**Geometry shader:**

#version 330 core

layout(triangles\_adjacency) in;

layout(triangle\_strip, max\_vertices = 12) out;

uniform float HalfWidth;

uniform float OverhangLength;

out float gDist;

out vec3 gSpine;

in vec3 FragPos[];

in vec3 Normal[];

in vec2 TexCoords[];

out vec3 geo\_FragPos;

out vec3 geo\_Normal;

out vec2 geo\_TexCoords;

bool IsFront(vec3 A, vec3 B, vec3 C) {

float area = (A.x \* B.y - B.x \* A.y) + (B.x \* C.y - C.x \* B.y) + (C.x \* A.y - A.x \* C.y);

return area > 0;

}

// article

void EmitEdge(vec3 P0, vec3 P1) {

vec3 E = OverhangLength \* vec3(P1.xy - P0.xy, 0);

vec2 V = normalize(E.xy);

vec3 N = vec3(-V.y, V.x, 0) \* HalfWidth;

gSpine = (P0 + 1.0) \* 0.5;

gDist = +HalfWidth;

gl\_Position = vec4(P0 - N - E, 1);

EmitVertex();

gDist = -HalfWidth;

gl\_Position = vec4(P0 + N - E, 1);

EmitVertex();

gSpine = (P1 + 1.0) \* 0.5;

gDist = +HalfWidth;

gl\_Position = vec4(P1 - N + E, 1);

EmitVertex();

gDist = -HalfWidth;

gl\_Position = vec4(P1 + N + E, 1);

EmitVertex();

EndPrimitive();

}

// zip file

//void EmitEdge(vec3 P0, vec3 P1) {

//vec3 E = OverhangLength \* vec3(P1.xy - P0.xy, 0);

//vec2 V = normalize(E.xy);

//vec3 N = vec3(-V.y, V.x, 0) \* HalfWidth;

//vec3 S = -N;

//float D = HalfWidth;

//gSpine = P0;

//gl\_Position = vec4(P0 + S - E, 1);

//gDist = +D;

//EmitVertex();

//gl\_Position = vec4(P0 + N - E, 1);

//gDist = -D;

//EmitVertex();

//gSpine = P1;

//gl\_Position = vec4(P1 + S + E, 1);

//gDist = +D;

//EmitVertex();

//gl\_Position = vec4(P1 + N + E, 1);

//gDist = -D;

//EmitVertex();

//EndPrimitive();

//}

void main() {

vec3 v0 = gl\_in[0].gl\_Position.xyz / gl\_in[0].gl\_Position.w;

vec3 v1 = gl\_in[1].gl\_Position.xyz / gl\_in[1].gl\_Position.w;

vec3 v2 = gl\_in[2].gl\_Position.xyz / gl\_in[2].gl\_Position.w;

vec3 v3 = gl\_in[3].gl\_Position.xyz / gl\_in[3].gl\_Position.w;

vec3 v4 = gl\_in[4].gl\_Position.xyz / gl\_in[4].gl\_Position.w;

vec3 v5 = gl\_in[5].gl\_Position.xyz / gl\_in[5].gl\_Position.w;

if (IsFront(v0, v2, v4)) {

if (!IsFront(v0, v1, v2))

EmitEdge(v0, v2);

if (!IsFront(v2, v3, v4))

EmitEdge(v2, v4);

if (!IsFront(v0, v4, v5))

EmitEdge(v4, v0);

}

for(int i = 0; i < 36; i++) {

geo\_FragPos = FragPos[i];

//EmitVertex();

geo\_Normal = Normal[i];

//EmitVertex();

geo\_TexCoords = TexCoords[i];

//EmitVertex();

}

}

**Fragment shader:**

#version 330 core

in vec3 geo\_Normal;

in vec3 geo\_FragPos;

in vec4 lightSpacePos;

uniform vec3 lightPos;

uniform vec3 camPos;

uniform vec3 lightColor;

out vec4 FragColor;

uniform vec3 objectColor;

uniform float alpha;

uniform sampler2D texture\_diffuse1;

in vec2 geo\_TexCoords;

uniform bool use\_textures;

vec3 computeAmbientComponent() {

float ambientStrength = 0.3;

return ambientStrength \* lightColor;

}

vec3 getLightDir() {

return normalize((camPos + lightPos) - geo\_FragPos);

}

vec3 computeDiffuseComponent() {

float diffuseStrength = 0.8;

vec3 norm = normalize(geo\_Normal);

vec3 lightDir = getLightDir();

float diff = max(dot(norm, lightDir), 0.0);

return diffuseStrength \* diff \* lightColor;

}

vec3 computeSpecularComponent() {

float specularStrength = 0.5;

vec3 viewDir = normalize(camPos - geo\_FragPos);

vec3 lightDir = getLightDir();

vec3 norm = normalize(geo\_Normal);

vec3 reflectDir = reflect(-lightDir, norm);

float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);

return specularStrength \* spec \* lightColor;

}

vec3 computeBasicShading() {

vec3 ambient = computeAmbientComponent();

vec3 diffuse = computeDiffuseComponent();

vec3 specular = computeSpecularComponent();

return ambient + diffuse + specular;

}

void main() {

if (use\_textures == false) {

FragColor = vec4(computeBasicShading() \* objectColor, alpha);

//FragColor = vec4(0.0, 0.0, 0.0, 1.0);

}

else {

FragColor = vec4(computeBasicShading(), alpha) \* texture(texture\_diffuse1, geo\_TexCoords);

}

}