centre adscrit a:
UNIVERSITAT DE BARCELONA

Computer Graphics

6. Geometry Shader Exercises

Dr Joan Llobera – joanllobera@enti.cat

Exercises geometry shaders 1 to 4

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

Hello Triangle (reminder of past class)

```
namespace  MyGeomShader {
void myInitCode();
void myCleanupCode();
void myRenderCode(double currentTime);
}
```

```
////////////////////////////////////My Geom Shader
namespace  MyGeomShader {

GLuint myRenderProgram;
GLuint myVAO;

void myCleanupCode() {
glDeleteVertexArrays(1, &myVAO);
glDeleteProgram(myRenderProgram);
}
GLuint myShaderCompile(void) {
static const GLchar * vertex_shader_source[] =
{
"#version 330\n\
\n\
void main() {\n\
const vec4 vertices[3] = vec4[3](vec4( 0.25, -0.25, 0.5, 1.0),\n\
   vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4( -0.25,  -0.25, 0.5, 1.0));\n\
gl_Position = vertices[gl_VertexID];\n\
}"
};
static const GLchar * fragment_shader_source[] =
{
"#version 330\n\
\n\
out vec4 color;\n\
\n\
void main() {\n\
color = vec4(0.0,0.8,1.0,1.0);\n\
};
```

```
GLuint vertex_shader;
GLuint fragment_shader;
GLuint program;

vertex_shader = glCreateShader(GL_VERTEX_SHADER);
glShaderSource(vertex_shader, 1, vertex_shader_source, NULL);
glCompileShader(vertex_shader);


fragment_shader = glCreateShader(GL_FRAGMENT_SHADER);
glShaderSource(fragment_shader, 1, fragment_shader_source, NULL);
glCompileShader(fragment_shader);

program = glCreateProgram();
glAttachShader(program, vertex_shader);
glAttachShader(program, fragment_shader);
glLinkProgram(program);

glDeleteShader(vertex_shader);
glDeleteShader(fragment_shader);

return program;
}

void  myInitCode(void) {
myRenderProgram = myShaderCompile();
glCreateVertexArrays(1, &myVAO);
glBindVertexArray(myVAO);
}

void myRenderCode(double currentTime) {

glUseProgram(myRenderProgram);
glDrawArrays(GL_TRIANGLES, 0, 3);

}
}
```

Trivial geom shader (1)

```
static const GLchar * geom_shader_source[] =
{ "" };
```

Inside myShaderCompile:

```
static const GLchar * geom_shader_source[] =
{ "" };
//…

GLuint geom_shader;

//…

geom_shader =
glCreateShader(GL_GEOMETRY_SHADER);
glShaderSource(geom_shader, 1,
geom_shader_source, NULL);
glCompileShader(geom_shader);

//…

glAttachShader(program, geom_shader);
```

Trivial geom shader (2)

```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 3)
out;\n\
void main()\n\
{\n\
const vec4 vertices[3] = vec4[3](vec4(0.25, -
0.25, 0.5, 1.0),\n\
vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4(-0.25,  -0.25, 0.5, 1.0)); \n\
for (int i = 0; i <3; i++)\n\
{\n\
gl_Position = vertices[i] +
gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}" };
```

b) Add an offset

```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 3)
out;\n\
void main()\n\
{\n\
vec4 offset = vec4(0.5,0.5,0.0,0.0);\n\
for (int i = 0; i <3; i++)\n\
{\n\
gl_Position = gl_in[i].gl_Position+offset;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}" };
```

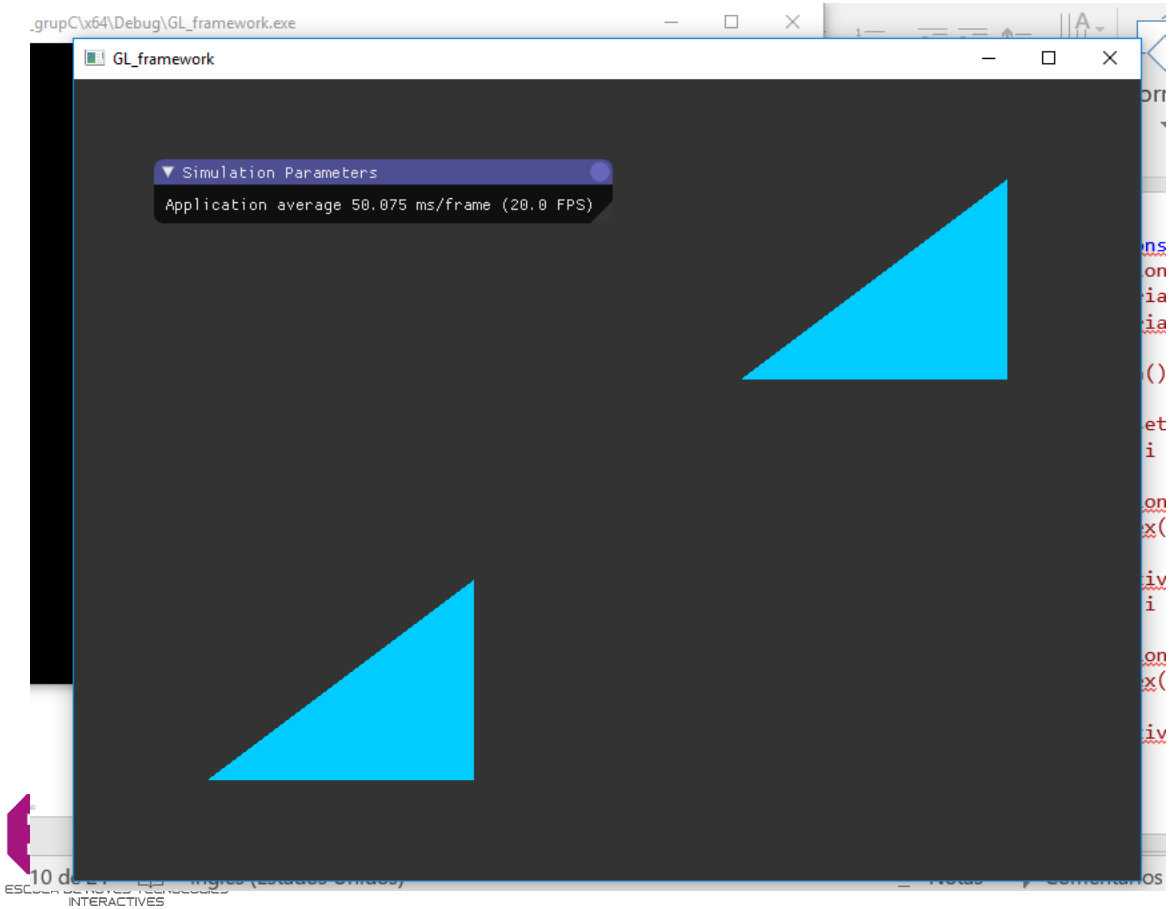
c) Duplicate the geometry in the shader

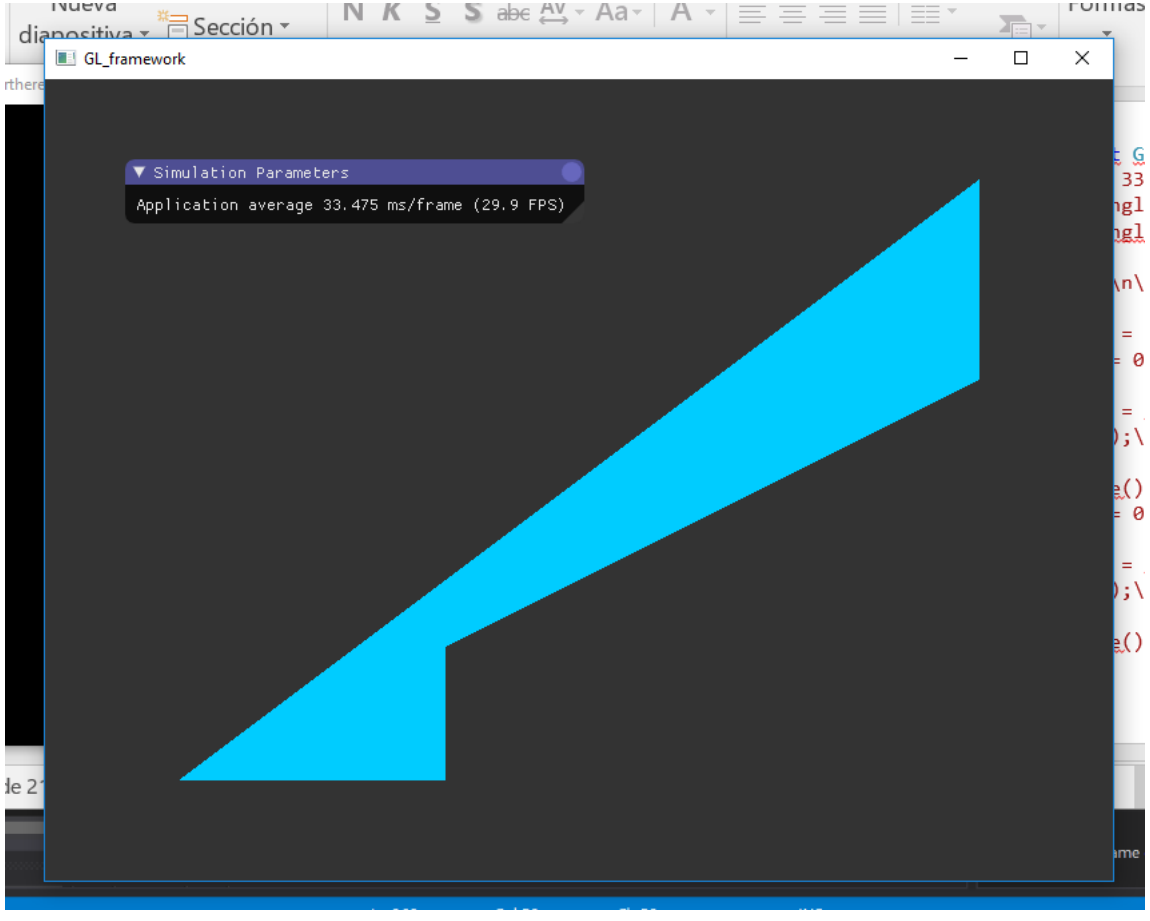
Notice that removing the first `EndPrimitive();\n\`

Will result in 2 united triangles (notice the triangle strip output)

```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 6)
out;\n\
void main()\n\
{\n\
vec4 offset = vec4(0.5,0.5,0.0,0.0);\n\
for (int i = 0; i <3; i++)\n\
{\n\
gl_Position = gl_in[i].gl_Position-offset;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
for (int i = 0; i <3; i++)\n\
{\n\
gl_Position = gl_in[i].gl_Position+offset;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}" };
```

Comparison with and without EndPrimitive()





d) Make Both triangles move

```
void myRenderCode(double currentTime) {
glUseProgram(myRenderProgram);
glUniform1f(glGetUniformLocation(myRender
Program, "time"), (GLfloat) currentTime);
glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

Notice the previous program passes time in the same way before we did things like:

```
glUniform1f(glGetUniformLocation(myRenderProgram,
"time"), (GLfloat) currentTime);
```

Is similar to:

```
//glUniformMatrix4fv(glGetUniformLocation(cubeProgram
, "mvpMat"), 1, GL_FALSE, glm::value_ptr(RV::_MVP));
//glUniform4f(glGetUniformLocation(cubeProgram,
"color"), 0.6f, 0.6f, 0.6f, 1.f);
```

```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
uniform float time;\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 6)
out;\n\
void main()\n\
{\n\
vec4 offset = vec4(0.5-sin(time),0.5,0.0,0.0);
\n\
for (int i = 0; i<3; i++)\n\
{\n\
gl_Position = gl_in[i].gl_Position+offset;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
for (int i = 0; i<3; i++)\n\
{\n\
gl_Position = gl_in[i].gl_Position - offset;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}" };
```

Caution!

Certain graphic cards do not like the instruction:

```
glCreateVertexArrays(1, &myVAO);
```

If we do this, they will create an exception.
You can use this alternative method to create the vertex array:

```
glGenVertexArrays(1, &myVAO);
```

Exercises geometry shaders 5 to 10

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 Cube example

Exercise 5 (1/4). Simplest solution

```
//SIMPLEST SOLUTION: same vertex shader
static const GLchar * vertex_shader_source[] =
{
"#version 330\n\
\n\
void main() {\n\
const vec4 vertices[3] = vec4[3](vec4( 0.25, -0.25, 0.5, 1.0),\n\
   vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4( -0.25,  -0.25, 0.5, 1.0));\n\
gl_Position = vertices[gl_VertexID];\n\
}" };
```

```
// we add three coordinates, from the gl_in[0]

static const GLchar * geom_shader_source[] =
{ "#version 330\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 6) out;\n\
void main()\n\
{\n\
const vec4 vertices[3] = vec4[3](vec4(0.25, -0.25, 0.5, 1.0),\n\
vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4(-0.25,  -0.25, 0.5, 1.0)); \n\
for (int i = 0; i <3; i++)\n\
{\n\
gl_Position = vertices[i] +gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}" };
```

Exercise 5 (2/4). But… this draws three triangles!

Notice: changing two aparent inocuous things paints 3 triangles. Why?

```
static const GLchar *
vertex_shader_source[] =
{
"#version 330\n\
\n\
void main() {\n\
const vec4 vertices[3] = vec4[3](vec4(
1, -0.25, 0.5, 1.0),\n\
vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4( -0.25,  -0.25, 0.5, 1.0));\n\
gl_Position = vertices[gl_VertexID];\n\
}"};
```

```
void myRenderCode(double currentTime) {
glUseProgram(myRenderProgram);
glDrawArrays(GL_TRIANGLES, 0, 9);
}
```

Answer: the geometry shader is considering EACH vertex as a seed for a new triangle. We did not see it before because drawArrays only painted 3 vertexes, and because the first position defined in the vertex_shader_source resulted in a shader out from the frame.

Exercise 5 (3/4). simplest solution

To draw 1 single triangles, we only need to give one vertex:

```
static const GLchar *
vertex_shader_source[] =
{
"#version 330\n\
\n\
void main() {\n\
const vec4 vertex = vec4( 1, -0.25,
0.5, 1.0);\n\
gl_Position = vertex;\n\
}" };
```

To pas the position as a uniform:

A vertex shader that does nothing:

```
static const GLchar *
vertex_shader_source[] =
{
"#version 330\n\
\n\
void main() {\n\
}" };
```

Exercise 5 (4/4). passing a uniform to adjust pos through code

A vertex shader that does nothing:

```
static const GLchar * vertex_shader_source[] =
{
"#version 330\n\
\n\
void main() {\n\
}" };
```

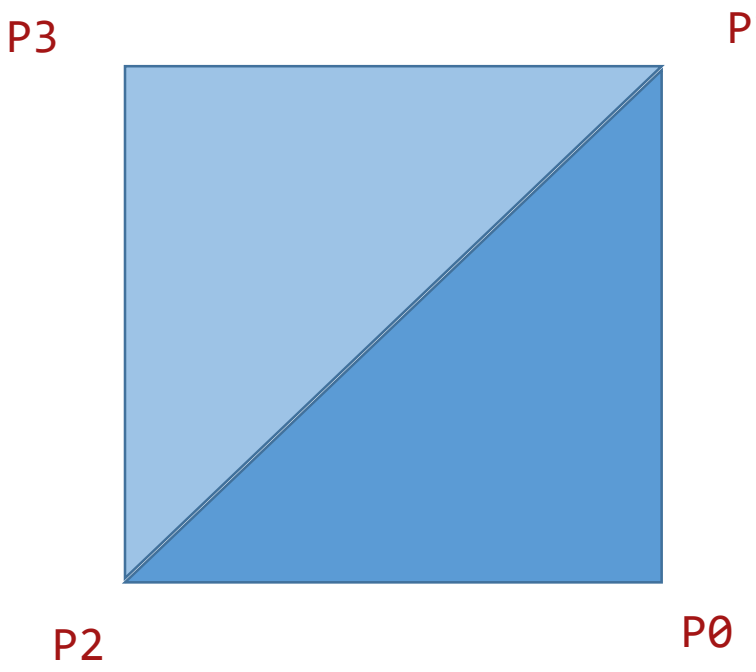
The render code passes a variable:

```
void myRenderCode(double currentTime) {
glUseProgram(myRenderProgram);
glUniform1f(glGetUniformLocation(myRenderProgram, "time"),
(GLfloat)currentTime);
glUniform4f(glGetUniformLocation(myRenderProgram, "initPos"), 0.0f,0.0f,-
1.0f,0.0f);

glDrawArrays(GL_TRIANGLES, 0, 9);
}
```

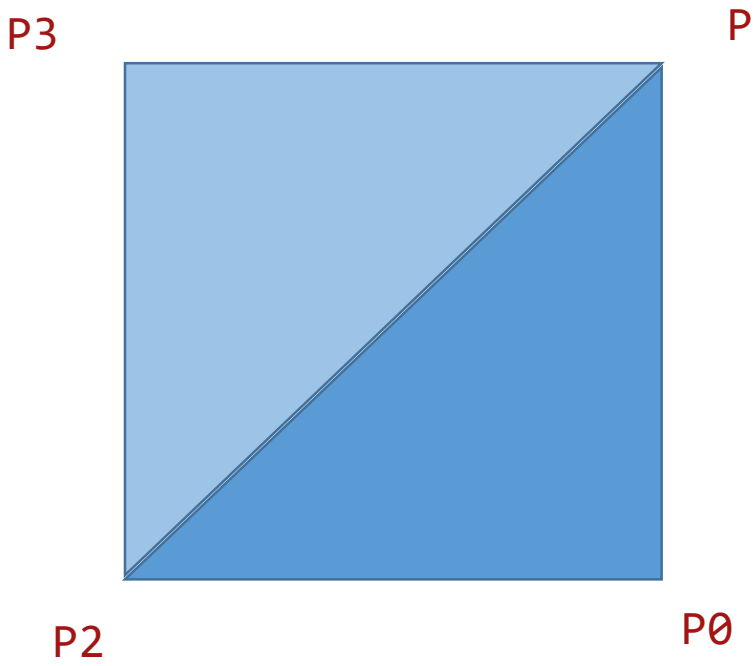
```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
uniform float time;\n\
uniform vec4 initPos;\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices =4)out;\n\
const vec4 vertices[4] = vec4[4](vec4(0.25, -0.25, 0.5, 1.0),\n\
vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4(-0.25,  -0.25, 0.5, 1.0),\n\
vec4(-0.25,  0.25, 0.5, 1.0)); \n\
void main(){\n\
for(int i=0; i< 4; i++){\n\
gl_Position = initPos +vertices[i];\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}" };
```

Exercise 6 (1/2). Paint a cube face




```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
uniform float time; \n\
layout(triangles) in; \n\
layout(triangle_strip, max_vertices = 6) out;
\n\
void main() {\n\
const vec4 vertices[4] = vec4[4](vec4(0.25, -
0.25, 0.5, 1.0),\n\
vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4(-0.25,  -0.25, 0.5, 1.0),\n\
vec4(-0.25,  0.25, 0.5, 1.0)); \n\
for (int i = 0; i < 4; ++i) {\n\
gl_Position = vertices[i] +
gl_in[0].gl_Position; \n\
EmitVertex(); \n\
}\n\
EndPrimitive(); \n\
}" };
```

Exercise 6 (2/2). Paint a cube face



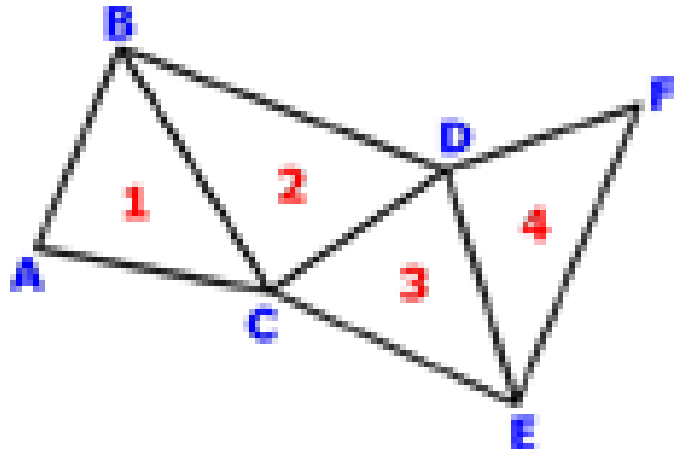
Notice the order: the first triangle needs to be drawn anti-clockwise (right hand rule).

The second is defined from the last 2 points + a new one.

Exercise6. Details: Normals in a triangle strip

GL_TRIANGLE_STRIP

Draws a series of triangles (three-sided polygons) using vertices v0, v1, v2, then v2, v1, v3 (note the order), then v2, v3, v4, and so on. The ordering is to ensure that the triangles are all drawn with the same orientation so that the strip can correctly form part of a surface.



Exercise6. Cleanup: A bit of optimization

The geometry shader does not need 6 spaces of memory, but only 4. Therefore,

```
layout(triangle_strip,max_vertices = 6) out;\n\
```

we can write

```
layout(triangle_strip,max_vertices = 4) out;\n\
```

The vertex shader does not need to give 3 vertices. It can simply be:

```
static const GLchar *
vertex_shader_source[] =
{"#version 330\n\
\n\
void main() {\n\
gl_Position = vec4( 1, -0.25, 0.5, 1.0);\n\
}" };
```

However, the input of the shader needs to keep being triangles (see render modes and inputs in the theory slides)

Exercise7. Make the cube face rotate on itself

```
static const GLchar * geometry_shader_source[] =
{
"#version 330\n\
uniform float time;\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 4) out;\n\
vec4 vertices[4] = vec4[4](vec4(0.25*cos(time) , -0.25,
0.25*sin(time), 1.0),\n\
 vec4(0.25*cos(time)  , 0.25, 0.25*sin(time), 1.0),\n\
 vec4(-0.25*cos(time)  , -0.25, 0.25*sin(time), 1.0),\n\
 vec4(-0.25*cos(time)  , 0.25, 0.25*sin(time), 1.0));\n\
\n\
void main()\n\
{\n\
for(int i= 0; i<4; i++){\n\
gl_Position = gl_in[0].gl_Position  + vertices[i];\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}"
};
```

Exercise7. Make the cube face rotate on itself

Option B:

Give a transformation matrix (a uniform variable) that corresponds to a rotation on the y axis.

hint: check the matrix mvpMat from the cube exercise

Exercise7. Make the cube face rotate on itself

Option B:

```
glm::mat4 myMVP;
void myRenderCode(double currentTime) {
glUseProgram(myRenderProgram);
glUniform1f(glGetUniformLocation(myRenderProgr
am, "time"), (GLfloat)currentTime);
glm::mat4 rot = glm::rotate(glm::mat4(),
0.05f, glm::vec3(0.f, 1.f, 0.f));
myMVP = rot *myMVP;
glUniformMatrix4fv(glGetUniformLocation(myRend
erProgram, "mvpMat"), 1, GL_FALSE,
glm::value_ptr(myMVP));
glDrawArrays(GL_TRIANGLES, 0, 3);
}
```

```
static const GLchar * geom_shader_source[] =
{ "#version 330\n\
uniform float time;\n\
uniform mat4 mvpMat;\n\
layout(triangles) in;\n\
layout(triangle_strip,max_vertices = 6) out;\n\
void main(){\n\
vec4 vertices[4]=vec4[4](vec4( 0.25, -0.25,0,
1.0),\n\
   vec4(0.25, 0.25, 0, 1.0),\n\
vec4( -0.25,  -0.25, 0, 1.0),\n\
vec4(-0.25,0.25,0,1.0));\n\
for(int i = 0; i < 4; i++){\n\
gl_Position = mvpMat*vertices[i];\n\
EmitVertex();\n\
}\n\
\n\
EndPrimitive();\n\
}" };
```

Exercise8. Draw the six faces of a cube. The render code

```
void myRenderCode(double currentTime) {

glUseProgram(myRenderProgram);
glm::mat4 rotation = { cos(currentTime), 0.f, -
sin(currentTime), 0.f,
0.f, 1.f, 0.f, 0.f,
sin(currentTime), 0.f, cos(currentTime), 0.f,
0.f, 0.f, 0.f, 1.f };
glUniformMatrix4fv(glGetUniformLocation(myRende
rProgram, "rotation"), 1, GL_FALSE,
glm::value_ptr(RV::_MVP));
glDrawArrays(GL_TRIANGLES, 0, 3);


}
```

Exercise 8. Draw the six faces of a cube. The geom. shader

```
static const GLchar * geom_shader_source[] = {
"#version 330 \n\
uniform mat4 rotation;\n\
layout(triangles) in;\n\
layout(triangle_strip, max_vertices = 24) out;\n\
void main()\n\
{\n\
const vec4 vertices[4] = vec4[4](vec4(0.25, -0.25, 0.25, 1.0),\n\
vec4(0.25, 0.25, 0.25, 1.0),\n\
vec4(-0.25, -0.25, 0.25, 1.0),\n\
vec4(-0.25, 0.25, 0.25, 1.0));\n\
\n\
//CARA 1\n\
for (int i = 0; i<4; i++)\n\
{\n\
gl_Position = rotation*vertices[i]+gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
\n\
//CARA 2\n\
const vec4 vertices2[4]= vec4[4](vec4(0.25, 0.25, 0.25, 1.0),\n\
vec4(0.25, 0.25, -0.25, 1.0),\n\
vec4(-0.25, 0.25, 0.25, 1.0),\n\
vec4(-0.25, 0.25, -0.25, 1.0));\n\
for (int i = 0; i<4; i++)\n\
{\n\
gl_Position = rotation*vertices2[i]+gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
//CARA 3\n\
const vec4 vertices3[4]= vec4[4](vec4(-0.25, -0.25, 0.25, 1.0),\n\
vec4(-0.25, 0.25, 0.25, 1.0),\n\
vec4(-0.25, -0.25, -0.25, 1.0),\n\
vec4(-0.25, 0.25, -0.25, 1.0));\n\
```

- `for (int i = 0; i<4; i++)\n\`

```
{\n\
gl_Position = rotation*vertices3[i]+gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
//CARA 4\n\
const vec4 vertices4[4]= vec4[4](vec4(-0.25, -0.25, -0.25, 1.0),\n\
vec4(-0.25, 0.25, -0.25, 1.0),\n\
vec4(0.25, -0.25, -0.25, 1.0),\n\
vec4(0.25, 0.25, -0.25, 1.0));\n\
for (int i = 0; i<4; i++)\n\
{\n\
gl_Position = rotation*vertices4[i]+gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
//CARA 5\n\
const vec4 vertices5[4]= vec4[4](vec4(-0.25, -0.25, 0.25, 1.0),\n\
vec4(-0.25, -0.25, -0.25, 1.0),\n\
vec4(0.25, -0.25, 0.25, 1.0),\n\
vec4(0.25, -0.25, -0.25, 1.0));\n\
for (int i = 0; i<4; i++)\n\
{\n\
gl_Position = rotation*vertices5[i]+gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
//CARA 6\n\
const vec4 vertices6[4]= vec4[4](vec4(0.25, -0.25, -0.25, 1.0),\n\
vec4(0.25, 0.25, -0.25, 1.0),\n\
vec4(0.25, -0.25, 0.25, 1.0),\n\
vec4(0.25, 0.25, 0.25, 1.0));\n\
for (int i = 0; i<4; i++)\n\
{\n\
gl_Position = rotation*vertices6[i]+gl_in[0].gl_Position;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
}"
};
```

ESCOLA DE NOVES TECNOLOGIES INTERACTIVES

UNIVERSITAT DE BARCELONA

Exercise8. Make each face of different color

gl_PrimitiveID is implicitly declared as an **output** of the geometry shader. To define its values in the geometry shader it needs to be adjusted before each vertex emitting. For example:

```
//FACE 1\n\
for (int i = 0; i<4; i++){\n\
gl_Position = rotation*vertices[i]+gl_in[0].gl_Position;\n\
gl_PrimitiveID = 0;\n\
EmitVertex();\n\
}\n\
EndPrimitive();\n\
```

gl_PrimitiveID is an **input** implicitly declared in the fragment shader.

```
static const GLchar * fragment_shader_source[] = {
"#version 330\n\
out vec4 color;\n\
void main() {\n\
const vec4 colors[6] = vec4[6](vec4( 0, 1, 0, 1.0),\n\
vec4(0.25, 0.25, 0.5, 1.0),\n\
vec4( 1, 0.25, 0.5, 1.0),\n\
vec4(0.25, 0, 0, 1.0),\n\
vec4( 1, 0, 0, 1.0),\n\
vec4( 0.25, 0.25, 0.5, 1.0));\n\
color = colors[gl_PrimitiveID ];\n\
}" };
```

Exercise9. Make the cube rotate around the center of the cube

In MyRenderCode, add a rotation. Remember to do it on the right side (the default order for objects is scale, rotation, translation).

```
glm::mat4 rot =
glm::rotate(glm::mat4(),
(float)(0.5f*currentTime),
glm::vec3(0.f, 1.f, 0.f));

myMVP = RV::_MVP * rot;

glUniformMatrix4fv(glGetUnifo
rmLocation(myRenderProgram,
“rotation"), 1, GL_FALSE,
glm::value_ptr(myMVP));
```

Notice that despite the variable is called “rotation”, it also includes the matrix corresponding to the camera position and rotation. A more apropriate name would be MvpMat

Exercise10. Use the fragment shader from the previous cube

What do you need to give to the graphics card to be able to use the following shader?

```
const char* cube_fragShader =
"#version 330\n\
in vec4 vert_Normal;\n\
out vec4 out_Color;\n\
uniform mat4 mv_Mat;\n\
uniform vec4 color;\n\
void main() {\n\
out_Color = vec4(color.xyz * dot(vert_Normal,
mv_Mat*vec4(0.0, 1.0, 0.0, 0.0)) + color.xyz *
0.3, 1.0 );\n\
}";
```

Exercises geometry shaders 5 to 10

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 Cube example