ICG 2024

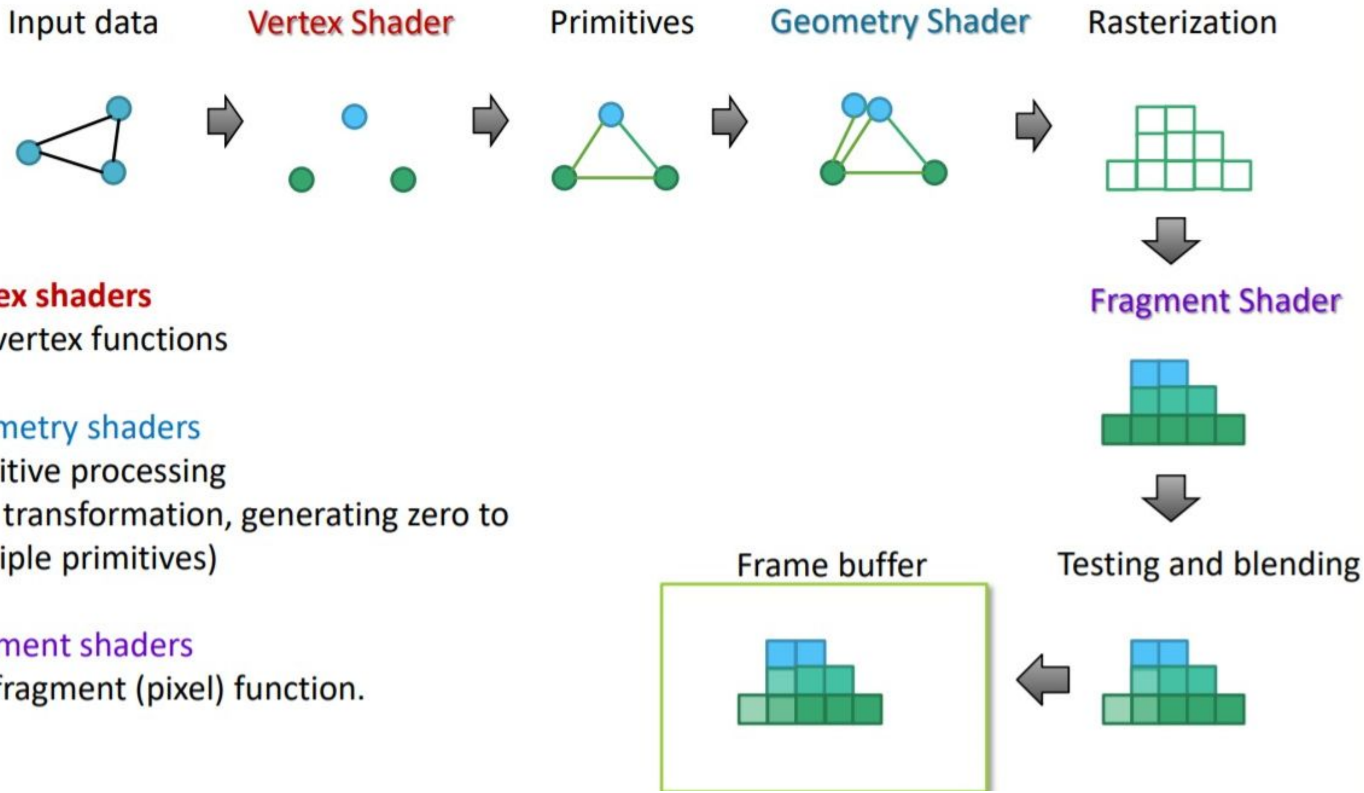
HW4

2024/11/26

Requirements

- 1~3 person per group (Group Registration Form)
- Make a 30 second video.
- Theme: Animation and Geometry shader.
- Must include:
  - At least one object.
  - Geometry shader to create new point, line or polygon.
    (You can change the position or shape of polygon and create additional polygon and so on).
  - You can refer to the examples on the Internet, but you must cite it in your report.
- Report
  - Must include role playing discussion.

Geometry Shader

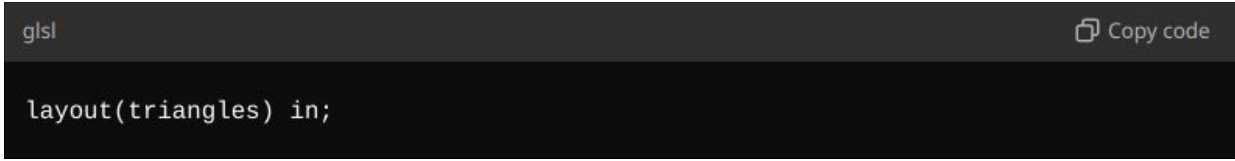


Vertex shaders
per-vertex functions

Geometry shaders
Primitive processing
(E.g. transformation, generating zero to multiple primitives)

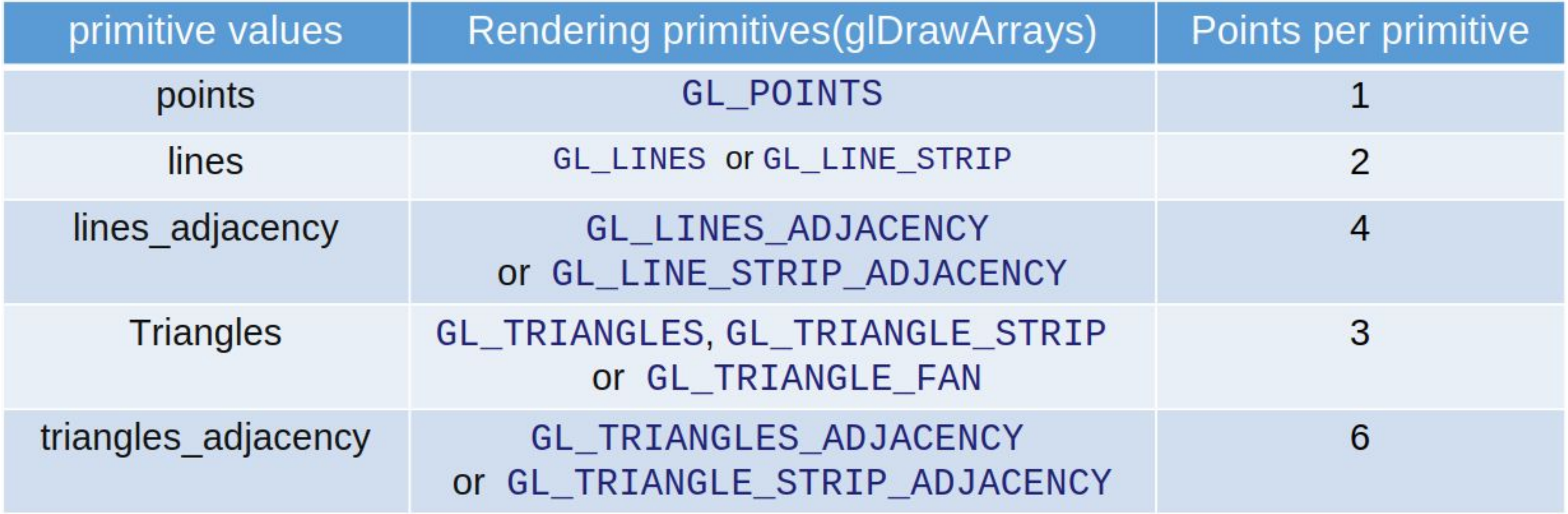
Fragment shaders
per-fragment (pixel) function.

Geometry shader



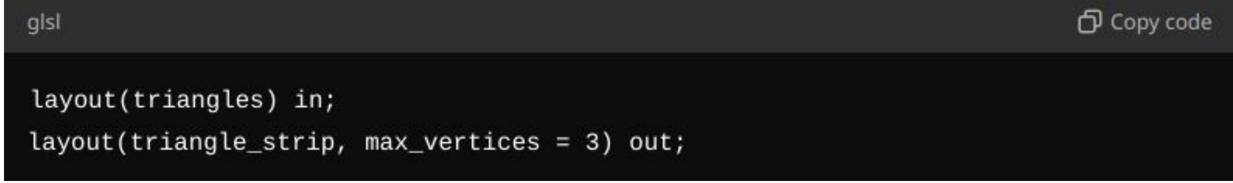
```glsl
layout(triangles) in;
```

- Declare the type of primitive input we're receiving from the vertex shader.
- Method：Declaring a layout specifier in front of the ”in” keyword.



primitive values	Rendering primitives(glDrawArrays)	Points per primitive
points	GL_POINTS	1
lines	GL_LINES or GL_LINE_STRIP	2
lines_adjacency	GL_LINES_ADJACENCY or GL_LINE_STRIP_ADJACENCY	4
Triangles	GL_TRIANGLES, GL_TRIANGLE_STRIP or GL_TRIANGLE_FAN	3
triangles_adjacency	GL_TRIANGLES_ADJACENCY or GL_TRIANGLE_STRIP_ADJACENCY	6

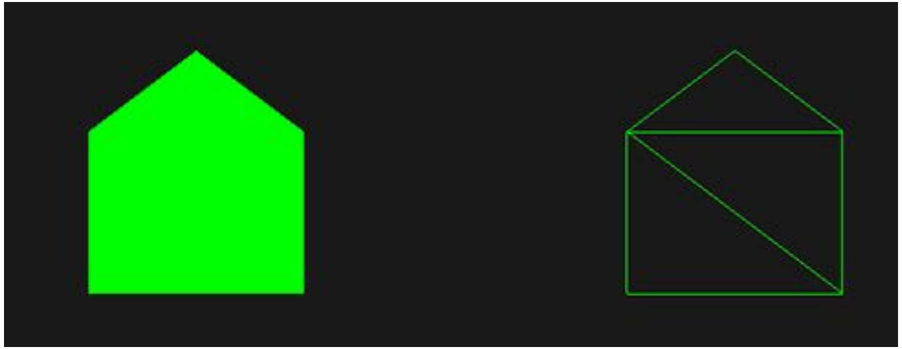
Geometry Shader



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

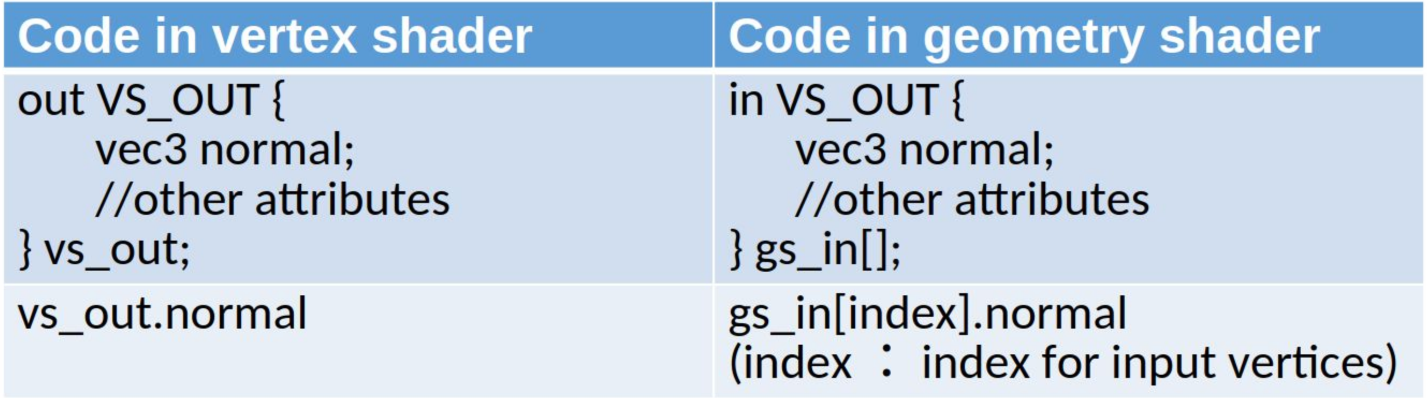
- We also need to specify a primitive type that the geometry shader will output.
- Method：Declaring a layout specifier in front of the ”out” keyword.

- primitive values：points, line_strip, triangle_strip
- max_vertices：If you exceed this number, OpenGL won't draw the extra vertices.



Geometry Shader

- We can update some attributes(color, normal) from vertex shader to the geometry shader.
- Method：Using an interface block.
- Array length：
  - Ex. layout(Triangles) in ; array length is 3.



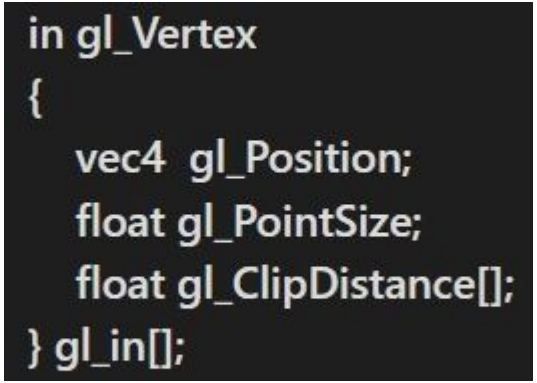
Code in vertex shader	Code in geometry shader

out VS_OUT {
vec3 normal;
//other attributes
} vs_out;
	in VS_OUT {
	vec3 normal;
	//other attributes
	} gs_in[];

vs_out.normal	gs_in[index].normal (index ： index for input vertices)

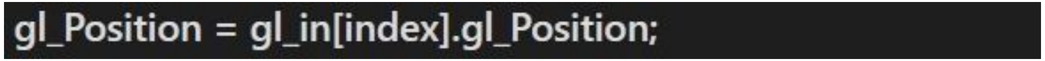
Geometry Shader

- GLSL gives us a built-in variable called gl_in that internally (probably) looks something like this:



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

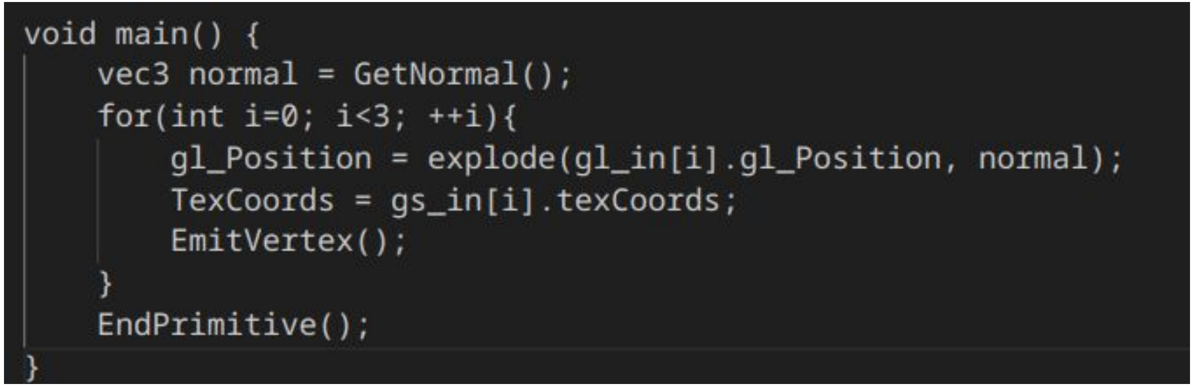
- Thus to access the position of the vertex, we simply can call



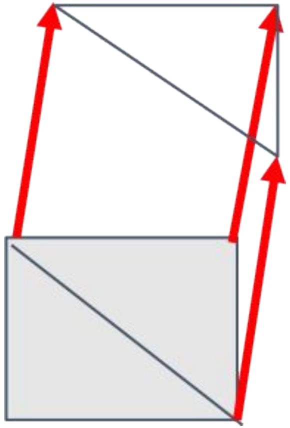
```
gl_Position = gl_in[index].gl_Position;
```

Geometry Shader

- Each time we call EmitVertex(), the vector currently set to gl_Position is added to the output primitive.
- Whenever EndPrimitive() is called, all emitted vertices for this primitive are combined into the specified output render primitive.

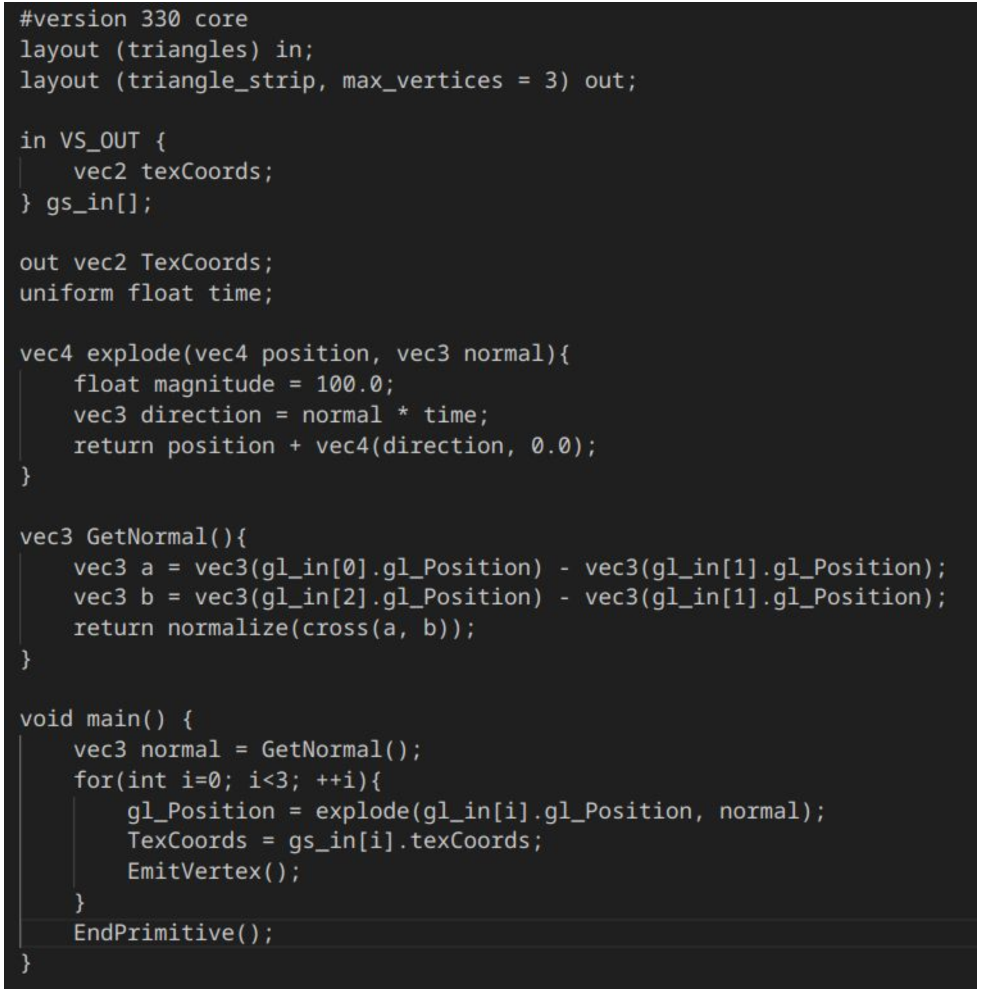


```
void main() {
    vec3 normal = GetNormal();
    for(int i=0; i<3; ++i){
        gl_Position = explode(gl_in[i].gl_Position, normal);
        TexCoords = gs_in[i].texCoords;
        EmitVertex();
    }
    EndPrimitive();
}
```



Geometry Shader

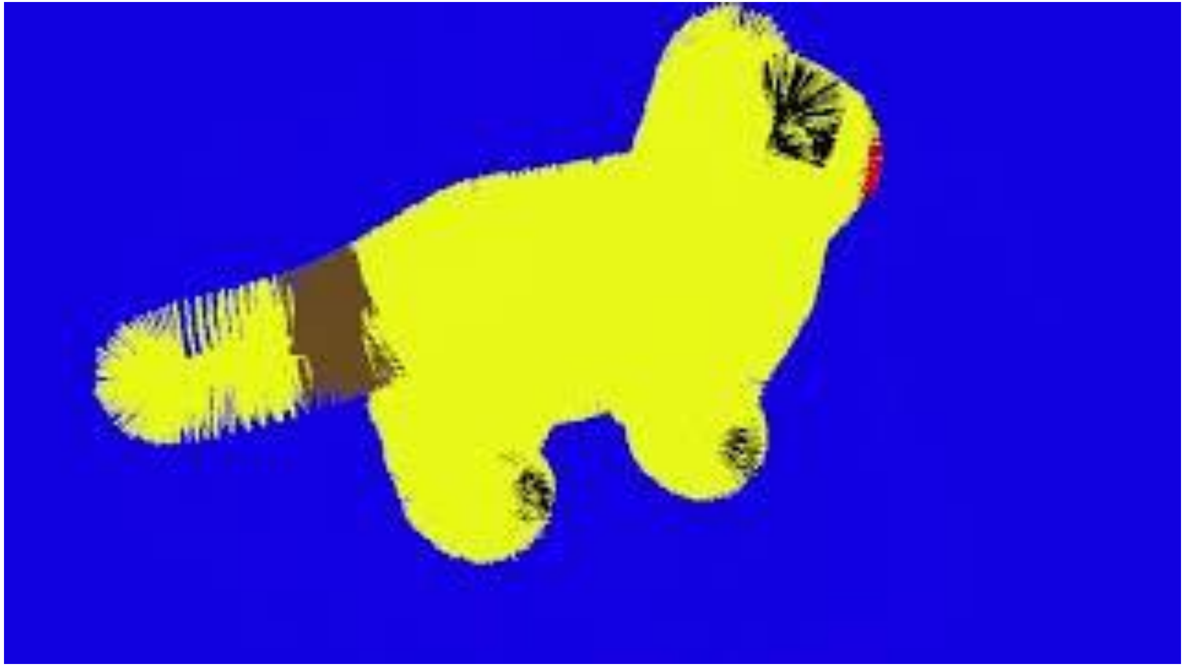
❖ Outline of the code



```
#version 330 core
layout (triangles) in;
layout (triangle_strip, max_vertices = 3) out;

in VS_OUT {
    vec2 texCoords;
} gs_in[];

out vec2 TexCoords;
uniform float time;

vec4 explode(vec4 position, vec3 normal){
    float magnitude = 100.0;
    vec3 direction = normal * time;
    return position + vec4(direction, 0.0);
}

vec3 GetNormal(){
    vec3 a = vec3(gl_in[0].gl_Position) - vec3(gl_in[1].gl_Position);
    vec3 b = vec3(gl_in[2].gl_Position) - vec3(gl_in[1].gl_Position);
    return normalize(cross(a, b));
}

void main() {
    vec3 normal = GetNormal();
    for(int i=0; i<3; ++i){
        gl_Position = explode(gl_in[i].gl_Position, normal);
        TexCoords = gs_in[i].texCoords;
        EmitVertex();
    }
    EndPrimitive();
}
```

Demo

- https://youtu.be/Q1bbpwYyEgk



Role playing

- Discussion of topic and implementation
  - 1 Person must take the role of proposer, proposing the initial idea.
  - 1 Person must take the role of critic, discussing possible flaws in the proposal.
  - 1 Person must take the role of the negotiator (optional if < 2 persons in the group).

❖ Example of role playing

Proposer (A) initially suggested an ambitious narrative where a crystal in a desolate alien wasteland transforms, using geometry shaders, into a majestic, living entity that emits waves of light to revive the barren land. The **Critic (B)** highlighted potential challenges, including the risk of overcomplicating the animation within the 30-second limit and performance issues from overly complex transformations. The **Negotiator (C)** bridged the ideas, proposing a focused approach where the crystal's transformation is the main event, with subtle effects like glowing veins and small plant growth to imply rebirth. This collaborative process ensured a balance between creativity and practicality, resulting in a visually compelling and achievable concept.

Proposer
Okay, picture this: a lone crystal in a desolate, alien wasteland. Suddenly, it starts growing—geometry shaders make it sprout fractals, spines, and glowing structures. But here's the twist—it's not just growing; it's alive. Over 30 seconds, the crystal becomes a massive, otherworldly creature. Its final transformation reveals it as the key to reviving the barren land, sending waves of light that spark new life around it. A mini story of creation and rebirth!

Critic
Cool idea, but that's a *lot* for 30 seconds. All those transformations might feel rushed, and if the creature is too complex, we'll end up battling performance issues. Plus, animating a whole 'reviving the world' scene could turn into a time sink. How do we balance story with practicality?

Negotiator
Good points. Let's focus on the crystal's transformation as the main event. It starts as a simple structure, grows into something majestic, and at the end, its light spreads subtly—just enough to hint at a rebirth without showing a full landscape change. That way, we tell the story while keeping it achievable.

Report Format

- Title
  - Include the story name and your groupID and the members of your team.
- Introduction
  - Include the outline of your story and what you will be doing.
- Implementation details
  - Highlight the techniques you used and briefly describe the implementation.
- Discussion
  - Your discussion from role playing.
  - Anything you would like to discuss.

Report Format (cont)

- Work assignment
  - Which person plays which role, … e.t.c

- References
  - If you used online resource you must cite in report.
  - You don’t need to cite the TAs code from previous homeworks :)

- Results
  - Record a 30 second demo video.
  - You can use OBS studio to record your video.
    Download OBS studio here: https://obsproject.com/download
  - Upload to youtube and paste the link here.

- No more then 5 pages excluding cover page.

Score

1. Basic requirements (40%)
2. Technique (20%)
3. Creativity (15%)
    a. Base on the votes from peers.
4. Peer review (5%)
    a. Vote for five videos.
5. Report (20%)

Submission

- Deadline: 2024/12/29 23:59:59, no late submission !
- Zip your report, source code and upload the zip file to E3.
- One submission is required for each group. If there are multiple submissions the last submission would be used.
- Zip name : GroupID_HW4.zip (-5% for incorrect file name/format)
  - e.g.

```
Group1_HW4.zip
    ├─ src/
    │    ├─ all the materials …
    │    ├─ all the source code …
    │    ├─ any additional files
    │    └─ README.md (must include and explain how to run your code !! )
    └─ report.pdf
```

Reference

- https://learnopengl.com/Advanced-OpenGL/Geometry-Shader
- https://www.khronos.org/opengl/wiki/Geometry_Shader