PBR / IBL Diffuse irradiance

Diffuse irradiance

-IBL : Image Based Lighting

-저번 장에서 사용한 direct analytical light가 아닌 주변 환경을 하나의 큰 광원이라고 취급하여 물체를 비추는 기술

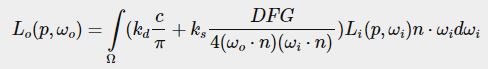
-일반적으로 큐브 맵 환경 맵을 조작해서 조명 방정식에 바로 사용할 수 있음

-큐브 맵 텍셀을 광원이라고 생각함, 환경의 전체적인 조명과 일반적인 느낌을 가질 수 있음

-IBL algorithms이 일부 환경의 조명을 캡쳐할 때, 입력은 더 정교한 ambient lighting으로 간주되고, 글로벌 조명의 대략적인 근사치도 고려됨

-주변 환경을 고려하면 IBL이 PBR보다 더 정확하게 보임

-반사율 방정식

-이전 장에서는 들어오는 빛의 방향 ωi를 미리 알고 있기 때문에 쉽게 계산하였음, 그러나 주변 환경으로부터 들어오는 모든 빛의 방향에 따라 약간의 광도가 생김, 적분이 어려워짐

-적분을 해결하는데 필요한 두 가지 요구사항

-1. 임의의 방향 벡터가 주어진 경우, 장면의 밝기를 검색 할 수 있는 방법

-2. 적분이 빠르고 실시간으로 이루어져야함

-1번의 경우, 환경을 큐브맵으로 만들 것이기 때문에, 큐브 맵의 모든 텍셀을 단일 광원으로 시각화해 큐브 맵을 임의의 ωi로 샘플링 함으로 그 방향에서 장면의 광도를 검색하면 됨

-방향이 주어지면, 밝기를 얻는 방법은 쉬움

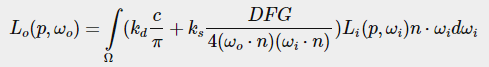
|  |
| --- |
| vec3 radiance = texture(\_cubemapEnvironment, w\_i).rgb; |

-그래도 적분을 풀기 위해선 한 방향이 아닌 모든 방향에서 샘플링해야 하기 때문에,

fragment당 쉐이더 호출이 너무 많이 일어남

-이것을 해결하기 위해선 사전에 계산이 이루어져야 쉐이더 호출이 적어짐

-이를 위한 반사 방정식에 대한 공부가 필요

-적분을 두개로 나눠서 diffuse와 specular에 개별적으로 계산하도록 변경



-diffuse의 적분 부분(왼쪽 항)을 살펴보면 diffuse lambert가 상수항인 것을 확인할 수 있음

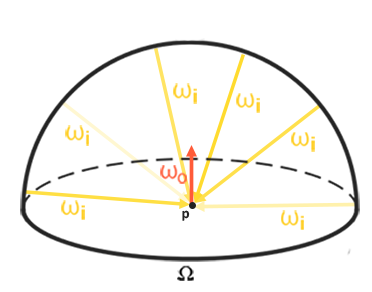
(색상 c, 반사율 kd, 그리고 π)

-점p가 중심에 있다고 가정하면 ωi에만 의존하는 적분이 생성됨

-이것을 사용하면 diffuse 적분의 결과를 미리 계산이 가능함, 각 텍셀(또는 샘플 방향)에 저장하는 새로운 큐브 맵을 계산해서 만들 수 있음

-환경 맵을 convolute하기 위해서 반구Ω에 있는 수많은 방향 ωi를 이산 적으로 샘플링하고, 광도를 평균화해서 ωo 샘플 방향에 대한 적분을 풀었음

-우리가 샘플 방향ωi를 만드는 반구는 ωo 샘플 방향이 출력 방향임

-이 미리 계산되어진 큐브 맵은 방향 ωo를 따라서 정렬되어진 표면에 부딪히는 모든 diffuse light의 사전 계산된 합으로 생각할 수 있음

-이 큐브 맵은 어떤 방향의 ωo의 irradiance를 직접 샘플링 할 수 있기 때문에

irradiance map이라고도 불림

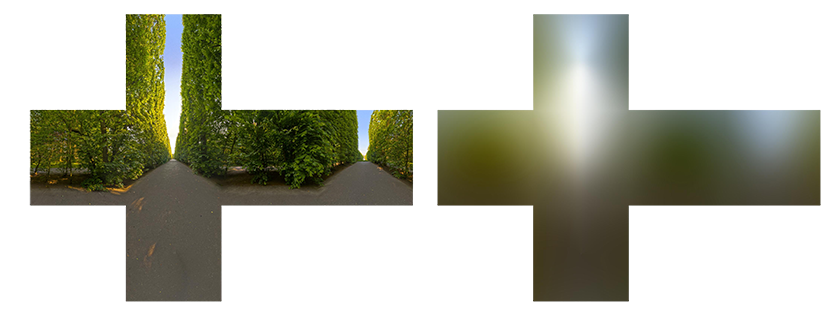
-radiance 방정식은 점p의 위치에 의존함, 이것은 단일 완경 맵에서 오는 모든 diffuse indirect light이 현실감을 무너트릴 수 있다는 것을 위미함

-Render 엔진은 이것을 반사 probes를 각 반사 probes를 주변 환경의 자체 irradiance map을 계산하는 장면 전체에 배치해서 문제를 해결함

-이 방식은 점p에서의 irradiance와 radiance는 가장 가까운 반사 probes 사이에 보간된 값이 됨

-이 튜토리얼에서는 항상 환경 맵의 중심에서 샘플을 추출할 것임

-큐브 맵 환경 맵이랑 irradiance(모든 방향에 대한 장면의 광도를 평균화) 맵의 예시임



PBR and HDR

-PBR에서 HDR은 매우 중요한 역할을 한다는 것을 배웠었음

-PBR은 유입되는 빛의 값을 물리적으로 동일한 값으로 일치시키는 것이 좋기 때문

-각 조명의 빛의 강도에 대해서 어떤 물리적인 이유를 사용하는지에 관계없이, 간단한 전구나 태양의 차이는 중요함, HDR렌더릴ㅇ에서는 각 조명의 상대 강도를 올바르게 지정할 수 있음

-따라서 PBR, HDR은 밀접한 관련이 있음

-IBL은 관련이 있는가?

-IBL의 경우 환경의 간접 조명 강도를 환경 큐브 맵의 색상에 기초해서 조명의 HDR 환경맵에 저장하는 방법이 필요함

-이전에 큐브맵을 공부했을 땐, LDR을 그래도 사용했었음, 이것은 시각적으로 문제는 없으나, 물리적인 입력 변수로 가져간다면 효과가 없을 것임

The radiance HDR file format

-Radiance 파일은 모든 6면이 부동 소수점 데이터로 가득 찬 전체 큐브 맵을 저장해서 누구나 0~1 범위 밖의 색을 지정해서 조명에 올바른 색상 강도를 제공할 수 있도록 함

-파일 형식은 채널당 32비트 값을 사용하지 않지만, 알파 채널을 지수로 사용하는 채널당 8비트를 사용함(정확도는 손실이 있음)

-parsing 프로그램일 경우에는 각 색상을 부동 소수점으로 다시 변환해야함

-HDR 환경 맵 <http://www.hdrlabs.com/sibl/archive.html>

-이미지가 왜곡되어있으며, 6면의 큐브맵을 표시하고 있지 않음

-이 환경맵은 구면에서 평면으로 투영되어진 모습이고, equirectangular map이라고 함

-대부분의 시각적 해상도가 수평 시아 방향으로 저장되어짐, 그러나 수직 방향(위아래)방향에는 더 작은 양의 시작적 해상도가 보존되기 때문에 주의해야함

HDR and stb\_image.h

-HDR이미지를 불러오는 방법은 stb\_image.h 해더파일에 있음

-HDR 이미지 로드하는 법

|  |
| --- |
| #include "stb\_image.h"  [...]  stbi\_set\_flip\_vertically\_on\_load(true);  int width, height, nrComponents;  float \*data = stbi\_loadf("newport\_loft.hdr", &width, &height, &nrComponents, 0);  unsigned int hdrTexture;  if (data)  {  glGenTextures(1, &hdrTexture);  glBindTexture(GL\_TEXTURE\_2D, hdrTexture);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGB16F, width, height, 0, GL\_RGB,  GL\_FLOAT, data);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S,  GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T,  GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);  stbi\_image\_free(data);  }  else  {  std::cout << "Failed to load HDR image." << std::endl;  } |

-기본적으로 부동 소수점 값에 매핑함 (채널당 32비트, 색상 당 3채널)

From Equirectangular to Cubemap

-Equirectangular 등장방형도법 : 구 -> 평면 // ex)세계지도

-lookup을 위해서 바로 등장방형도법을 직접 사용할 수도 있지만, direct 큐브 맵 샘플보 상대적으로 비용이 많이 들어감

-Equirectangular 이미지를 먼저 큐브맵으로 변환, 이 과정에서 3D 환경 맵인 것처럼 대각선 맵을 샘플링하는 법을 사용할 것임, 다른 방법도 있음

-equirectangular 이미지를 큐브 맵으로 변환하려면 큐브를 렌더링, 내부에서 모든 큐브 면에 대한 equirectangular맵을 투영하고 각 6면의 이미지를 큐브 맵으로 가져와야함

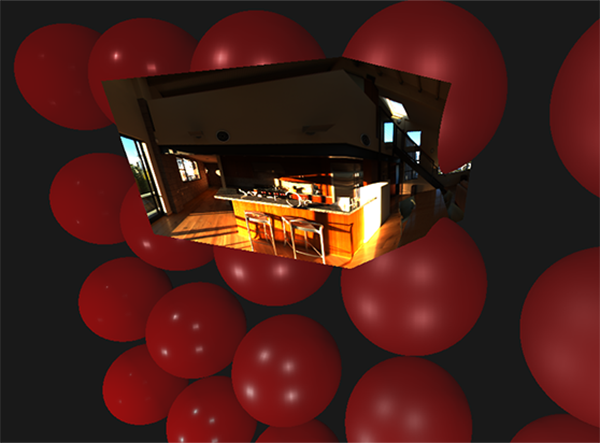
-이 큐브의 vertex shader은 큐브를 그대로 렌더링하고, 위치를 3D 샘플 벡터로 fragment shader로 전달함

|  |
| --- |
| #version 460 core  layout (location = 0) in vec3 aPos;  out vec3 localPos;  uniform mat4 projection;  uniform mat4 view;  void main()  {  localPos = aPos;  gl\_Position = projection \* view \* vec4(localPos, 1.0);  } |

-fragment shader은 각면에 equirectangular을 깔끔하게 접은 것 처럼 큐브의 각 부분을 채워넣음

|  |
| --- |
| #version 460 core  out vec4 FragColor;  in vec3 localPos;  uniform sampler2D equirectangularMap;  const vec2 invAtan = vec2(0.1591, 0.3183);  vec2 SampleSphericalMap(vec3 v)  {  vec2 uv = vec2(atan(v.z, v.x), asin(v.y));  uv \*= invAtan;  uv += 0.5;  return uv;  }  void main()  {  vec2 uv = SampleSphericalMap(normalize(localPos));  vec3 color = texture(equirectangularMap, uv).rgb;    FragColor = vec4(color, 1.0);  } |

-장면의 중앙에 큐브를 렌더링하면 보이는 모습

-equirectangular이미지를 큐브로 매핑하는데 성공한 모습이 모임

-이것을 큐브 맵 텍스처로 변환할려면 각 큐브의 개별면을 보면서 6번을 렌더링해야함

|  |
| --- |
| unsigned int captureFBO, captureRBO;  glGenFramebuffers(1, &captureFBO);  glGenRenderbuffers(1, &captureRBO);  glBindFramebuffer(GL\_FRAMEBUFFER, captureFBO);  glBindRenderbuffer(GL\_RENDERBUFFER, captureRBO);  glRenderbufferStorage(GL\_RENDERBUFFER, GL\_DEPTH\_COMPONENT24, 512, 512);  glFramebufferRenderbuffer(GL\_FRAMEBUFFER, GL\_DEPTH\_ATTACHMENT,  GL\_RENDERBUFFER, captureRBO); |

-6개의 면에 각각 메모리를 할당

|  |
| --- |
| unsigned int envCubemap;  glGenTextures(1, &envCubemap);  glBindTexture(GL\_TEXTURE\_CUBE\_MAP, envCubemap);  for (unsigned int i = 0; i < 6; ++i)  {  // note that we store each face with 16 bit floating point values  glTexImage2D(GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, 0, GL\_RGB16F,  512, 512, 0, GL\_RGB, GL\_FLOAT, nullptr);  }  glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_S,  GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_T,  GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_WRAP\_R,  GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MIN\_FILTER,  GL\_LINEAR);  glTexParameteri(GL\_TEXTURE\_CUBE\_MAP, GL\_TEXTURE\_MAG\_FILTER,  GL\_LINEAR); |

-이후 투영된 2D 텍스처를 큐브 맵 면에 캡쳐

-부동 소수점 프레임 버퍼에 렌더링

|  |
| --- |
| glm::mat4 captureProjection = glm::perspective(glm::radians(90.0f), 1.0f, 0.1f, 10.0f);  glm::mat4 captureViews[] =  {  glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3( 1.0f, 0.0f, 0.0f),  glm::vec3(0.0f, -1.0f, 0.0f)),  glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(-1.0f, 0.0f, 0.0f),  glm::vec3(0.0f, -1.0f, 0.0f)),  glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3( 0.0f, 1.0f, 0.0f),  glm::vec3(0.0f, 0.0f, 1.0f)),  glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3( 0.0f, -1.0f, 0.0f),  glm::vec3(0.0f, 0.0f, -1.0f)),  glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3( 0.0f, 0.0f, 1.0f),  glm::vec3(0.0f, -1.0f, 0.0f)),  glm::lookAt(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3( 0.0f, 0.0f, -1.0f),  glm::vec3(0.0f, -1.0f, 0.0f))  };  // convert HDR equirectangular environment map to cubemap equivalent  equirectangularToCubemapShader.use();  equirectangularToCubemapShader.setInt("equirectangularMap", 0);  equirectangularToCubemapShader.setMat4("projection", captureProjection);  glActiveTexture(GL\_TEXTURE0);  glBindTexture(GL\_TEXTURE\_2D, hdrTexture);  glViewport(0, 0, 512, 512);  glBindFramebuffer(GL\_FRAMEBUFFER, captureFBO);  for (unsigned int i = 0; i < 6; ++i)  {  equirectangularToCubemapShader.setMat4("view", captureViews[i]);  glFramebufferTexture2D(GL\_FRAMEBUFFER, GL\_COLOR\_ATTACHMENT0,  GL\_TEXTURE\_CUBE\_MAP\_POSITIVE\_X + i, envCubemap,  0);  glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);  renderCube(); // renders a 1x1 cube  }  glBindFramebuffer(GL\_FRAMEBUFFER, 0); |

-프레임 버퍼의 color을 가져와서 큐브 맵의 모든 면에 텍스처 타겟을 전환해 scene을 큐브 맵의 면 중 하나로 직접 렌더링함

-이것이 끝나면 envCubemap은 원래의 HDR 이미지의 큐브 맵 환경 버전 이어야함

-간단한 skybox shader

|  |
| --- |
| #version 460 core  layout (location = 0) in vec3 aPos;  uniform mat4 projection;  uniform mat4 view;  out vec3 localPos;  void main()  {  localPos = aPos;  mat4 rotView = mat4(mat3(view)); // remove translation from the view matrix  vec4 clipPos = projection \* rotView \* vec4(localPos, 1.0);  gl\_Position = clipPos.xyww;  } |

-큐브 조각의 깊이 값이 항상 10이 되도록 하는 xyww를 유의

-깊이 비교함수 변경

|  |
| --- |
| glDepthFunc(GL\_LEQUAL); |

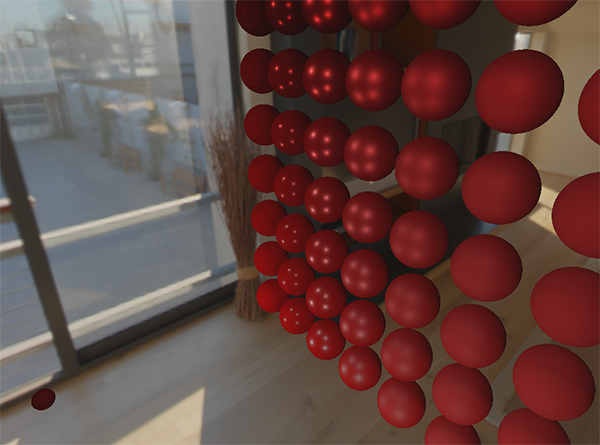
-skybox fragment shader

|  |
| --- |
| #version 460 core  out vec4 FragColor;  in vec3 localPos;    uniform samplerCube environmentMap;    void main()  {  vec3 envColor = texture(environmentMap, localPos).rgb;    envColor = envColor / (envColor + vec3(1.0));  envColor = pow(envColor, vec3(1.0/2.2));    FragColor = vec4(envColor, 1.0);  } |

-카메라의 translate 구성 요소가 무시됨으로 이 쉐이더를 큐브 위에 렌더링하면 환경맵이 움직이지 않는 배경으로 제공이 됨

-또한 HDR의 거의 모든 색상이 linear color space에 있음으로 감마 보정을 해야함

-

-HDR 맵을 읽고, equirectangular를 큐브 맵으로 전환하고, 큐브 맵을 스카이 박스로 렌더링 하였음, 또한 큐브 맵의 6개면 보드에 렌더링 할 수 있는 작은 시스템도 제작함

-이 맵은 환경 맵을 convolution할 때 다시 필요할 것임