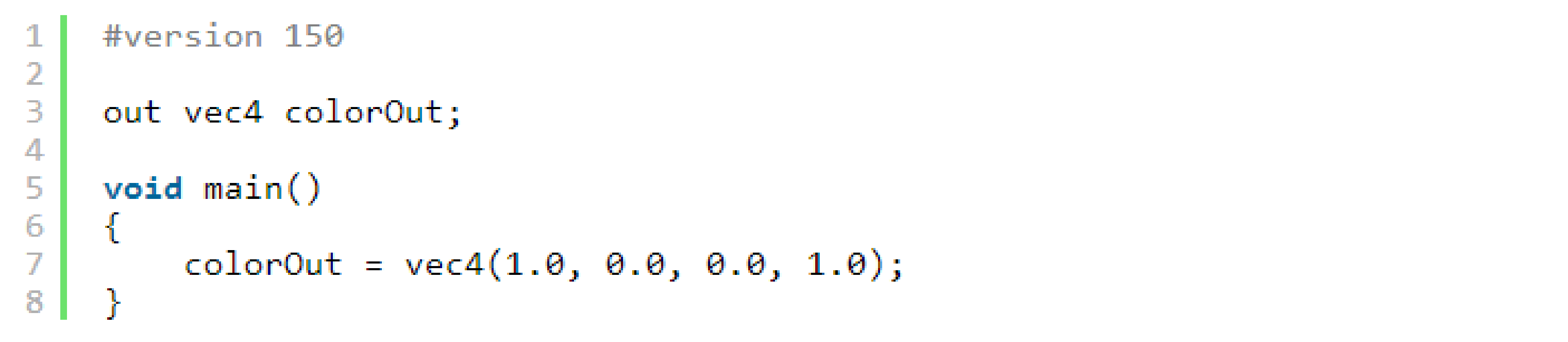


Fragments

Fragment Shader

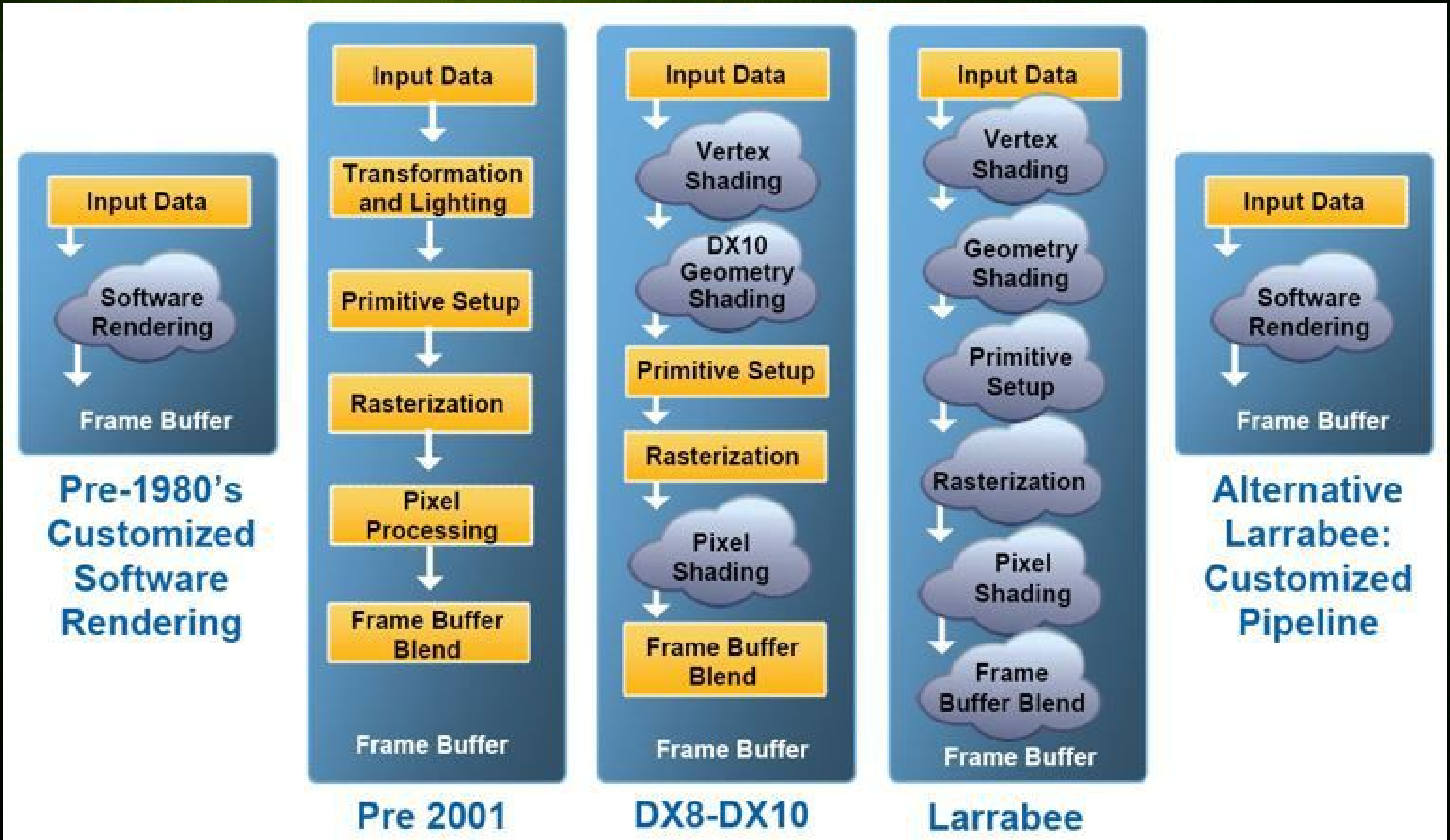
Shaded fragments

- Simple Pixel Shader



```
#version 150

out vec4 colorOut;

void main()
{
    colorOut = vec4(1.0, 0.0, 0.0, 1.0);
}
```

Rendering Pipeline Variants



The End