Advanced Lighting / SSAO

SSAO Screen Space Ambient Occlusion

-사물에 의해 생기는 빛의 감쇠를 리얼타임으로 나타내는 3D 쉐이딩기법

-GPU를 기반으로 픽셀 쉐이더를 통해 구현, 순수하게 그래픽카드의 성능에 의존

-Ambient Lighting은 빛의 산란인 시뮬레이션하기 위해서 Scene에 추가하는 고정된 광원 상수

-실제의 빛은 사양한 강도와 다양한 방향, 모든 종류의 산란을 일으킴

-간접 조명 근사치중 한 유형은 Ambient Occlusion이라고 불리는데, 주름, 구멍, 빛 표면을 어둡게 해서 간접 조명을 비슷하게 만드는 방법임

-이러한 영역들은 Geometry로 가려져 있어서 빛이 통과할 곳이 더 적어서 더 어둡게 보임

-SSAO의 이미지 예제

-특정 빛이 들어가지 못하는 지역에는 더 어두워 보이는 곳이 있음

-엄청 큰 변화는 아니지만, SSAO가 있을 때의 사진이 더 현실적으로 보임

SSAO의 기본 원리

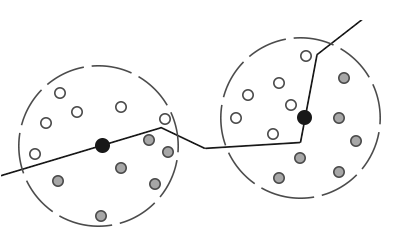
-screen-filled된 쿼드의 각 부분에 대해서 fragment의 주변 깊이 값을 기반으로 하는 폐색 계수(Occlusion factor)를 계산함

-이후 폐색 계수는 fragment의 주변 빛 요소를 줄이거나 무효화 하는데 사용됨

-폐색 계수는 fragment position을 둘러싼 구체 샘플 커널에서 여러 깊이 샘플들을 추출함

-각 샘플을 현재 fragment 깊이 값과 비교함

-fragment의 깊이보다 깊이 값이 더 높게 가진 샘플의 수가 폐색 계수를 나타냄

-선은 오브젝트의 표면

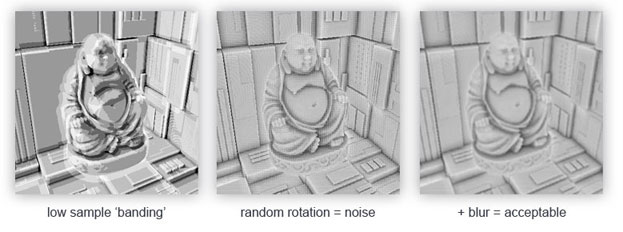
-보이는 검은 점이 fragment의 position이고, 회색 샘플이 폐색 요인에 영향을 줌

-Geometry 내부에서 더 많은 샘플을 찾을수록 fragment가 받는 조명이 적어지는 것

-샘플의 수가 품질이랑 관련이 있음을 분명히 알 수 있음

-샘플의 수가 너무 적으면 정밀도가 떨어지는 banding이라는 인공물이 생김

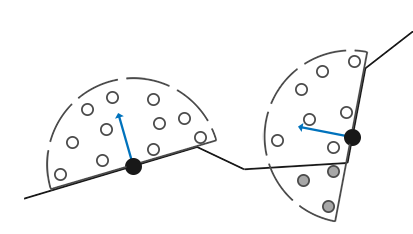
-이 때 샘플에 임의성을 추가해서 샘플이 적어도 좋은 품질을 낼 수 있음

-왼쪽이 낮은 샘플로 생긴 banding, 중간이 회전의 임의성을 넣어 품질의 상향 그러나 노이즈의 발생, 오른쪽이 블러를 추가해서 노이즈의 감소

-Crytek에서 개발한 SSAO 방법은 시각적인 스타일을 가짐

-사용되어진 샘플의 커널은 구(원)형 이였기 때문에 평평한 벽에서는 절반의 샘플을 폐색 계수로 가지기 때문에 회색빛을 띄게 됨

-이러한 이유 때문에 우리는 구형을 사용하지 않고, 반구형 샘플 커널을 사용할 것임

-파란선은 normal vector

-이렇게 사용하면 Ambient Occlusion의 회색 느낌은 사라지고 일반적으로 사실적인 결과가 생성되어짐

Sample Buffers

-SSAO에 필요한 기하학적 정보들

-Fragment당 Positon vector

-Fragment당 Normal vector

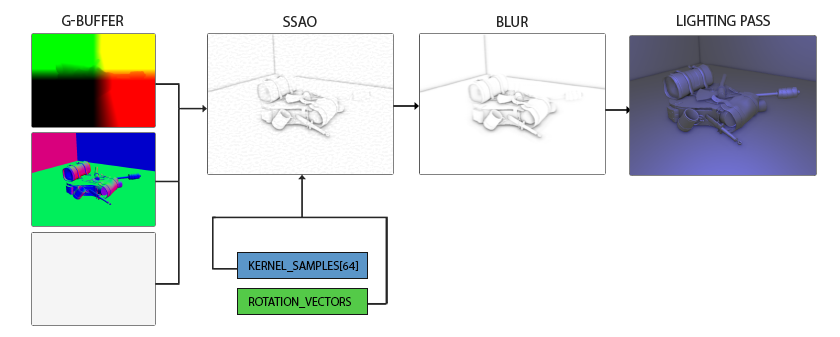
-Fragment당 albedo color

-Sample Kernel

-Sample Kernel의 회전을 위한 랜덤 순환 벡터

-Fragment view space Position을 사용하면 fragment의 view-space 표면 법선을 중심으로 샘플 반구형 커널의 방향을 맞출 수 있음

-이 커널을 사용해 다양한 오프셋에서 위치 버퍼 텍스처를 샘플링할 수 있음

-SSAO기술은 Screen-space 기술이라서 screen-filled된 2D 쿼드에서 각 fragment에 미치는 영향을 계산하지만, 이는 장면의 기하학적 정보가 없음을 의미함

-기하학적인 조각 데이터를 screen-space 텍스처로 렌더링 한 다음에 나중에 SSAO 쉐이더에 전송하는 것

-그렇게 해서 fragment당 기하학적 데이터에 접근을 할 수 있음

-이전에 사용한 deferred shading에 이어서 작성할 예정

-G-buffer에서 데이터가 있음으로 geometry의 fragment shader

|  |
| --- |
| #version 460 core  layout (location = 0) out vec4 gPosition;  layout (location = 1) out vec3 gNormal;  layout (location = 2) out vec4 gAlbedoSpec;  in vec2 TexCoords;  in vec3 FragPos;  in vec3 Normal;  void main()  {  // store the fragment position vector in the first gbuffer texture  gPosition = FragPos;  // also store the per-fragment normals into the gbuffer  gNormal = normalize(Normal);  // and the diffuse per-fragment color, ignore specular  gAlbedoSpec.rgb = vec3(0.95);  } |

-SSAO는 view-space에서 구현하는 것이 좋음

-geometry stage에서 vertex shader가 제공하는 FragPos는 view공간으로 변환되어짐

-이 view 공간에서 추가 계산이 이루어지기 때문에 G-buffer의 위치와 normal vector가

view공간에 있는지 확인해야함

-gPosition color buffer

|  |
| --- |
| glGenTextures(1, &gPosition);  glBindTexture(GL\_TEXTURE\_2D, gPosition);  glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA16F, SCR\_WIDTH, SCR\_HEIGHT, 0,  GL\_RGBA, GL\_FLOAT, NULL);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_CLAMP\_TO\_EDGE);  glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_CLAMP\_TO\_EDGE); |

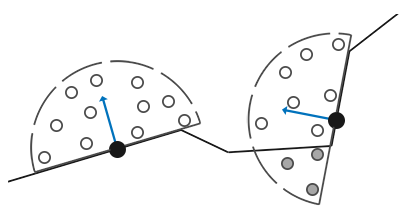
-위치 값이 0~1로 고정되지 않음, 또한 GL\_CLAMP\_TO\_EDGE 랩핑 방법을 사용함

-이렇게 하면 텍스처의 기본 좌표 영역 외부의 화면 공간에서 Position/Depth 값을 오버 샘플링하지 않음

Normal-Oriented hemisphere

-실제 반구 샘플 커널과 임의로 회전 시키는 방법

-법선 벡터가 양의 z 방향을 가리키는 tangent 공간에서 샘플 커널을 생성할 것임

-샘플 커널

|  |
| --- |
| std::uniform\_real\_distribution<float> randomFloats(0.0, 1.0);  std::default\_random\_engine generator;  std::vector<glm::vec3> ssaoKernel;  for (unsigned int i = 0; i < 64; ++i)  {  glm::vec3 sample(  randomFloats(generator) \* 2.0 – 1.0, //x  randomFloats(generator) \* 2.0 – 1.0, //y  randomFloats(generator) //z  );  sample = glm::normalize(sample);  sample \*= randomFloats(generator);  ssaoKernel.push\_back(sample);  } |

-x, y는 –1 ~ 1 사이의 값을 가짐

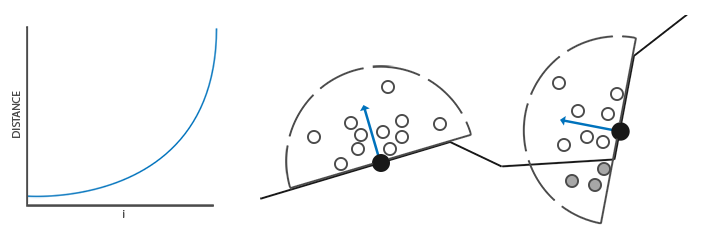
-z는 반구형이기 때문에 0 ~ 1의 값을 가짐, -1~1이면 구형을 가지게 될 것임

-샘플 커널은 표면 법선을 따라서 정방향으로 배치되기 때문에 모두 반구에 있게 됨

-실제 fragment에 가까운 폐색에 더 큰 가중치를 두기 위해서 가속 보간 함수를 사용

|  |
| --- |
| float scale = (float)i / 64.0;  scale = lerp(0.1f, 1.0f, scale \* scale);  sample \*= scale;  ssaoKernel.push\_back(sample);  } |
| float lerp(float a, float b, float f)  {  return a + f \* (b - a);  } |

-이렇게 하면 샘플들을 원래의 fragment position에 가깝게 배치가 됨

-각 커널의 샘플은 view space fragment positon을 샘플 주변 geometry로 오프셋하는데 사용되어짐

-사실적인 결과를 얻을려면 자원을 많이 소모하는 많은양의 샘플이 필요하게 됨

-그러나 fragment단위로 semi-random rotation/noise를 도입 하면 필요한 샘플 수를 줄일 수 있음