Advanced OpenGL / Depth Testing

-이전 강좌에서 3D컨테이너를 여러 개 생성할 때, 화면에 이상하게 출력되는 것을 방지 하기 위해 코드를 추가하였었음

|  |
| --- |
| glEnable(GL\_DEPTH\_TEST); |
| glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT); |

-이번 강좌에서는 실제로 깊이 버퍼가 저장하는 이러한 깊이 값과, 조각이 다른 조각 뒤에 있는지 여부를 결정하는 방법에 대해 배울 것

-깊이 버퍼는 윈도우 시스템에 의해서 자동으로 생성됨, 16, 24, 32 비트로 만들어지는데 보통 24비트인 깊이 버퍼가 표시됨

-깊이 테스트가 활성화 되면 OpenGL은 깊이 버퍼의 내용에 대해서 fragment의 깊이 값을 테스트 함

-테스트가 통과하면 깊이 버퍼가 새 깊이 값으로 업데이트 됨

-깊이 테스트는 fragment shader가 실행 된 후, 스크린 공간에서 수행되어짐

-gl\_FragCoord 변수를 통해 엑세스 가능, 이때, x, y, z 구성요소(x=0, y=0이 왼쪽아래)중 z구성요소가 깊이를 나타냄

-깊이 테스트가 활성 된 경우에 깊이 마스크를 버퍼에 쓰지 못하게 할 수 있음

|  |
| --- |
| glDepthMask(GL\_FALSE); |

Depth test function

-깊이 테스트에 사용되는 비교 연산자를 수정할 수 있음

|  |
| --- |
| glDepthFunc( ㅁㅁㅁ ); |

-사용되는 연산자

-GL\_ALWAYS : 깊이 테스트가 항상 통과함

-GL\_NEVER : 깊이 테스트가 절대 통과하지 못함

-GL\_LESS : Fragment의 깊이 값이 저장된 깊이 값보다 작은 경우 통과함

-GL\_EQUAL : Fragment의 깊이 값이 저장된 깊이 값이랑 같은 경우 통과함

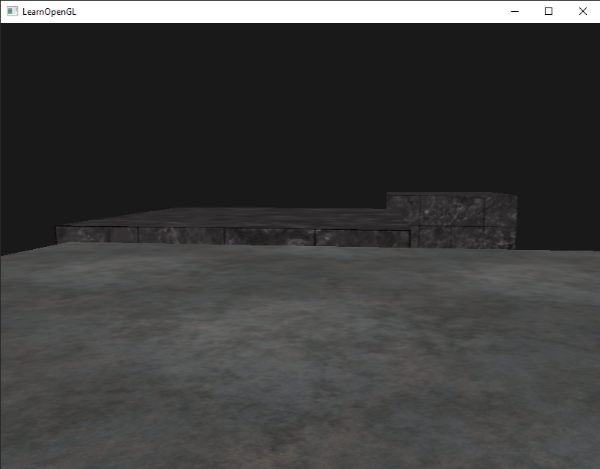
-GL\_LEQUAL : Fragment의 깊이 값이 저장된 깊이 값이랑 작거나 같으면 통과함

-GL\_GREATER : Fragment의 깊이 값이 저장된 깊이 값보다 크면 통과함

-GL\_NOTEQUAL : Fragment의 깊이 값이 저장된 깊이 값과 다르면 통과함

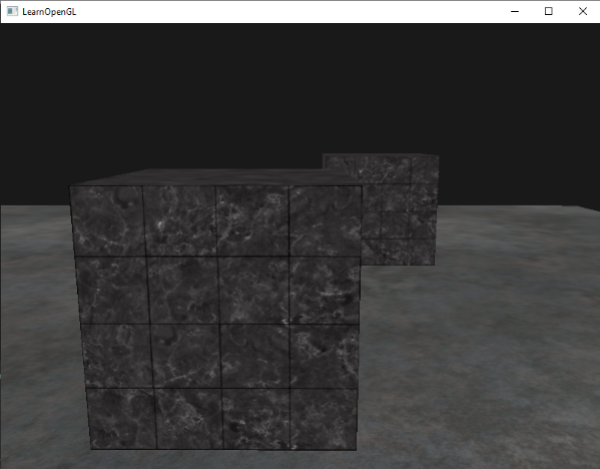
-GL\_GEQUAL : Fragment의 깊이 값이 저장된 깊이 값보다 크거나 같으면 통과함

ex) GL\_ALWAYS

-깊이 테스트가 항상 통과함으로 마지막에 그려진 Fragment는 항상 앞에 위치하게 됨

-그려진 순서 : 큐브 -> 바닥

ex) GL\_LESS

-생성하고 있는 오브젝트(바닥)의 깊이가 현재 저장되어진 값(큐브의 깊이 값)보다 크기 떄문에 통과하지 못하여 큐브가 앞에 위치하게 됨

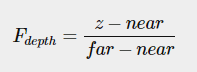
-생성 순서 : 큐브 -> 바닥

Depth value precision 깊이 값 정밀도

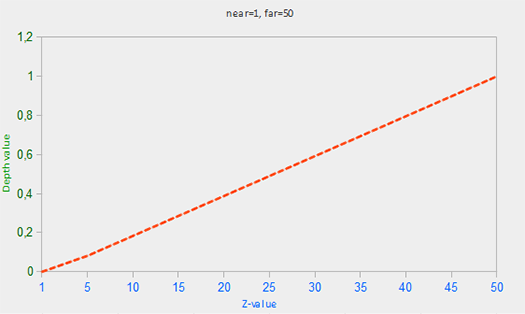
-깊이 버퍼는 0.0 ~ 1.0사이의 깊이 값을 포함하며, 뷰어에서 보았을 때의 Scene의 모든 객체의 z값을 비교함

-이 z값은 Projection Frustum의 근사치와 근사치 사이의 값이 될 수 있음

1)

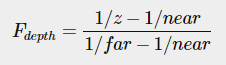


-이 방정식은 어떤 물체를 밑면과 평행하게 잘랐을 때, 밑면과 평면 부분의 깊이 값 z를 [0, 1]로 변환함

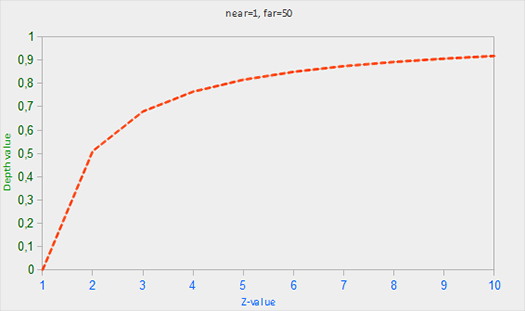
-모든 물체가 가까우면 0에 가까운 값, 멀면 1에 가까운 값을 제공하게 됨

-그러나 이러한 방정식은 사용되지 않음

2) 1/z에 비례함

-1/z에 비례하기 때문에 1.0~2.0 사이의 z값은 1.0~.05사이의 깊이 값을 발생

-작은 z에서 엄청난 정밀도를 제공함, 이보다 큰 50~100사이의 값은 정밀도의 2%정도만 차지함



-이렇게 사용되는 이유에 대한 내 생각

|  |
| --- |
| 실제 사람이 먼 공간을 바라볼 때, 일정 이상의 거리(a)를 보면 잘 보이지 않는다.  이때, 이 이상의 거리(a+@)와 그 거리(a) 에 대해서 얼마나 보인다, 보이지 않는다는 중요하지 않다고 생각된다. 어짜피 보이지 않는다는 점에서는 똑같다.  가까운 거리에서는 물체가 가까운 것과 멀리 있는 것의 정밀도 차이는 보이지 않는 곳에서의 정밀도 변화보다는 커야 한다고 생각한다.  정확한 생각은 아닐지 몰라도 이러한 이유 때문에 선형이 아닌 방정식을 사용하는 것 같다. |

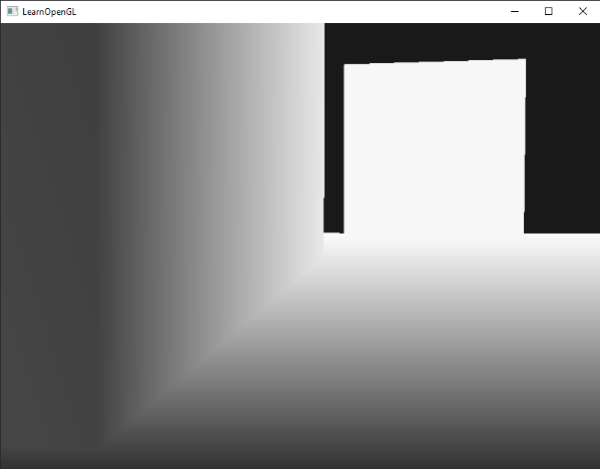
Visualizing the depth buffer 깊이버퍼 시각화

-gl\_FragCoord의 z값이 깊이 값을 가진다고 했음

-이 z값을 이용해서 색상으로 출력한다면 장면의 조각의 깊이 값을 시각화가 가능함

|  |
| --- |
| void main()  {  FragColor = vec4(vec3(gl\_FragCoord.z), 1.0);  } |

-멀수록 1에 가깝기 때문에 밟아지고, 가까울수록 0에 가깝기 때문에 어두워질 것임

-가까운 부분에서는 천천히 어두워짐, 높은 정밀도를 가진것이 나타남

-먼 부분에서는 갑자기 하얀색으로 확 바뀌는데, 낮은 정밀도를 가진것이 나타남

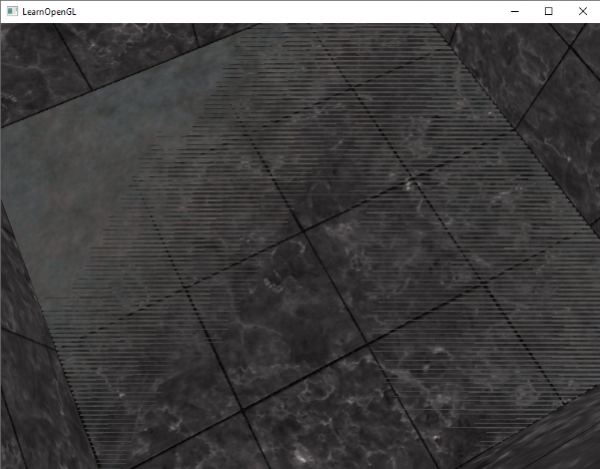
-이는 깊이 값 정밀도가 비선형이기 때문에 나타나는 현상

-비선형 방성식을 선형으로 바꾸는 방법도 있음

|  |
| --- |
| #version 460 core  out vec4 FragColor;  float near = 0.1;  float far = 100.0;    float LinearizeDepth(float depth)  {  float z = depth \* 2.0 - 1.0; // back to NDC  return (2.0 \* near \* far) / (far + near - z \* (far - near));  }  void main()  {  float depth = LinearizeDepth(gl\_FragCoord.z) / far;  FragColor = vec4(vec3(depth), 1.0);  } |

Z-fighting

-두 개의 평면이나 삼각형이 너무 가깝게 정렬되는 경우, 누가 더 가까지에 있는지 알아내기 힘든 경우에 이상한 모양이 나타남

-이는 완전히 막을수는 없지만, scene에서는 완화하거나 예방하는데 도움이되는 방법이 있음

Prevent Z-fighting

1)오브젝트가 너무 가깝지 않게해서 겹치지 않게 하는 것

-사용자의 입장에서 거의 눈에 띄지 않는 간격만큼 띄우기를 생성하면 제거할 수 있음

-그러나 각 수동으로 설정해야 하기 때문에, 모든 물체를 테스트해야 함

2)가까운 두 평면을 최대한 멀리 설정

-근사 평면 가까이에 있을 때, 정밀도가 크다고 하였음, 가까운 평면을 뷰어에서 멀리 설정하게 하면 전체 절두체 범위에 대해서 더 높은 정밀도를 가지게 되지만, 가까운 평면을 너무 멀리 설정하면 가까운 물체가 잘릴 수 있음

3)더 높은 정밀도의 버퍼를 사용하는 것

-32비트의 깊이 버퍼를 사용하면 정밀하게 z-fighting을 줄일 수 있음