How to Use Shader

Vertex & Geometry & Fragment

OpenGL

- glDrawArrays(GL_TRIANGLES, first, count);
- glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, position)));
- glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, sizeof(VertexAttribute), (void*)(offsetof(VertexAttribute, texcoord)));

Vertex

- layout(location = 0) in vec3 position;
- layout(location = 1) in vec3 texcoord;
- out vec4 geom_texcoord

Geometry

- layout(triangles) in;
- layout(triangle_strip, max_vertices=3) out; //Output Primitive
- in vec2 geom_texcoord[3]
- out vec4 frag_texcoord

Fragment

- in vec4 frag_texcoord;
- out vec4 outColor;

Special Variable

- Vertex Shader
  - gl_Position: Position in view space
- Geometry Shader
  - gl_Position: Position in view space
  - gl_in[]: All input vertex of the primitive
    - ex: gl_in.length(), gl_in[0].gl_Position, gl_in[0].your_own_variable
  - EmitVertex(): Output the vertex after assign attributes of the vertex
  - EndPrimitive(): Output new primitive including the vertices emitted before
- Fragment Shader
  - out vec4 outColor: Final output Color

HW3 Hint

OpenGL

- VertexAttribute must include position, normal, texcoord
- New a VertexAttribute array with all sphere vertices and pass to vertex shader
- Pass Projection matrix, ModelView matrix, textures and triggers by Uniform

Vertex
(Per Vertex)

- Pass normal, texcoord to geometry shader
- Set gl_Position (or you can set in geometry shader)

Geometry
(Per Primitive)

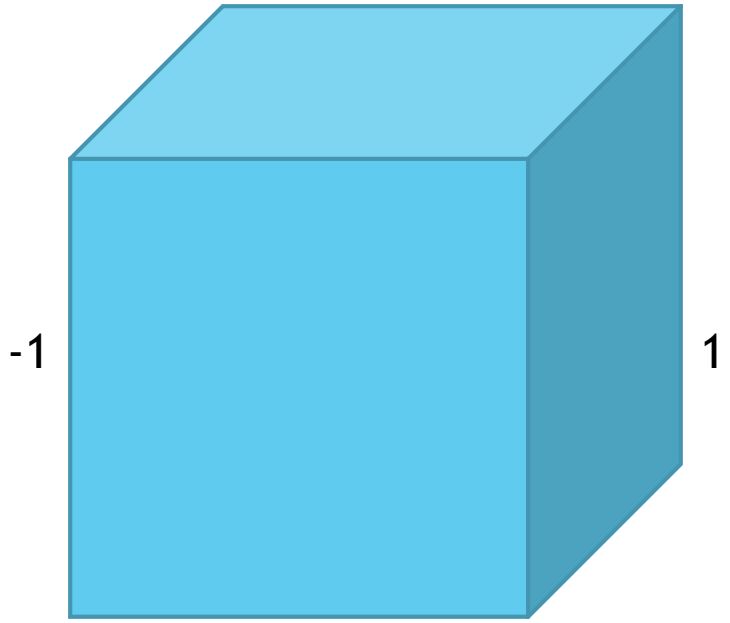
- Get all vertex attributes of the polygon and compute tangent and bitangent
- Assign normal, tangent, bitangent, texcoord on each vertice and EmitVertex()
- Tangent & bitangent can be computed in either geometry shader or OpenGL
- Emit all vertices and EndPrimitive() to pass primitive to fragment shader

Fragment
(Per Pixel)

- All data in fragment shader would be rasterized
- Get texture map, normal map, specular map
- Get normal map color, and compute new normal by normal, tangent, bitangent
- Apply Phong shading by using new normal, viewing direction, light direction

View Space

- ViewSpace = Projection * ModelView * WorldSpace



Location & Direction

- Location
  - $P = [x \quad y \quad z \quad 1]^T$
  - $P' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{bmatrix}$
- Direction
  - $D = [x \quad y \quad z \quad 0]^T$
  - $D' = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}$