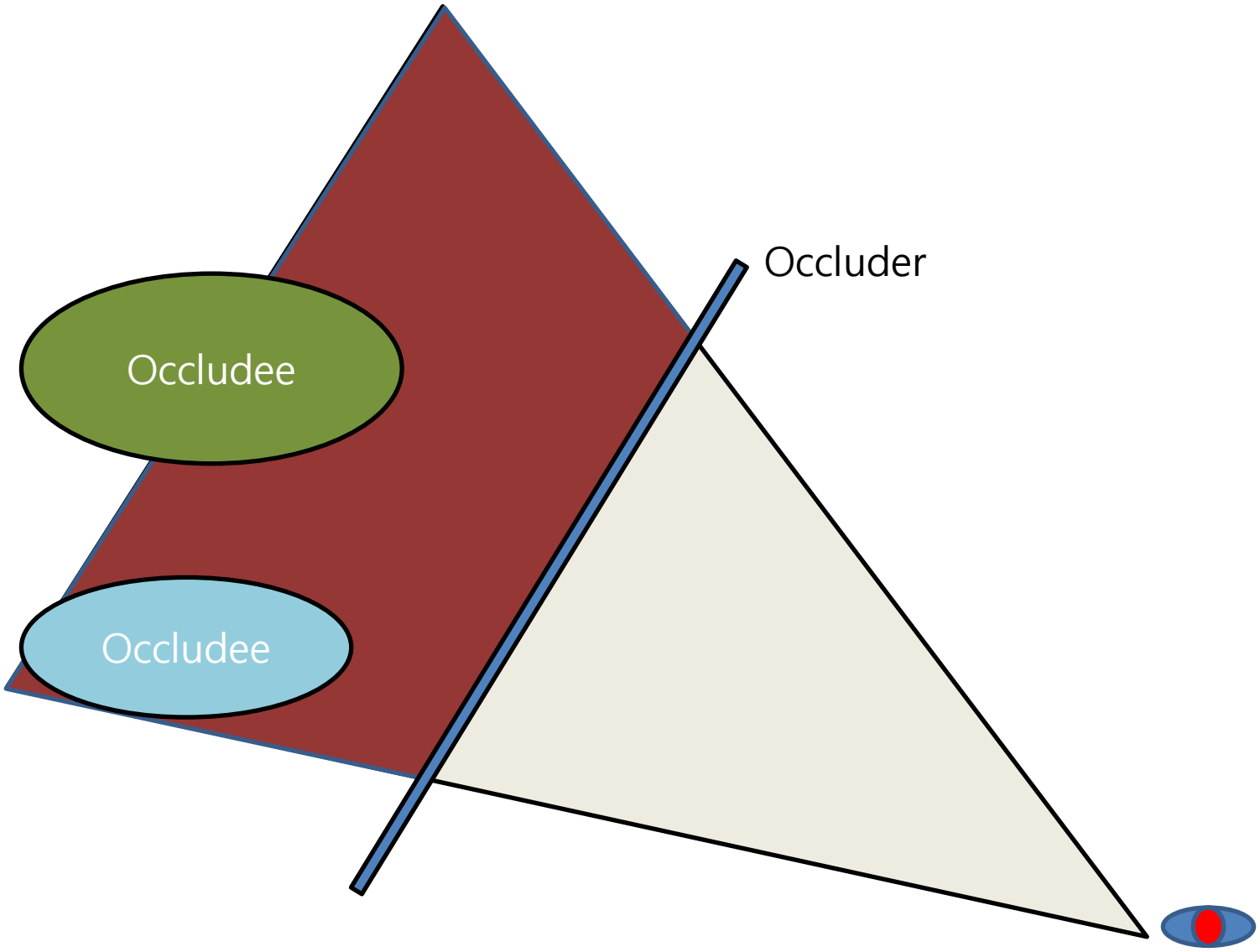
Hierarchical Z Map Occlusion Culling

유영천

KASA Study

Occlusion Culling

- 가려진 물체를 그리지 말자.
- Occluder가 Occludee를 가림


Occluder
Occludee
Occludee

Simple Occlusion Culling

- 디자이너가 맵에 Plane(판대기) 배치
- 런타임에 카메라의 Eye포인트와 판대기를 연결한 뷰프러스텀 계산-Near Plane은 Occlusion면이 되어야함.
- 이 뷰프러스텀에 들어가는 오브젝트는 렌더링 안함.
- 은근 실제 게임에 많이 쓰인다.

장점

- 구현이 쉽다.(프로그래머한테 쉽다)
- 효과는 매우 확실!

단점

- 구현하기 ~~귀찮다~~어렵다.(디자이너한테 어렵다)
- Occlusion을 정밀한 단위로 만들 수 없다.

H/W Occlusion Culling

- Occluder가 렌더링 된 상태에서 (Z-Buffer를 유지한 상태에서) Occludee를 렌더링 g 하여 정말 그려지나 테스트.
- D3D9에서부터 표준적으로 지원.
- 물론 D3D9 디바이스에서도 지원 안하는 놈이 있다.
- D3D11 API에서도 D3D9 API와 비슷한 형태로 지원함.

원리

- Occluder(raw맵 데이타나 거의 꼭 맞는 바운딩매쉬)를 그려서 Z-Buffer를 구성한다.
- Occludee(아마도 캐릭터나 맵데이타,기타 배치 가능한 오브젝트)를 그리되 pixel은 write하지 않고 z-test만 한다.
- 그려지는 픽셀 수를 GPU측 메모리에 기록해둔다.
- 나중에 GPU로부터 픽셀수를 얻어온다.

HW Occlusion Query @D3D9

- // Initialize
- LPDIRECT3DQUERY9 Query
- pDevice->CreateQuery(D3DQUERYTYPE_OCCLUSION, &pQuery);

- // Begin Test
- pQuery->Issue(D3DISSUE_BEGIN);

- //
- // Drawing
- //

- // End Test
- pQuery->Issue(D3DISSUE_END);

- // 실제로 렌더링된 픽셀 수가 dwPixelCount로 저장되어 나옴.
- DWORD dwPixelCount = 0;
- pQuery->GetData(&dwPixelCount,sizeof(DWORD), D3DGETDATA_FLUSH);

장점

- 구현이 쉽다.(프로그래머한테 쉽다.)
- 구현이 쉽다.(디자이너는 할 일이 아예 없다.)

단점

- 성능이 별로다.
- 심지어 더 느려지기도 한다!!!

왜 성능이 안나올까?

- DrawCall 회수를 줄이지 못한다.(멀티패스로 오브젝트를 그리는 경우는 제외)
- 두번 그려야 하므로 쉐이더에서 엄청나게 부하가 걸리지 않는 이상 더 빨라질리가 없다. 더 느려진다.
- Occludee의 크기가 커질수록 Z-Test 회수도 많아진다. Z-Test
- 픽셀 수를 받아올때 PCI버스를 타고 GPU Memory-> System Memory로 가져온다.

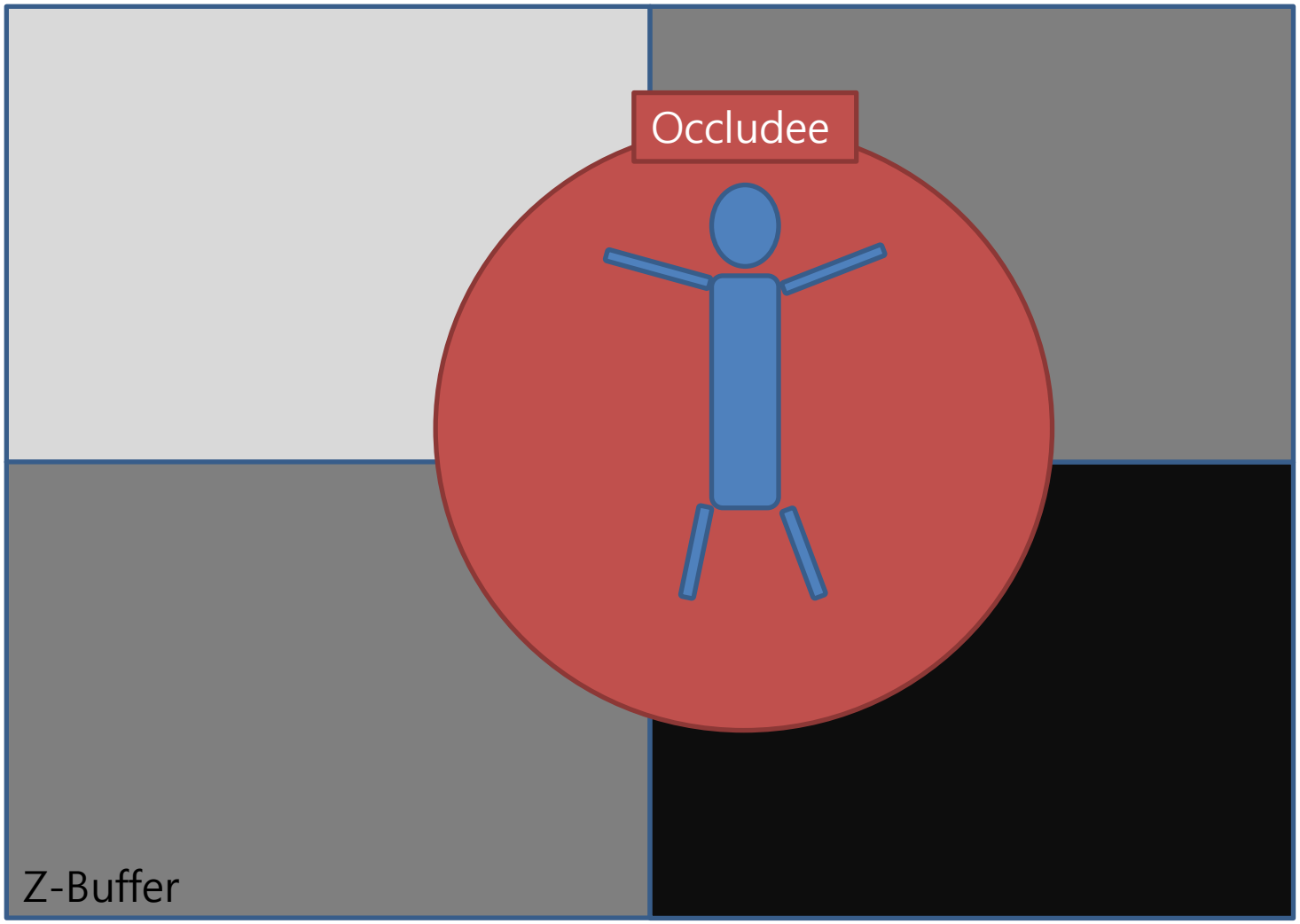
Hierarchical Z Map Occlusion Culling

- 정말 그려지는지 테스트-기본 아이디어는 같다.
- Occluder와 Occludee가 꼭 정밀해야하는가? 간단한 매쉬(경계구)로 대치하자.
- 그렇다면 Z-Buffer의 해상도도 정밀할 필요가 없다.
- 낮은 해상도의 Z-Buffer를 사용하면 Z-Buffer의 읽기/쓰기 시간을 줄일 수 있다.
- 경계구 가지고 테스트할건데 실제 그려볼 필요나 있을까? 아예 그리지도 말고 구의 한 점만 프로젝션 해서 depth값을 비교하자.
- 직접 버퍼에 그릴 필요가 없다면 Pixel Shader쓸 필요도 없으니 Compute Shader로 경계구 하나마다 스레드를 할당해서 depth테스트를 병렬화 시키자.

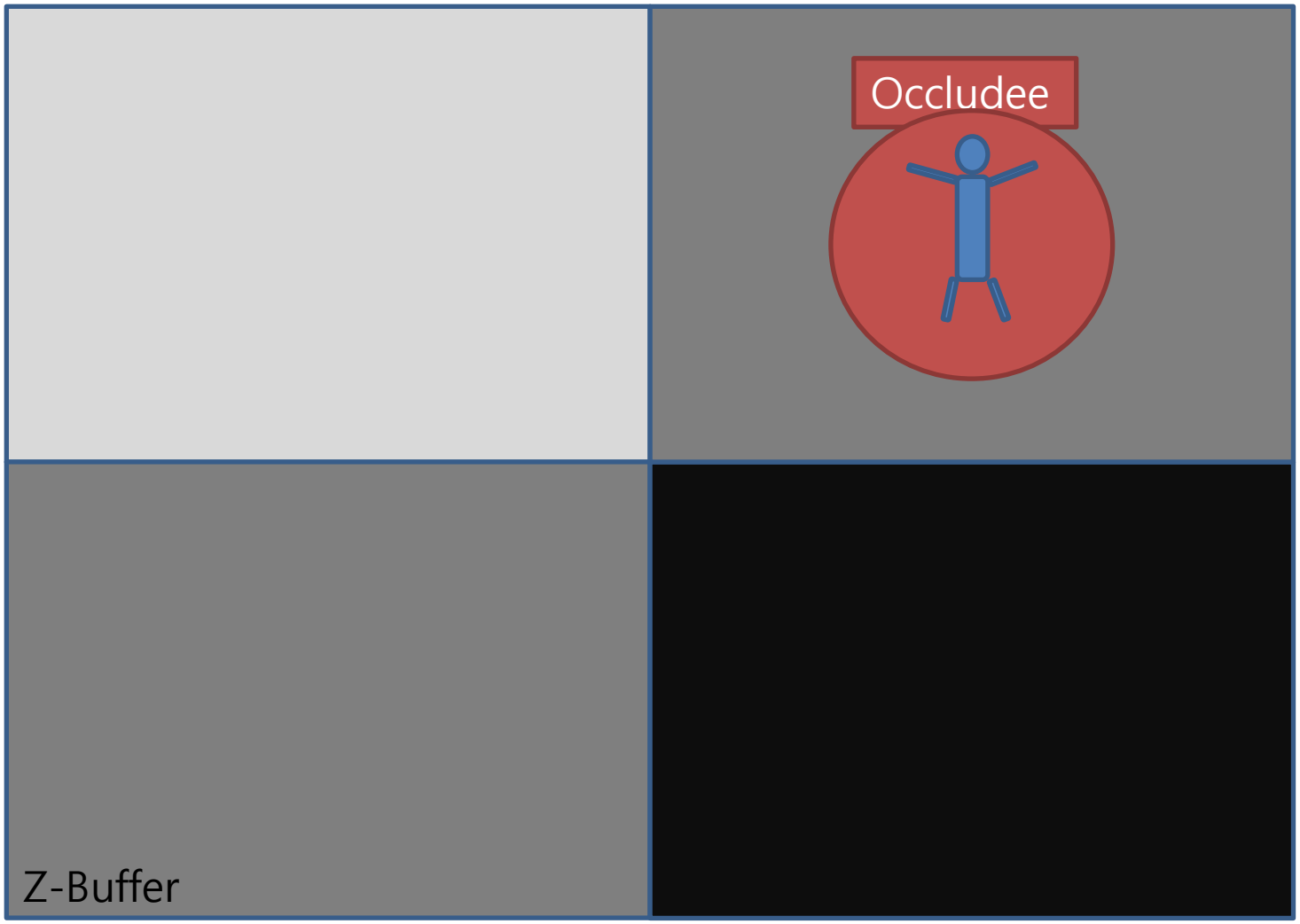
기본전략

- screen space에서 Occludee를 대표하는 한 점과 Occludee가 그려질 영역의 depth값을 비교.
- Occludee의 크기가 크면 Z-Buffer해상도는 낮아도 된다.
- Occludee의 크기가 작으면 Z-Buffer의 해상도가 높아야 한다.
- Occludee의 크기에 따라 가능하면 작은 크기의 Z-Buffer를 억세스한다.
- 기준은 Occludee가 screen space에서 2x2텍셀에 맞아들어가는지.

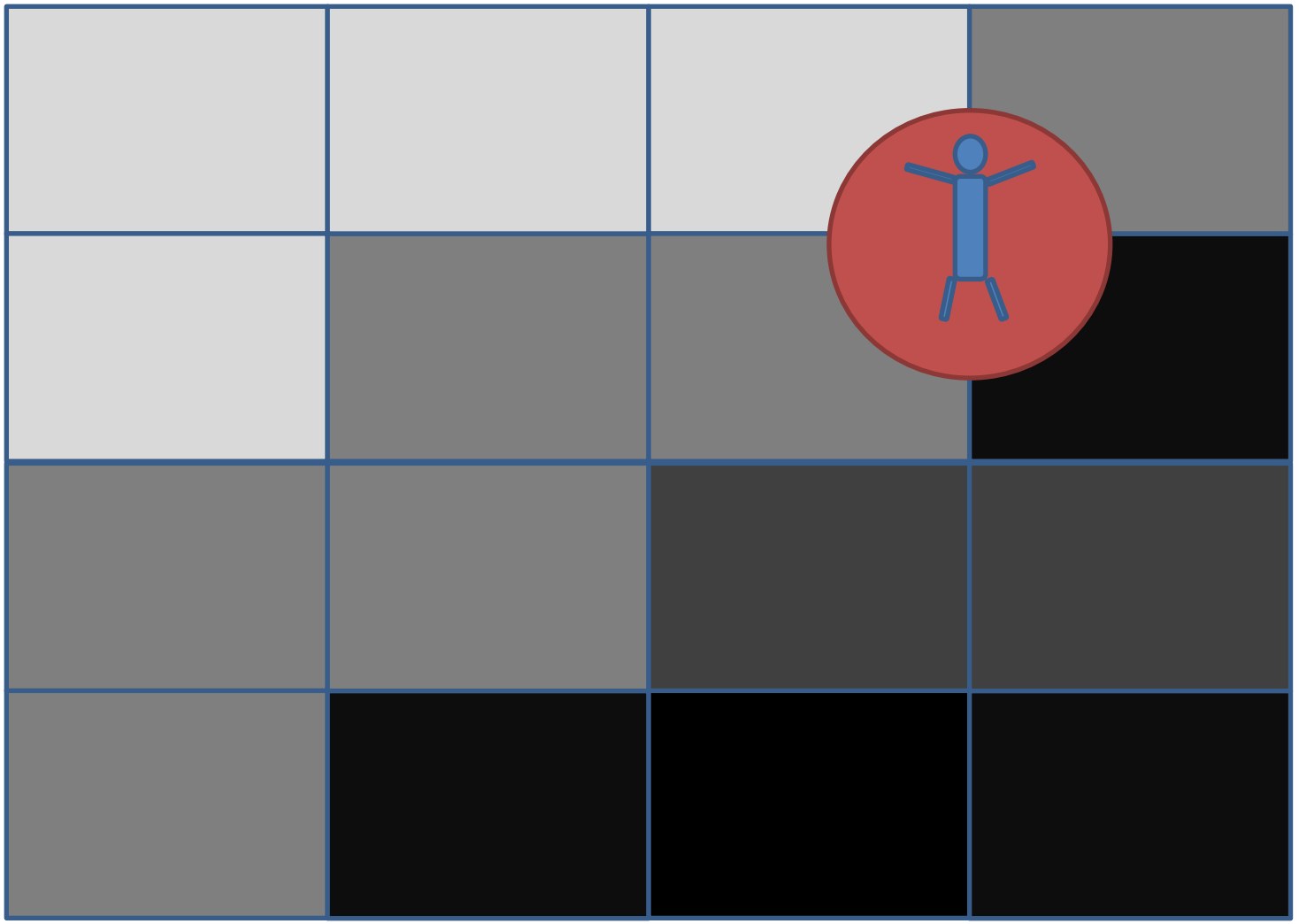
극단적으로 오브젝트 크기가 이 정도면 Z-Buffer의 해상도는 2x2로 충분



오브젝트 크기가 이 정도면 Z-Buffer의 해상도가 충분치 않음



오브젝트 크기가 이 정도면 Z-Buffer의 해상도는 4x4가 적당



왜 2x2?

- screen space 상에서 항상 Occludee가 그려지는 영역 전체를 커버할 수 있는 사이즈.
- 1x1일 경우 screen space에서 Occludee가 텍셀의 경계에 걸칠 경우 잘못된 depth비교를 하게 된다.
- 2x2보다 크면 샘플링 회수가 늘어나서 성능이 떨어진다.
- Occludee의 경계구가 screen space상에서 반드시 1x1 - 2x2가 되도록 공식을 사용한다.

  -> float fLOD = ceil(log2(Width));

구현전략-초기화

- 적당한 해상도(ex: 512x256)의 렌더타겟을 만든다.
- 1x1까지 밉체인을 생성한다.
- 512x256 -> 256x128 -> 128x64 -> 64x32 -> 32x16.....

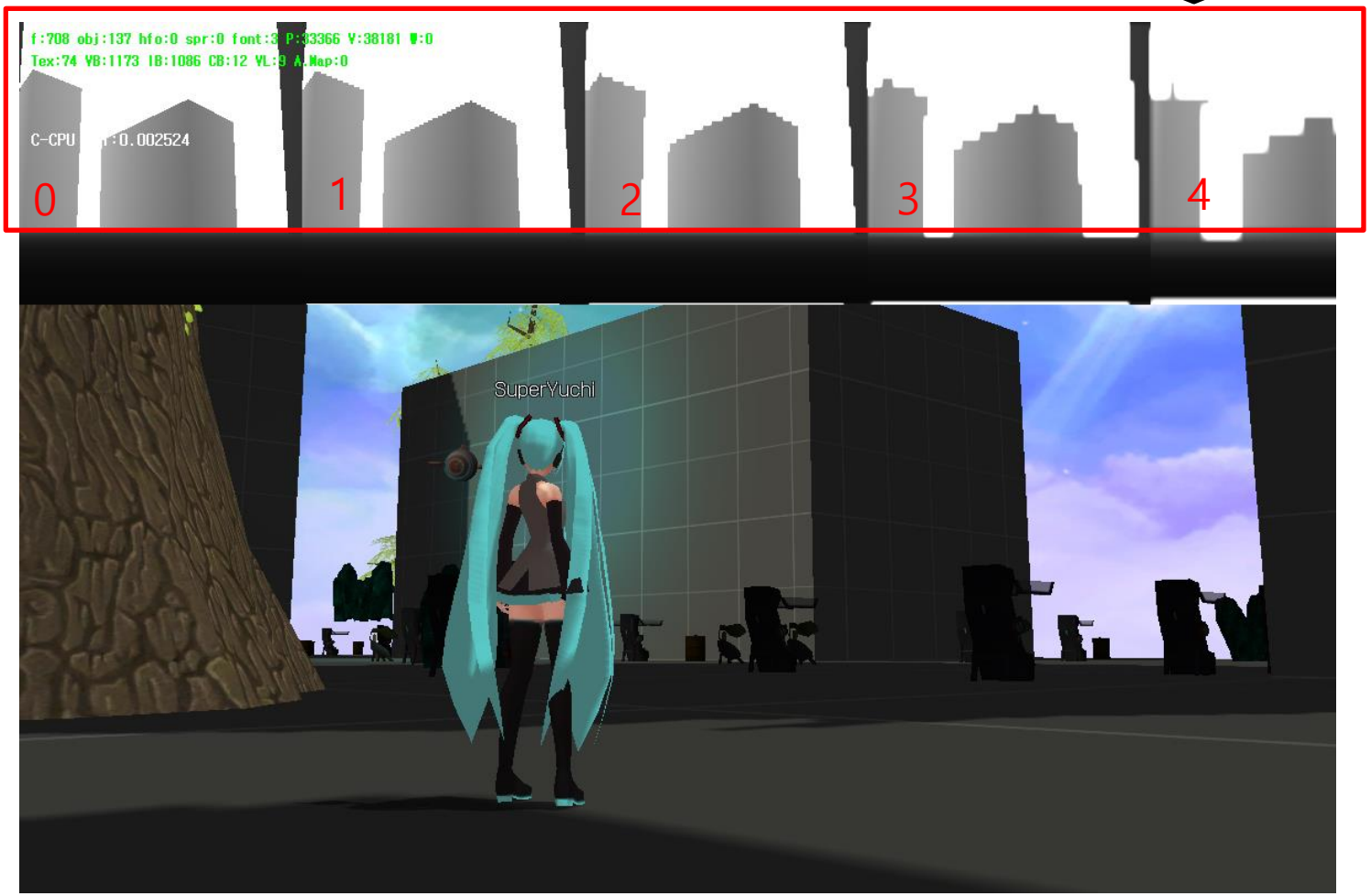
구현 - Occluder렌더링

- Mip level 0에서 Occluder를 렌더링. -> Depth map을 텍스쳐로 보관.

구현 – Depth map Down sapmling

- Depth map을 level 0부터 마지막 level(1x1)맵까지 다운샘플링해서 보관.

Z-buffer Texture mip chain



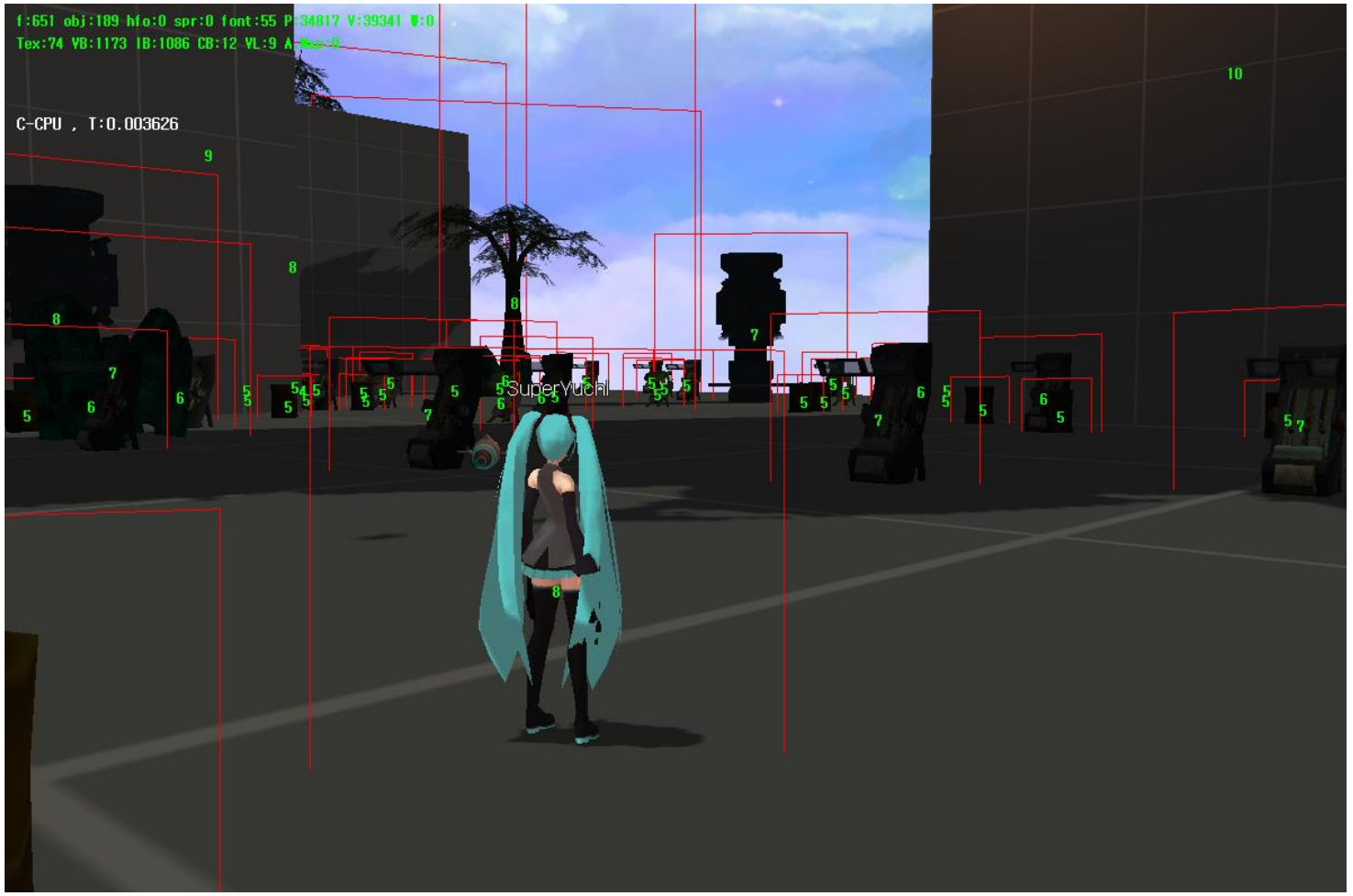
구현 – Occludee렌더링

- Occludee를 직접 렌더링 하지는 않는다.
- 화면에 렌더링될 Occludee의 경계구 목록을 만든다.
- 경계구 목록을 Compute Shader로전달.

구현 – 깊이비교(with Compute Shader)

- screen space에서 Occludee의 경계구 목록이 렌더링 될 영역을 구한다.
- Occludee가 screen space에서 차지할 영역을 계산한다.
- screen space에서의 Occludee 가로길이에 따라 깊이 비교를 할 Z-Map텍스쳐의 Mip레벨을 선택한다.
- 카메라 eye로부터 가장 가까운 경계구위의 한점을 구하고 screen space로 변환한다.
- screen space에서 경계구가 위치하는 영역의 2x2텍셀을 샘플링한다.
- 경계구를 대표하는 한 점의 Depth값과 2x2텍셀로 얻은 depth값 비교.
- 경계구의 Depth > 텍셀의 Depth이면 culling.

빨간박스 : screen space에서 Occludee가 그려지는 영역
녹색 숫자 : Occludee가 z-test에 사용할 밉맵 레벨



미쿠 캐릭터의 경우 mip level 8의 depth map으로 테스트한다.
Lv 0가 512x256일때 미쿠의 박스의 width는 화면 길이의 절반 정도를 차지하므로 256정도로 볼 수 있다.
log2(256) = 8 이므로 mip level 8의 2x1짜리 depth map을 사용

구현 – 결과 얻어오기

- GPU메모리는 직접 읽기가 불가하다.
- D3D11_USAGE_STAGING ,D3D11_CPU_ACCESS_READ 으로 만든 버퍼로 copy, CopyResource() 사용.
- 위의 버퍼에 대해 Map(),UnMap()을 사용해서 결과값을 읽어옴.

장점

- Occludee를 직접 그리지 않으니 Raster Operation이 없다.
- Occludee의 모든 픽셀이 아닌 대표하는 한 점만으로 Z-Test를 하므로 Z-Test회수 자체가 적다.
- 가능한 작은 사이즈의 Z-Buffer Texture를 억세스하므로 텍스쳐 캐쉬효율 증가, bandwidth절약.
- 2x2 = 4번만 depth값을 읽어오므로 Depth맵 읽기 시간 절약.
- GPGPU(Compute Shader)를 이용한 병렬처리 가능.

단점

- Occluder 렌더링 시간을 줄여주지는 못한다. 이것은 다른 차원의 문제.
- Culling결과를 가져오기 위해 PCI-E버스를 통해 GPU Memory->System Memory로 전송하므로 이 부분은 여전히 느리다.
- 경계구의 한 점으로 depth테스트를 하므로 약간 부정확하다. 가려진 오브젝트가 렌더링 될 수 있다. 단 반대의 경우는 일어나지 않는다.

성능비교

<입력 오브젝트 대략 700개일때>

No Occulusion Culling
-> 430 FPS, 렌더링된 오브젝트 수 : 376개

D3DQuery Occlusion Culling
-> 561 FPS, 렌더링된 오브젝트 수 : 31개

Hierarchical Z Map Occlusion Culling
-> 713 FPS, 렌더링된 오브젝트 개수 : 31개

테스트하는 Occludee 개수가 많아질수록 격차가 더 커짐

더 생각해보기

Compute Shader에서 Thread개수와 Group개수

*Thread 개수는 GPU의 SP(Core)에 연관
*Group은 GPU의 SM에 맵핑

Threads : 256일때
Occludee가 256이면 Group은 1개만 필요.

일반적으로 nVidia의 데스크탑용 GPU라면 SM은 4개 – 16개 장착. 따라서 대부분의 SM이 단 1개의 SM의 작업이 끝나기를 기다린다.

SM이 16개라면
Threads : 16 , Thread Group : 16 이 적당.

SM이 4개라면
Threads : 64 , Thread Group : 4 이 적당.

Threads per group의 수는 많은 것이 좋지만 SM을 놀지 않도록 하는 것이 더 중요.

자동화된 Occluder생성

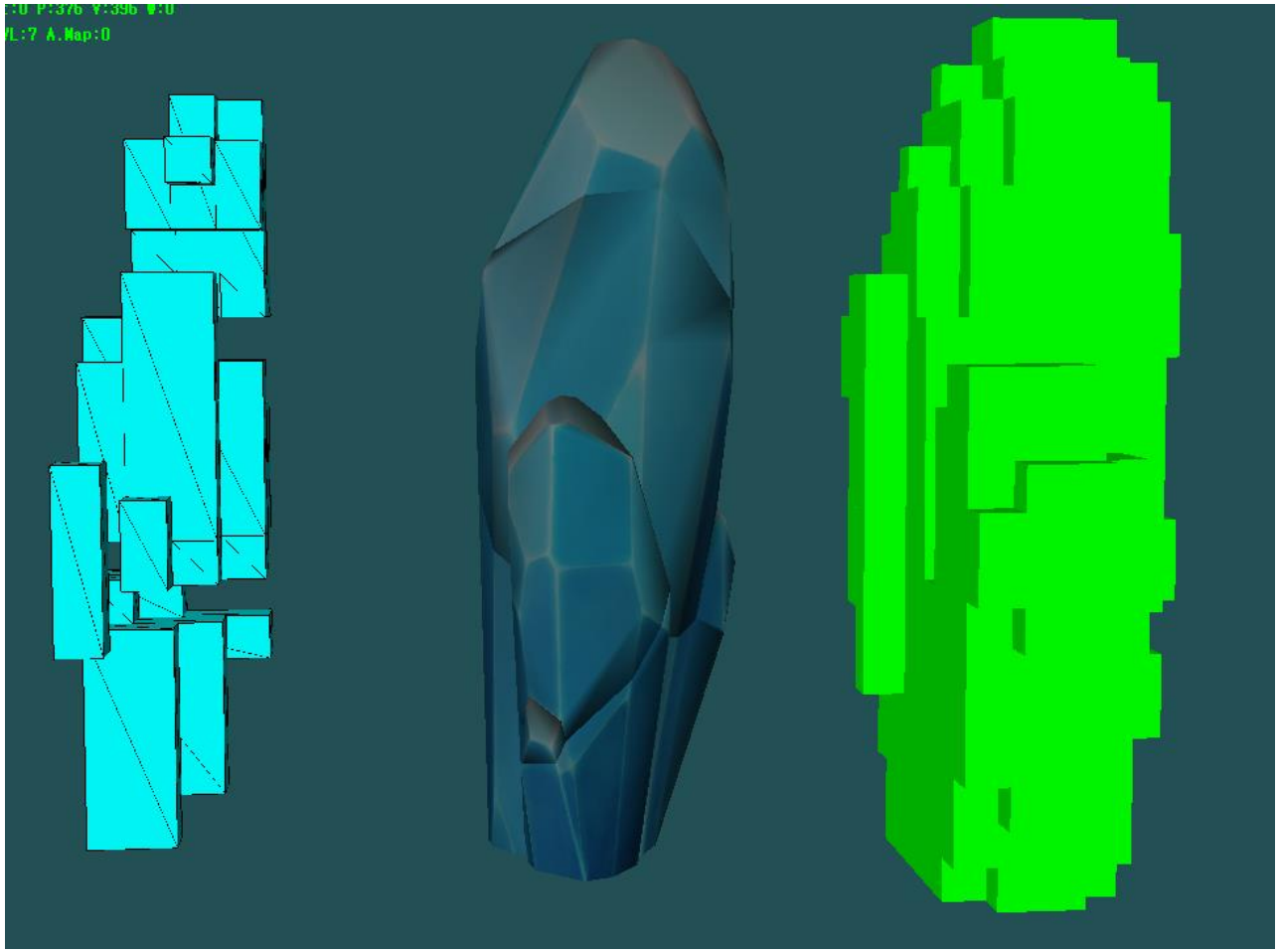
1. 삼각형매쉬를 voxelize시킨다.
2. Voxel들 안에서 가장 밀도가 높은 voxel을 찾는다.
3. 해당 voxel위치에서 박스를 만든다.
4. 외곽에 부딪칠때까지 박스를 확장한다.
5. 3으로 돌아간다. 일정 용적률을 채울때까지 반복

<참고>

http://www.nickdarnell.com/2011/06/hierarchical-z-buffer-occlusion-culling-generating-occlusion-volumes/

http://www.mpi-inf.mpg.de/~mschwarz/papers/vox-siga10.pdf

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.6993


VL:7 A.Map:0

참고문헌

- http://www.nickdarnell.com/2010/06/hierarchical-z-buffer-occlusion-culling/

Q / A