Advanced Lighting // Point shadow

Omnidirectional Shadow maps

-2D텍스처 대신 큐브 맵 텍스처를 깊이 맵으로 바인딩

-light projection의 far plane 변수를 쉐이더에 전달

-대략적인 렌더링 코드

|  |
| --- |
| glViewport(0, 0, SCR\_WIDTH, SCR\_HEIGHT);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  shader.use();  // ... send uniforms to shader (including light's far\_plane value)  glActiveTexture(GL\_TEXTURE0);  glBindTexture(GL\_TEXTURE\_CUBE\_MAP, depthCubemap);  // ... bind other textures  RenderScene(); |

-vertex shader에서는 방향 벡터를 사용해서 깊이 값을 샘플링 하기 때문에 light space에서 fragment의 위치가 필요가 없음

-vertex shader

|  |
| --- |
| #version 460 core  layout (location = 0) in vec3 aPos;  layout (location = 1) in vec3 aNormal;  layout (location = 2) in vec2 aTexCoords;  out vec2 TexCoords;  out VS\_OUT {  vec3 FragPos;  vec3 Normal;  vec2 TexCoords;  } vs\_out;  uniform mat4 projection;  uniform mat4 view;  uniform mat4 model;  void main()  {  vs\_out.FragPos = vec3(model \* vec4(aPos, 1.0));  vs\_out.Normal = transpose(inverse(mat3(model))) \* aNormal;  vs\_out.TexCoords = aTexCoords;  gl\_Position = projection \* view \* model \* vec4(aPos, 1.0);  } |

-fragment shader // 이전 shadow map의 shader와 동일

|  |
| --- |
| #version 460 core  out vec4 FragColor;  in VS\_OUT {  vec3 FragPos;  vec3 Normal;  vec2 TexCoords;  } fs\_in;  uniform sampler2D diffuseTexture;  uniform samplerCube depthMap;  uniform vec3 lightPos;  uniform vec3 viewPos;  uniform float far\_plane;  float ShadowCalculation(vec3 fragPos)  {  [...]  }  void main()  {  vec3 color = texture(diffuseTexture, fs\_in.TexCoords).rgb;  vec3 normal = normalize(fs\_in.Normal);  vec3 lightColor = vec3(0.3);  // ambient  vec3 ambient = 0.3 \* color;  // diffuse  vec3 lightDir = normalize(lightPos - fs\_in.FragPos);  float diff = max(dot(lightDir, normal), 0.0);  vec3 diffuse = diff \* lightColor;  // specular  vec3 viewDir = normalize(viewPos - fs\_in.FragPos);  vec3 reflectDir = reflect(-lightDir, normal);  float spec = 0.0;  vec3 halfwayDir = normalize(lightDir + viewDir);  spec = pow(max(dot(normal, halfwayDir), 0.0), 64.0);  vec3 specular = spec \* lightColor;  // calculate shadow  float shadow = ShadowCalculation(fs\_in.FragPos);  vec3 lighting = (ambient + (1.0 - shadow) \* (diffuse + specular)) \* color;    FragColor = vec4(lighting, 1.0);  } |

-약간의 차이점은 samplerCube가 있으며, ShadowCalculation 함수에서는 light space fragment Position 대신 world space fragment Position을 받음

-fragment가 그림자일 때는 1.0, 아닐 때는 0.0을 반환함

-shadowCalculation의 상세한 내용

|  |
| --- |
| float ShadowCalculation(vec3 fragPos)  {  vec3 fragToLight = fragPos - lightPos;  float closestDepth = texture(depthMap, fragToLight).r;  } |

-fragment와 light의 위치 사이의 차이 벡터를 구해서 그 벡터를 방향벡터로 큐브맵을 샘플링, 정규화가 필요가 없음, 최종 closetDepth의 값이 광원과 가장 가까운 정규화된 깊이 값임

|  |
| --- |
| closestDepth \*= far\_plane; |

-closestDepth가 0~1 범위에 있음으로 far\_plane을 곱해서 0~far\_plane의 범위로 변환

|  |
| --- |
| float currentDepth = length(fragToLight); |

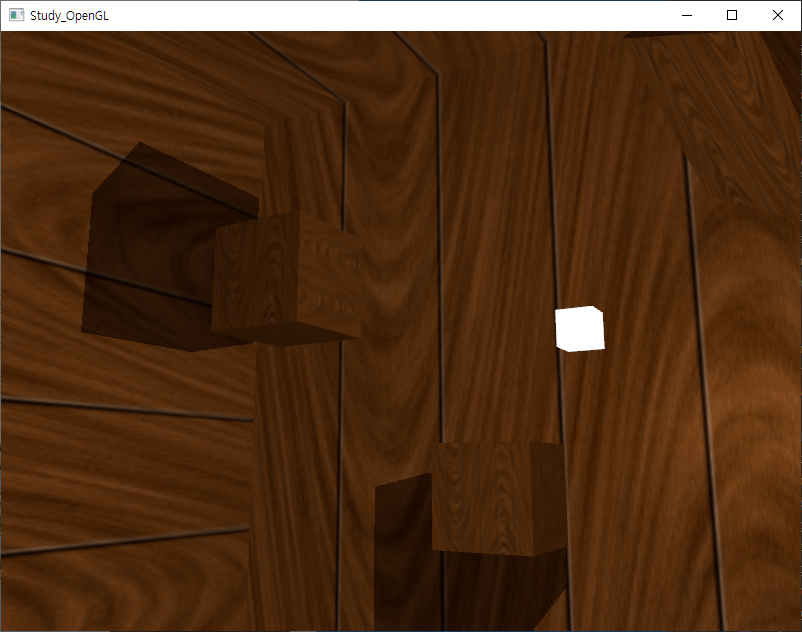
-fragment와 light 사이의 깊이 값을 검색함, closestDepth와 같은 범위의 깊이 값을 반환함

|  |
| --- |
| float bias = 0.05;  float shadow = currentDepth - bias > closestDepth ? 1.0 : 0.0; |

-bias 추가

-최종 ShadowCalculation 함수

|  |
| --- |
| float ShadowCalculation(vec3 fragPos)  {  // get vector between fragment position and light position  vec3 fragToLight = fragPos - lightPos;  // use the light to fragment vector to sample from the depth map  float closestDepth = texture(depthMap, fragToLight).r;  // it is currently in linear range between [0,1]. Re-transform back to original value  closestDepth \*= far\_plane;  // now get current linear depth as the length between the fragment and light position  float currentDepth = length(fragToLight);  // now test for shadows  float bias = 0.05;  float shadow = currentDepth - bias > closestDepth ? 1.0 : 0.0;  return shadow;  } |

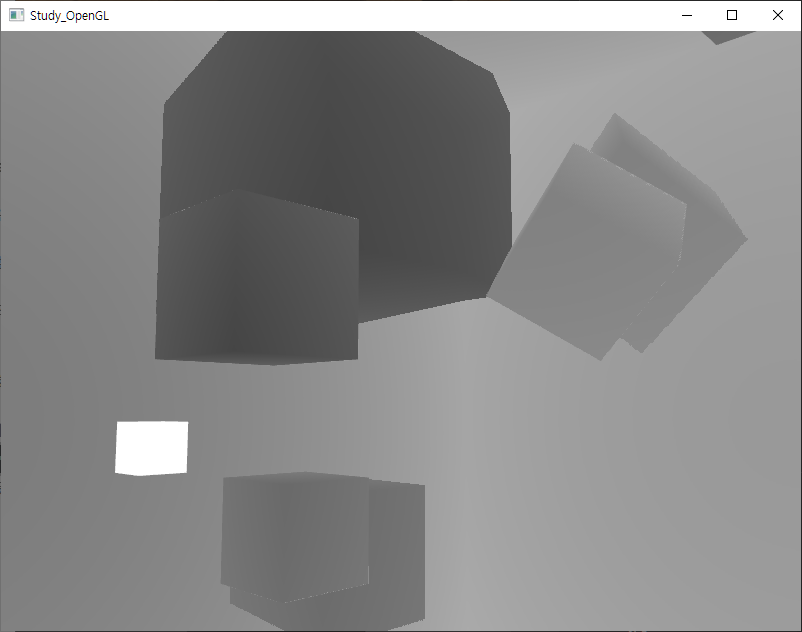
-보기 힘들어서 텍스처 변경

Visualizing cubemap depth buffer

-깊이 맵을 흑백으로 시각화하기

-ShadowCalculation 함수를 살짝 수정

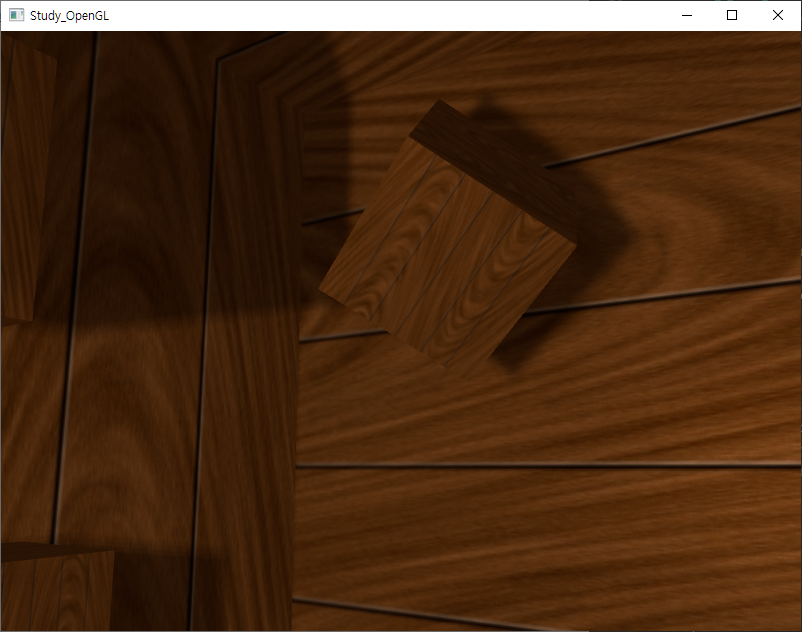
|  |
| --- |
| FragColor = vec4(vec3(closestDepth / far\_plane), 1.0); |



PCF

-계단모양의 그림자를 해결하기 위해 PCF넣기

|  |
| --- |
| float shadow = 0.0;  float bias = 0.05;  float samples = 4.0;  float offset = 0.1;  for(float x = -offset; x < offset; x += offset / (samples \* 0.5))  {  for(float y = -offset; y < offset; y += offset / (samples \* 0.5))  {  for(float z = -offset; z < offset; z += offset / (samples \* 0.5))  {  float closestDepth = texture(depthMap, fragToLight + vec3(x, y, z)).r;  closestDepth \*= far\_plane; // undo mapping [0;1]  if(currentDepth - bias > closestDepth)  shadow += 1.0;  }  }  }  shadow /= (samples \* samples \* samples); |

-샘플을 4.0으로 하면 fragment들이 64개의 샘플을 가짐으로 너무 많음

-샘플들의 대부분의 원래 방향 벡터에 가깝게 샘플링 한다는 점에서 중복됨

-따라서 샘플 방향 벡터의 수직 방향으로 샘플링 하는 것이 합리적일 수 있음

-대략적인 오프셋 방향 배열을 생성

|  |
| --- |
| vec3 sampleOffsetDirections[20] = vec3[]  (  vec3( 1, 1, 1), vec3( 1, -1, 1), vec3(-1, -1, 1), vec3(-1, 1, 1),  vec3( 1, 1, -1), vec3( 1, -1, -1), vec3(-1, -1, -1), vec3(-1, 1, -1),  vec3( 1, 1, 0), vec3( 1, -1, 0), vec3(-1, -1, 0), vec3(-1, 1, 0),  vec3( 1, 0, 1), vec3(-1, 0, 1), vec3( 1, 0, -1), vec3(-1, 0, -1),  vec3( 0, 1, 1), vec3( 0, -1, 1), vec3( 0, -1, -1), vec3( 0, 1, -1)  ); |

-PCF 알고리즘으로 sampleOffsetDirections에서 일정량의 샘플을 가져와 큐브 맵을 샘플링

-PCF에서 훨씬 적은 샘플이 필요함

|  |
| --- |
| float shadow = 0.0;  float bias = 0.15;  int samples = 20;  float viewDistance = length(viewPos - fragPos);  float diskRadius = 0.05;  for(int i = 0; i < samples; ++i)  {  float closestDepth = texture(depthMap, fragToLight + sampleOffsetDirections[i] \* diskRadius).r;  closestDepth \*= far\_plane; // undo mapping [0;1]  if(currentDepth - bias > closestDepth)  shadow += 1.0;  }  shadow /= float(samples); |

-또한, 사용자의 거리에 따라 오프셋 반경을 변경해 멀리있을 때 부드럽고, 가까이 있을 때는 날카롭게 만들 수 있음

