ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

centre adscrit a:

UNIVERSITAT DE BARCELONA

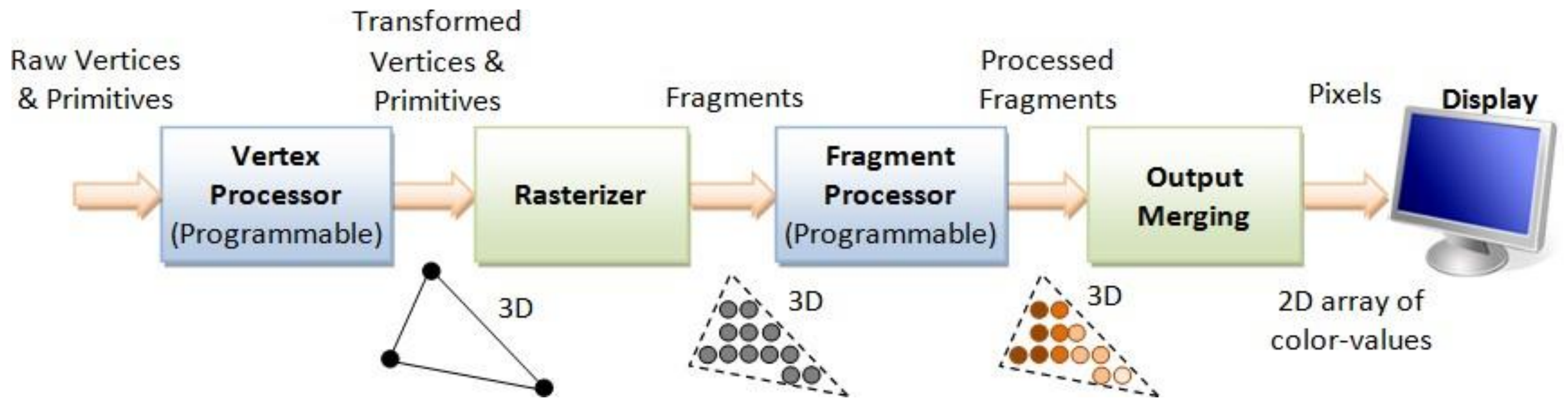
Computer Graphics

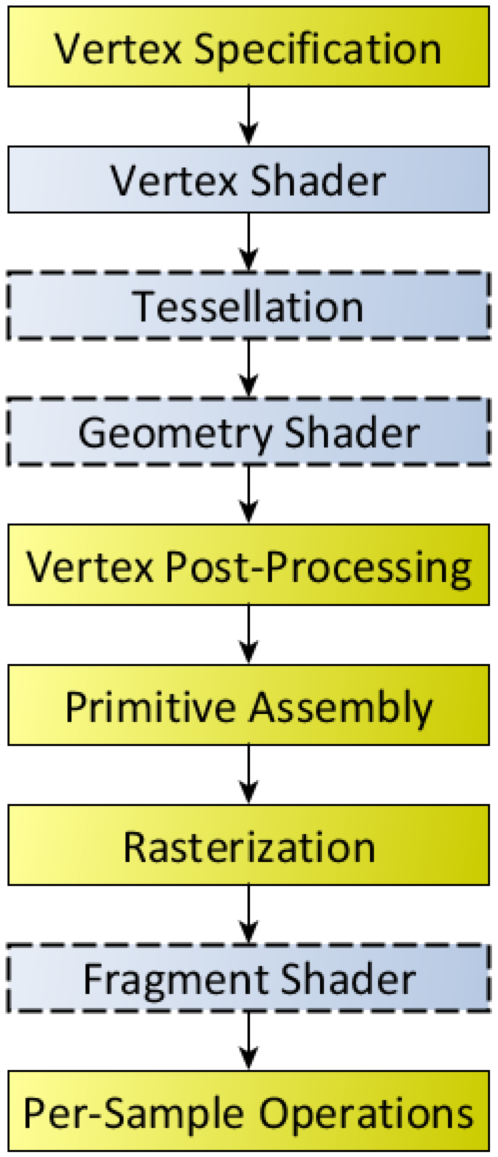
6. Introduction to Geometry Shaders

Dr Joan Llobera – joanllobera@enti.cat

Outline

1. Reminder of the graphics pipeline


Raw Vertices & Primitives
Vertex Processor (Programmable)
Transformed Vertices & Primitives
3D
Rasterizer
Fragments
3D
Fragment Processor (Programmable)
Processed Fragments
3D
Output Merging
Pixels
Display
2D array of color-values



2. A geometry shader example

```
const char* sphere_geomShader =
"#version 330\n\
layout(points) in;\n\
layout(triangle_strip, max_vertices = 4) out;\n\
out vec4 eyePos;\n\
out vec4 centerEyePos;\n\
uniform mat4 projMat;\n\
uniform float radius;\n\
vec4 nu_verts[4];\n\
void main() {\n\
vec3 n = normalize(-gl_in[0].gl_Position.xyz);\n\
vec3 up = vec3(0.0, 1.0, 0.0);\n\
vec3 u = normalize(cross(up, n));\n\
vec3 v = normalize(cross(n, u));\n\
nu_verts[0] = vec4(-radius*u - radius*v, 0.0); \n\
nu_verts[1] = vec4( radius*u - radius*v, 0.0); \n\
nu_verts[2] = vec4(-radius*u + radius*v, 0.0); \n\
nu_verts[3] = vec4( radius*u + radius*v, 0.0); \n\
centerEyePos = gl_in[0].gl_Position;\n\
for (int i = 0; i < 4; ++i) {\n\
eyePos = (gl_in[0].gl_Position + nu_verts[i]);\n\
gl_Position = projMat * eyePos;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}";
```

3. Input to geometry shaders

gl_in is implicitly declared as an array

```
in gl_PerVertex{\n\
       vec4 gl_Position;\n\
       float gl_pointSize;\n\
       float gl_clipDistance[];\n\
       float gl_CullDistance[];\n\
}gl_in[];\n\
```

To loop across gl_in is sufficient to do:

```
for (int n = 0; n<gl_in.length(); n++){\n\
…
}\n\
```

Notice that the size will depend on the kinds of inputs given to the geometry shader

3. Input to geometry shaders

Notice that the size will depend on the kinds of inputs given to the geometry shader:

Primitive Type	Input Array Size
Points	1
Lines	2
Triangles	3
Lines_adjacency	4
Triangles_adjacency	6

Notice also that, just as with vertex shaders, you can use:

```
gl_VertexID
```

For example:

```
gl_Position = vertices[gl_VertexID];
```

3. Input to geometry shaders

Notice additional input and output variables declared in our shader:

```
layout (triangles) in;\n\
layout (triangle_strip, max_vertices = 3) out;
```

The first line implies the geom. Shader will be processed once for each triangle.

Geometry shader Primitive Type	Accepted Drawing Command Modes
Points	GL_POINTS, GL_PATCHES
Lines	GL_LINES, GL_LINE_STRIP, GL_LINE_LOOP, GL_PATCHES
Triangles	GL_TRIANGLES, GL_TRANGLE_STRIP, GL_TRIANGLE_FAN, GL_PATCHES
Lines_adjacency	GL_LINES_ADJACENCY, GL_LINE_STRIP_ADJACENCY
Triangles_adjacency	GL_TRIANGLES_ADJACENCY, GL_TRIANGLE_STRIP_ADJACENCY

4. Output from geometry shaders

Notice additional input and output variables declared in our shader:

```
layout (triangles) in;\n\
layout (triangle_strip, max_vertices = 3) out;
```

Geometry shaders will generate ONLY:

```
points,
line strips, or
triangle strips
```

Geometry shaders will NOT generate fans, loops, or individual triangles.

4. Output from geometry shaders

A default output variable is:

```
gl_Position = …
```

It will often be followed by:

```
EmitVertex();
```

Which will generate a new vertex.

In addition, once we created several vertices and we want to assemble them in a primitive, we call:

```
EndPrimitive();
```

As said, primitives can be points, line strips, or triangle strips. The extra vertices not forming a primitive will be discarded.

4. Output from Geometry Shaders (vs Vertex Shaders)

We define outputs from **Vertex Shaders** as follows:

```
out vec4 position;
out vec3 normal;
out vec4 color;
out vec2 tex_coord;
```

In Geometry Shaders we must declare them as arrays, as follows:

```
in vec4 position[];
in vec3 normal[];
in vec4 color[];
in vec2 tex_coord[];
```

4. Interfaces in Geometry Shaders (vs Vertex Shaders)

In Vertex Shaders, we can declare more complex data structures as interfaces:

```
out VS_GS_INTERFACE
{
        out vec4 position;
        out vec3 normal;
        out vec4 color;
        out vec2 tex_coord[4];
}vs_out;
```

In the geometry shaders, we must declare them as arrays:

```
out VS_GS_INTERFACE
{
        out vec4 position;
        out vec3 normal;
        out vec4 color;
        out vec2
        tex_coord[4];
}gs_in[];
```

ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

UNIVERSITAT DE BARCELONA

4. Built-in members in Geometry Shaders (vs Vertex Shaders)

In Vertex Shaders, built-in inputs are:

```
gl_PrimitiveID
gl_InvocationID
```

These work in the same way that

```
gl_VertexID
```

Remember:

```
        gl_Position =
vertices[gl_VertexID];
```

However, in Geometry shaders, since they create geometry,

```
gl_PrimitiveID
gl_InvocationID
```

Are Outputs

In the geometry shaders, we must declare them as arrays:

```
gl_PrimitiveIDIn
gl_InvocationIDIn
```

5. Exercises

We will use the Hello Triangle exercise of last time as starting point

1. Make a trivial geometry shader (Paint again the same triangle)
2. Add an offset to the position of the triangle
3. Duplicate the geometry in the hsader
4. Pass a variable that moves with time

-> See separate slides and project for solutions

5. Give the shader only the triangle position, the shader will build the triangle

    hint: do not change the vertex shader, use only the first vertex input

6. Paint a cube face instead of a triangle

7. Make the cube face rotate on itself

8. Build the other 5 faces of the cube

    hint: pass the _MVP matrix as a uniform to have the movement with a mouse

    hint: to facilitate visibility, change the color of each face

9. Make the cube rotate around the cube center. Make the cube center be the only input given to the shader.

10. Integrate the fragment shader from the previous cube

Resources

- [geeks2018] unknown authors (last retrieved 03/2018) Simple introduction to geometry shaders http://www.geeks3d.com/20111117/simple-introduction-to-geometry-shader-in-glsl-part-2/
- [Sellers2016] Graham Sellers, Richard S. Writght, Jr. Nicholas Haemel (2016) *OpenGL SuperBible*, 6th Edition. Pearson education (**chapter 4 for data management), chapter 8, part 2, for geometry shaders**)
- [Kessenich2017] John Kessenich, Graham Sellers, Dave Shreiner (2017) OpenGL Programming guide. Ninth Edition. Pearson Education **(chapter 10 for geometry shaders)**