SW Occlusion Culling

유영천

Blog: megayuchi.com

Tw : @dgtman

Culling

- View Frustum Culling
- BSP/PVS
- BSP/Room/Portal
- Occlusion Culling
  - SW Occlusion Culling
  - HW Occlusion Culling with D3DQuery
  - Hierarchical Occlusion Culling with Compute Shader

Occlusion Culling

- HW Occlusion Culling
  - D3DQuery
  - Hierarchical Occlusion Culling with Compute Shader

- SW Occlusion Culling

HW Occlusion Culling with D3DQuery

- 렌더링할 오브젝트들을 수집
- Occluder로 그릴 오브젝트들을 선별
- Occludee로 z-test할 오브젝트들을 선별
- Occluder 렌더링
- D3DQuery를 on하고 Occludee를 렌더링
- 렌더링된 pixel 개수를 얻어서 0이면 렌더링 파이프라인에서 제외
- 일반적인 렌더링과 거의 같다. 다음의 항목만 빼고
  - GPU상의 pixel out off, texturing off
  - 최대한 간단한 shader사용

HW Occlusion Culling with D3DQuery

- Shader로 인한 부하, texturing으로 인한 부하가 큰 경우에만 의미가 있다.
- Z-Test를 위한 준비작업에 너무 많은 자원이 필요하다.
- 많은 경우 더 느려진다.

Hierarchical Occlusion Culling with Compute Shader

[CPU코드에서]

- KD-Tree View Frustum Culling으로 오브젝트들을 수집
- Occluder로 그릴 오브젝트들을 선별
- Occludee로 test할 오브젝트들을 선별
- Occluder 렌더링
- Z-Buffer를 1x1사이즈까지 리사이즈
- Occludee로 사용할 오브젝트의 Bounding Sphere를 CS로 전달.

[Compute Shader에서]

- Bounding Shpere를 화면 좌표계로 변환하고 가장 1픽셀에 대응할 수 있는 zbuffer를 선택
- Z값 비교 후 결과를 SRW Buffer에 저장.

Hierarchical Occlusion Culling with Compute Shader

- Occluder를 그리는데 걸리는 시간은 D3Dquery방식과 똑같음.
- 추가적으로 z-buffer resize시간이 필요함.
- Occludee를 그리는 대신 CS를 호출. 이 부분에선 D3DQuery보다 명백히 빠름.
- Culling에 걸리는 시간 자체는 대개 D3DQuery보다 50%이상 빠름. 대신 정밀도는 떨어짐.

- 참고 :

https://www.slideshare.net/dgtman/hierachical-z-map-occlusion-culling

HW Occlusion Culling사용시 주의 사항

- D3DQuery – 더 느려지기 쉽상. 복잡한 shader를 쓰는 경우만 사용할것.
- Hi-Z – Occluder의 수를 제한할 경우는 거의 언제나 쓰는게 좋다.
- 어느쪽이라도 오브젝트들이 너무 많으면 효율이 극도로 나빠짐.

SW Occlusion Culling

- CPU 코드로 구현한 z-buffer
- Throughput은 HW방식에 비해 크~~~게 떨어진다. 대신 Draw Call 준비가 필요없고 응답성이 빠르다. CPU를 사용해서 z-buffer를 구현한다.
- Texturing이 없는 SW Rasterizer를 구현한다.
- GPU가 없던 시절 고대의 프로그래머들은 작은 사이즈의 buffer로 SW Occlusion Culling을 이미 사용했었다.
- 이후 SW Z-Buffer Rasterizer라 부르자.
- 물론 요새는 거의 사용하지 않는다…그러나..

SW Occlusion Culling의 재발견

KD-Tree

- X,Y,Z 축에 정렬된 BSP Tree.
- 각 node를 3가지 축과 축방향의 거리 D값으로 표현할 수 있다.
- KD-Tree를 탐색할때 카메라로부터 가까운 node와 먼 node를 찾을 수 있다.
- Ray의 교차판정 등에 많이 사용한다.
- Near node와 Far node를 구분하지 않는다면 Quad-Tree와 별 차이는 없다.
- 게임의 월드를 KD-Tree로 관리하는 경우가 많다.

KD-Tree를 이용한 오브젝트 수집

- node에 대해 view frustum culling. Frustum에 포함되지 않으면 다음 node로.
- Leaf이면 leaf가 포함하는 오브젝트 수집
- 오브젝트에 대해 view frustum culling.
- 다음 node로 진행

KD-Tree를 이용한 오브젝트 수집

- Tree 탐색중에 view frustum culling을 통과한 node라도 오브젝트에 가려져서 보이지 않을 수 있다. (사실은 이런 경우가 아주 많다.)
- Tree탐색중에 가려져서 보이지 않는 node를 바로 걸러낼 수 없나?
- Tree탐색중에 Z-buffe를 구성해가면 가능하겠네?

Occlusion Culling in KD-Tree

- node째로 culling하면 하위 node와 leaf에 포함된 오브젝트들도 같이 cullin된다.
- KD-Tree탐색 단계에서 node에 대해서 Occlusion Culling을 수행하면 렌더링될 오브젝트들을 원천적으로 줄일 수 있다.
- 가까운 node에서 먼 node로 탐색해가면서 가까운 node(leaf)의 오브젝트를 z-buffe에 test& rasterize 한다.
- KD-Tree를 사용하면 가까운 node와 먼 node를 구분해서 가까운 node부터 탐색할 수 있다.(KD-Tree Traversal)
  - 가까운 곳의 물체는 크게 보인다.
  - 따라서 최초 몇번의 z-test만으로도 상당히 많은 수의 가려지는 오브젝트들을 걸러낼 수 있다.

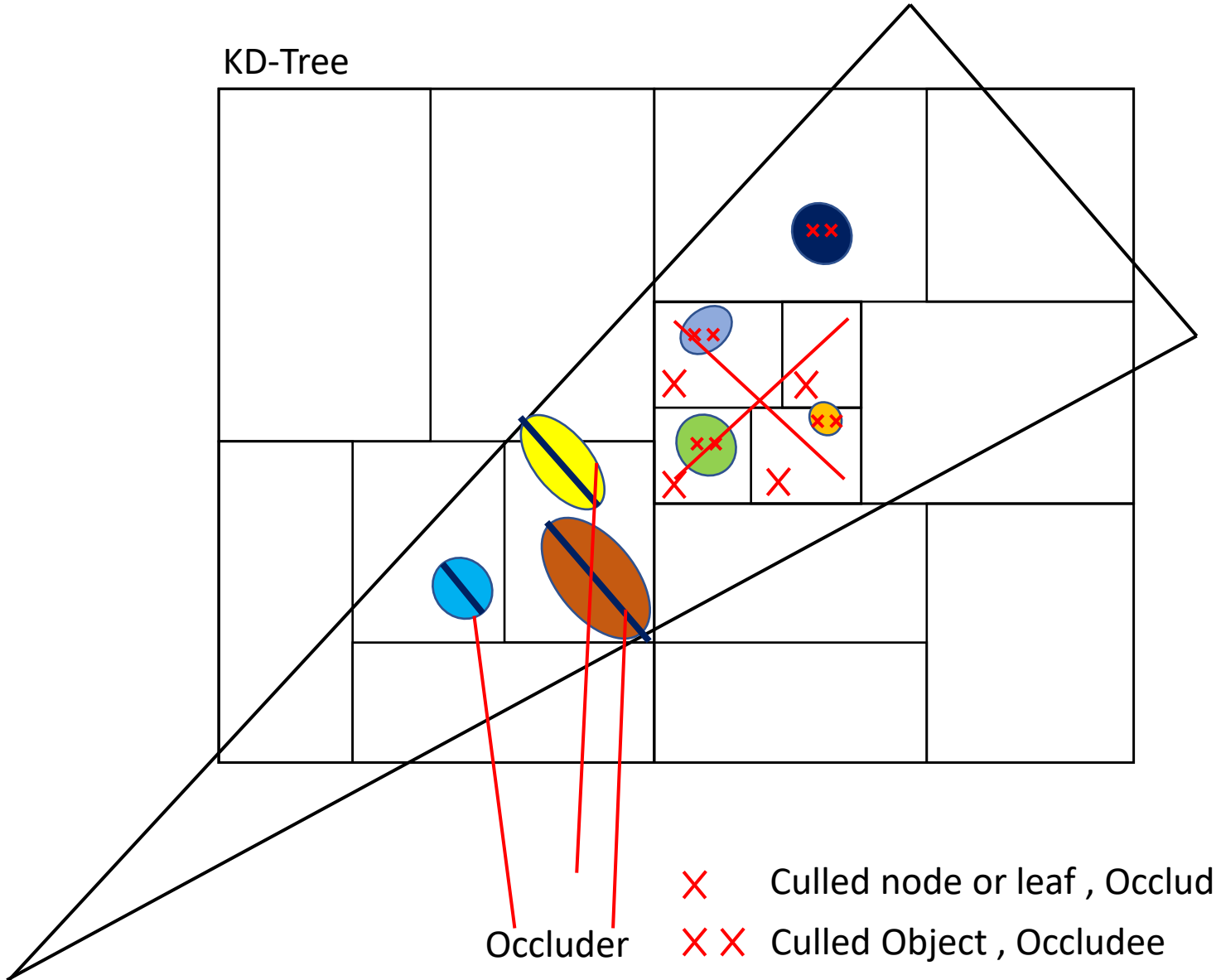
KD-Tree + SW Occlusion Culling을 이용한 오브젝트 수집

- Node에 대해 view frustum culling
- Node의 AABB-를 육면체(삼각형 12개)를 z-test. Z-test결과 렌더링 되는 픽셀이 하나도 없으면 다음 node로.
- Node에 대해 view frustum culling. Frustum에 포함되지 않으면 다음 node로.
- Leaf이면 leaf가 포함하는 오브젝트 수집
- 오브젝트에 대해 view frustum culling.
- 오브젝트에 대해 z-test & z-write수행. 렌더링되는 픽셀이 하나도 없으면 수집하지 않음.
- 다음 node로 진행

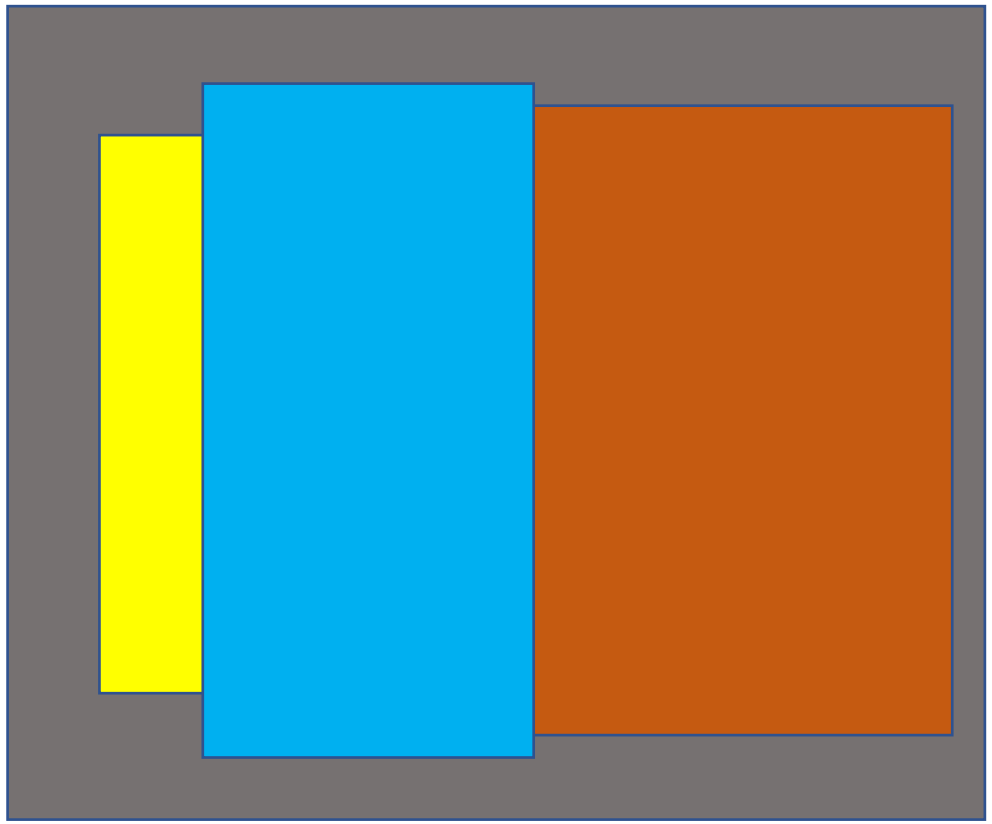
SW Occlusion Culling in KD-Tree

KD-Tree순회중에 먼저 발견한 오브젝트들(Occluder)을 z-buffer에 그린다. 이로 인해 다음번 순회할 node(or leaf)를 통째로 제외시킬수 있다.

KD-Tree



Frame Buffer or Z-Buffer



Culled node or leaf , Occludee

Culled Object , Occludee

KD-Tree + S/W Occlusion Culling

- HW Occlusion Culling은 Occluder/Occludee를 렌더링하기 위해 준비 시간이 너무 길고 GPU로부터 결과를 얻어오는데도 많은 시간이 걸린다. 따라서 KD-Tree탐색중에는 사용할 수 없다.
- 삼각형 몇십 몇백개 정도를 rasterize & test하기 때문에 throughput이 크게 중요하진 않다.
- 응답성은 아주아주아주 중요!!!
- 따라서 KD-Tree탐색중에 사용할 Occlusion Culling은 SW방식을 사용한다.

S/W Z-Buffer Rasterizer구현

CPU코드로 z-buffer에 삼각형을 rasterize하고 test하는 기능을 구현한다.


Voxel Horizon Client x64 Release D3D11 - D3D_FEATURE_LEVEL_11_1 - Oct 4 2017
f:950 obj:230 hfo:0 spr:3 font:10 P:5863 V:4663 W:0
Tex:41 VB:343 IB:324 CB:21 VL:7 A.Map:0 font:7
DEV. Version Build 1
Ping:T_T
f:950 obj:230 hfo:0
Tex:41 VB:343 IB:3
SuperYuchi
S/W Z Rasterizer
UI initialized sucessfully.
Trying load voxels from (C:₩Users₩megay₩AppData₩Local₩VoxelHorizon₩v101.vxl).

SW Z-Buffer Rasterizer요구사항

- 기본적으로 90년대 게임들의 SW Rasterizer와 크게 다르지 않다.
- Texturing은 필요하지 않음.
- 임의의 삼각형 리스트를 입력 받아 버퍼에 z값을 기록한다. GPU의 z-buffe와 기본적으로 같다.
- 화면의 left,right,top,bottom,near, far에 대해 클리핑 기능이 필요.
- Z-write & z-test, z-test only 두가지 함수를 노출한다.

SW Z-Buffer Rasterizer (Phase 1 - Transform)

1. 월드공간의 삼각형 -> N-Polygon으로 변환(클리핑시 억만조각 나는걸 막기 위해)
2. N-Polygon을 view 공간으로 변환
3. view 공간으로 변환된 N-Polygon을 near, far 평면에 대해서 절단. 여기서부터 삼각형이 아니게 된다.
4. view공간의 N-Polygon을 projection공간으로 변환 후 w로 나눠서 $-1 < x < 1, -1 < y < 1 , 0 < z < 1$ 좌표계로 변환.

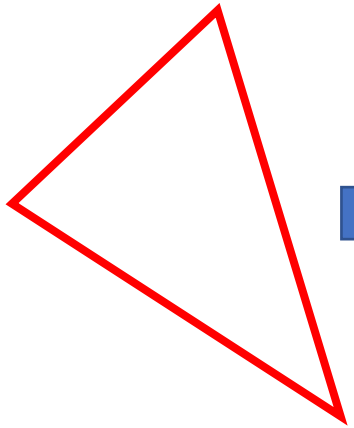
SW Z-Buffer Rasterizer (Phase 1 - Transform)

5. 정규좌표로 변환된 N-Polygon을 left,right,top,bottom 4개의 평면으로 절단. 이 과정을 거치면 3각형 -> 최대 8각형까지 증가.
6. N-Polygon -> N-2개의 삼각형으로 분리.
7. 각 삼각형을 y좌표가 같은 2개의 삼각형으로 분리.

SW Z-Buffer Rasterizer (Phase 2 – Rasterize)

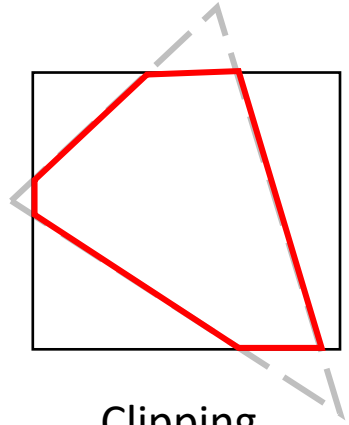
8. 삼각형의 왼쪽 오른쪽 두 변의 기울기를 구한다. 이제 y좌표가 주어질때 양쪽 변의 x0,x1좌표를 얻을 수 있다.
9. 삼각형의 윗쪽 꼭지점으로부터 한 라인씩 내려오며 x0부터 x1까지 스캔라인을 따라 z값을 보간해서 얻어온다.
10. 보간해서 얻은 z값을 스크린 버퍼로부터 얻어온 z값과 비교, 보간한 z값이 앞에 있을 경우 덮어 쓴다.

Near/far에 대해 한번, left/top/right.bottom에 한번 총 두번의 클리핑을 나눠서 하는 이유는 w값이 1이 아닌 상태에서의 클리핑이 까다롭기 때문이다. W <= 0 일 경우, 그러니까 카메라 안쪽에 넘어온 점이 있을 경우 계산 불능이 된다. 그러니까 -1 < x,y < 1의 좌표로 변환하기 전에 클리핑을 수행한다.

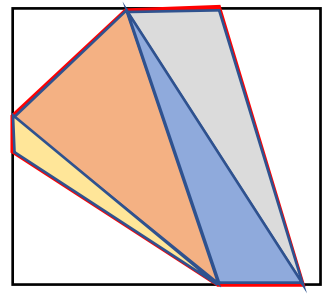


Local -> World-> Projection Matrix

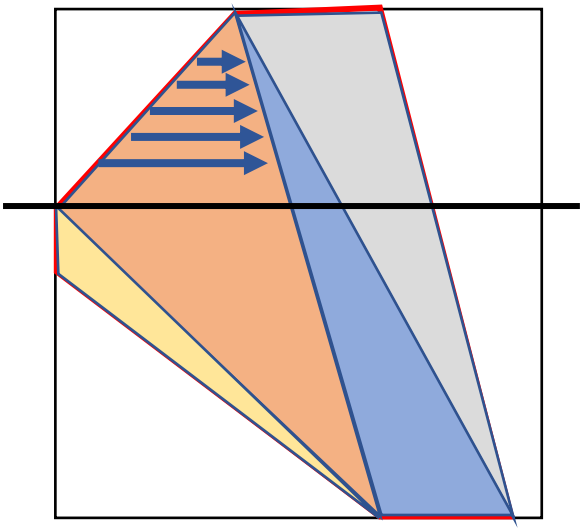
Transform with MVP Matrix

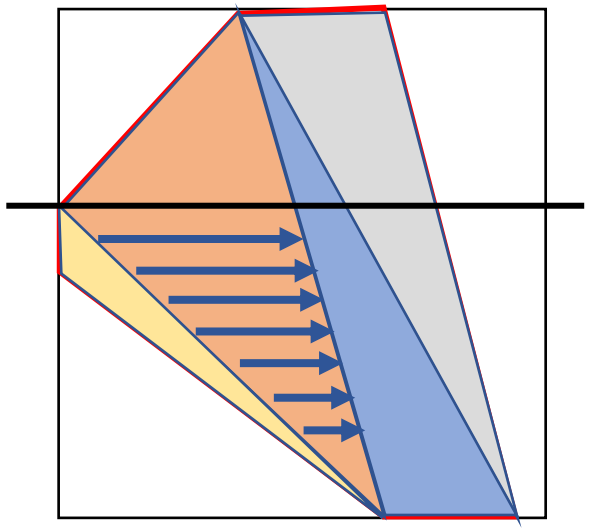


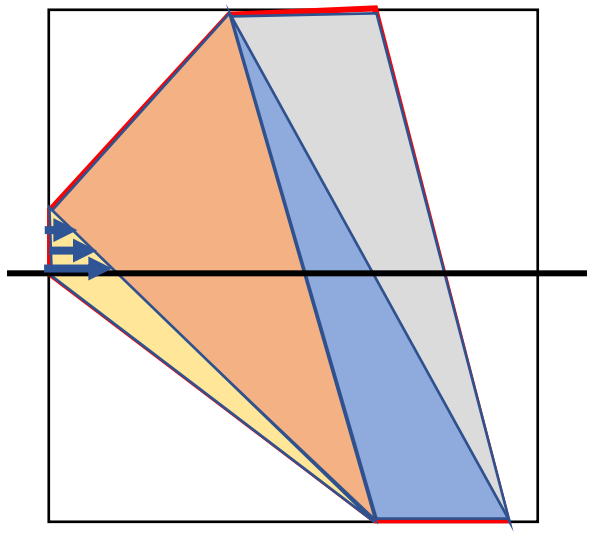
Clipping

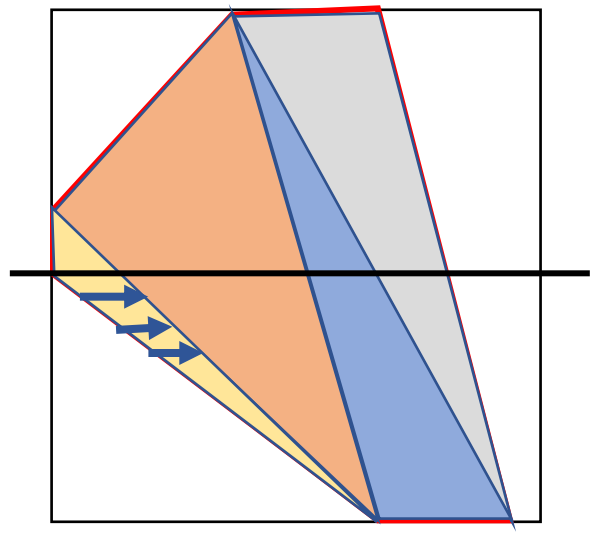


Polygon -> Triangles







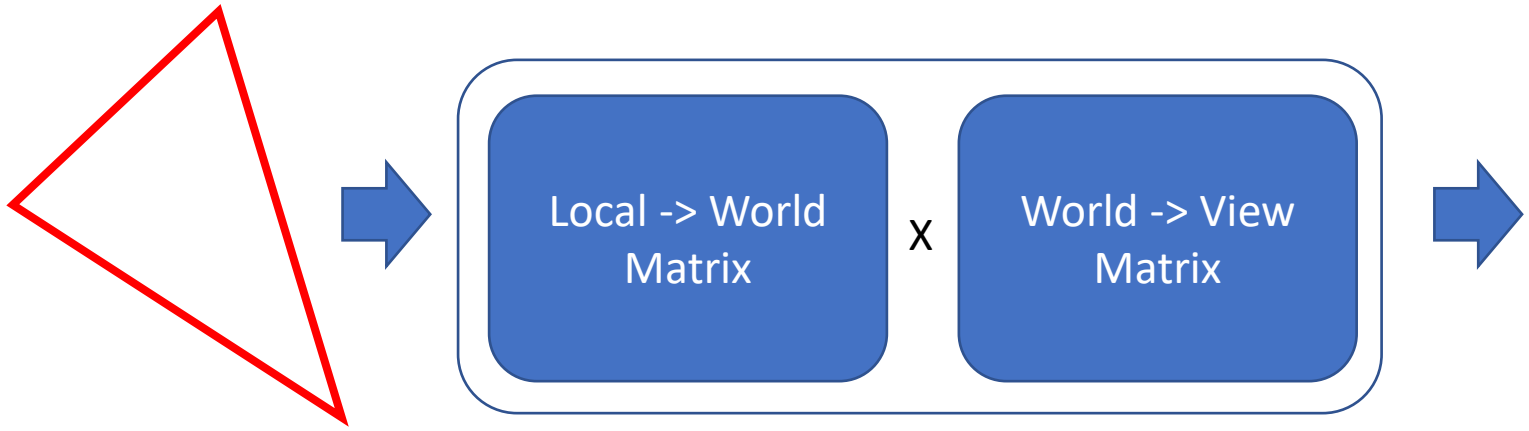


Rasterize

Transform Trinagles (phase 1)

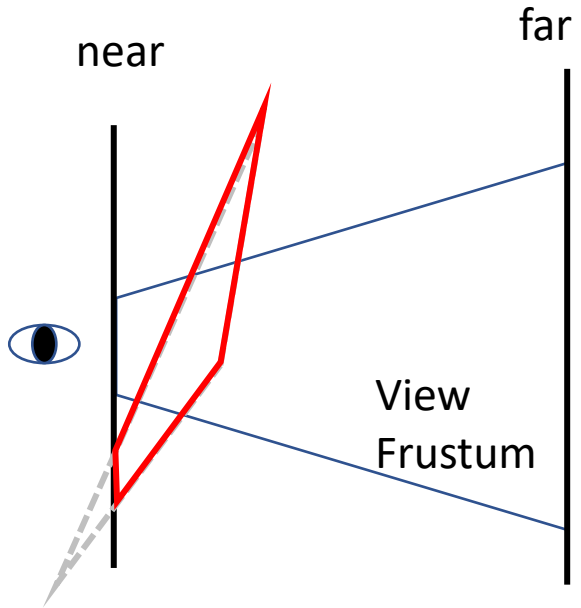
월드 공간의 삼각형을 디바이스 좌표계(-1 <= x,y <= 1)로 변환

Transform (World -> NDC)



Transform with MV Matrix

Clipping with left,top,right,bottom



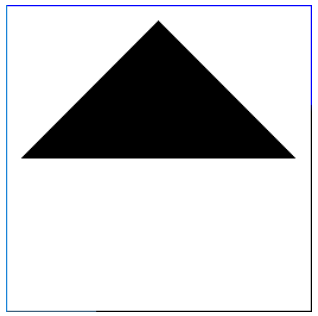
Clipping with Near/far

클리핑 코드를 단순화시키기 위해서 near/far로 클리핑 후 left/top/right/bottom으로 클리핑

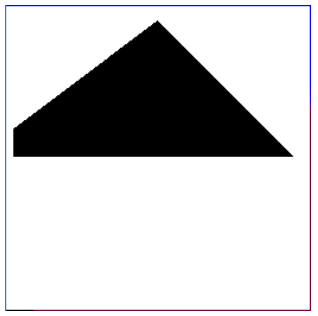
Clipping

- 삼각형에서 대해서 총 6개 면으로 클리핑.
- 삼각형을 클리핑하면 최종 삼각형 개수가 크게 늘어남.
- 따라서 N Polygon으로 변환해서 클리핑.

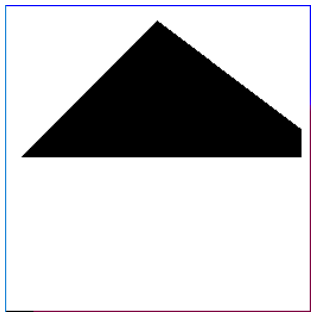
Clipping



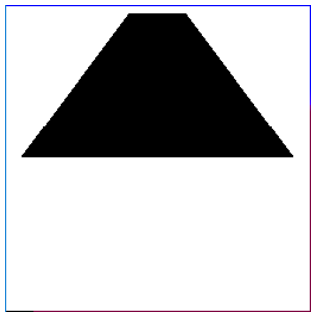
No clipping



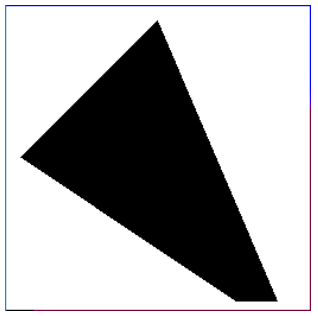
Left



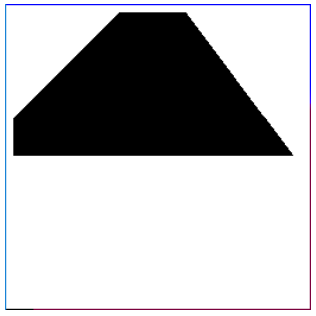
Right



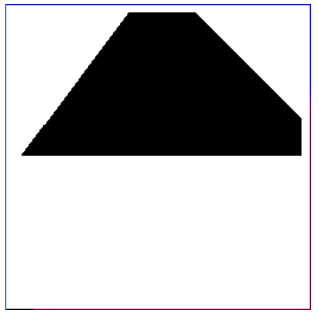
Top



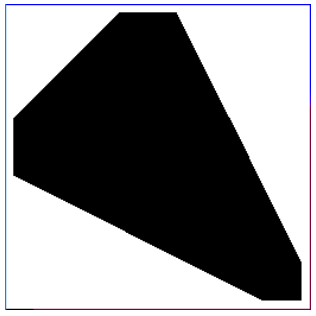
Bottom



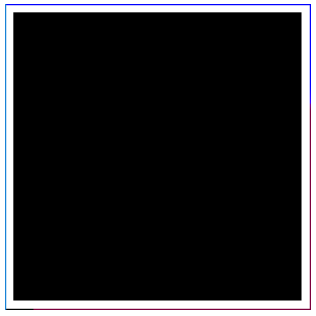
Left-top



Right-Top

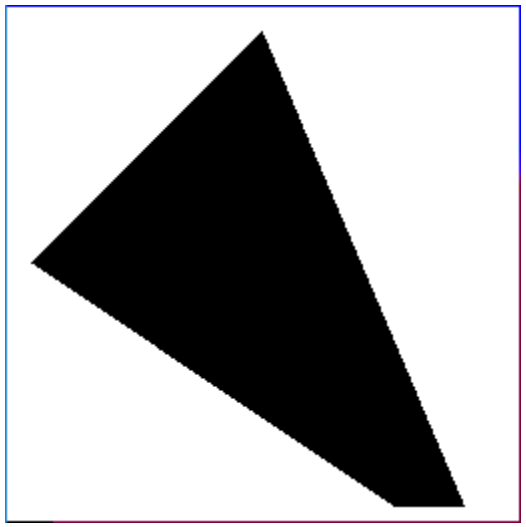


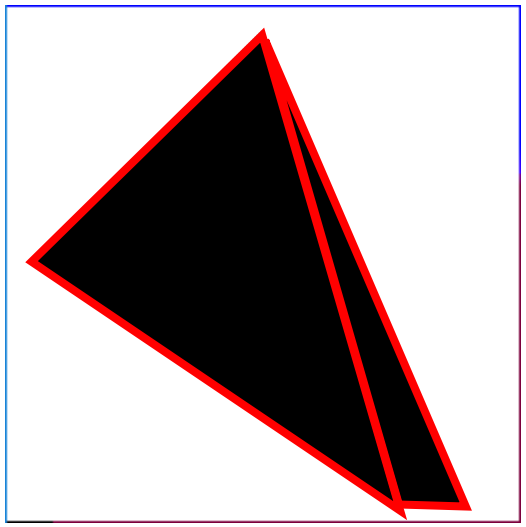
Left – Top –
Right - Bottom



Left – Top –
Right - Bottom

N Polygon -> Triangles

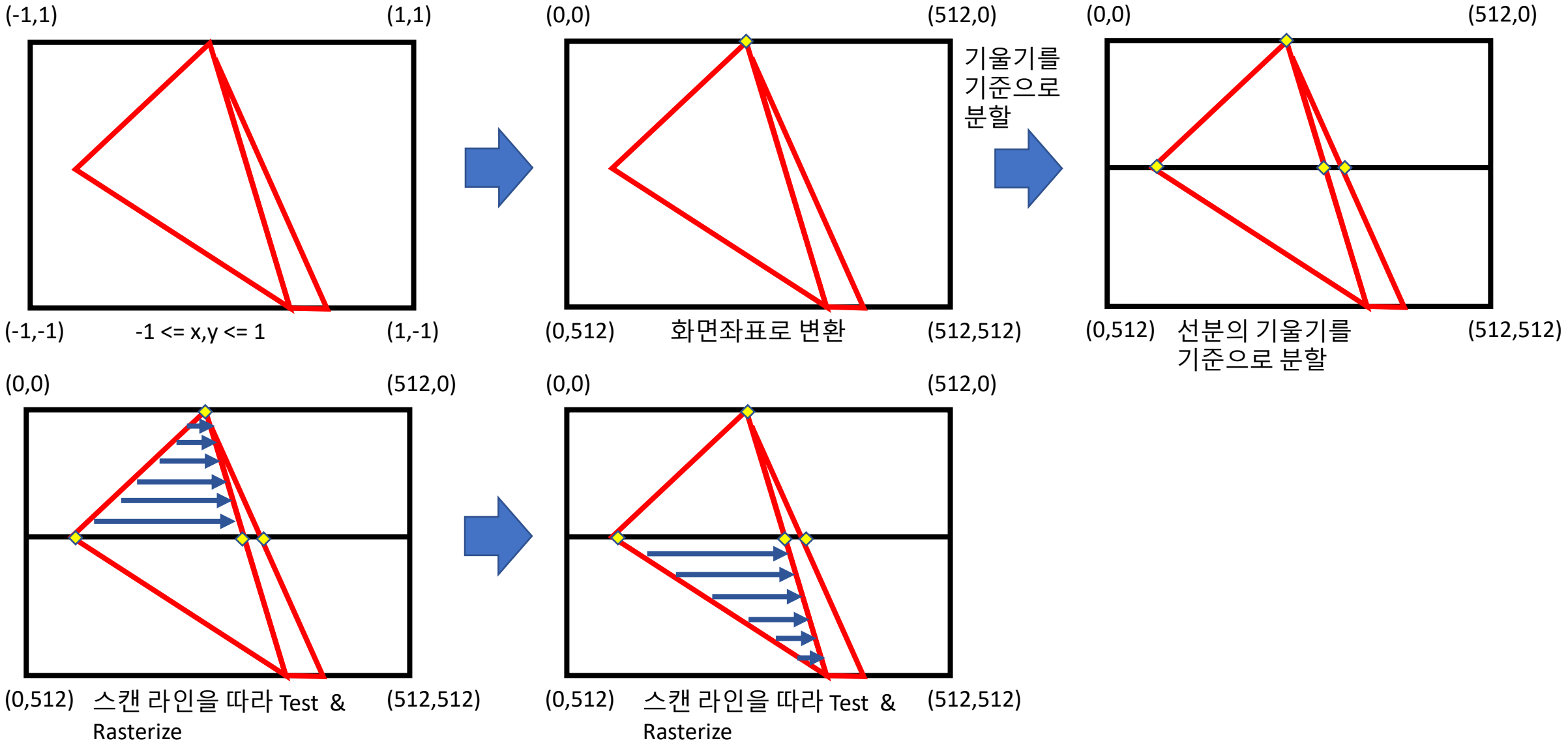




Rasterize & Test (Phase 2)

월드공간으로부터 디바이스 좌표계(-1 <= x,y <= 1)로 변환된 삼각형을 화면 좌표계의 버퍼에 rasterize.

Rasterize(or test)



최적화

- 생각보다 느리다.
- 최적화가 필요하다.

최적화

- SIMD 최적화
- Multi Thread
- 시간제한
- Tile Buffer

SIMD 최적화

- SSE (4픽셀 동시처리)
- AVX (8픽셀 동시처리)

https://megayuchi.com/2016/10/23/sse를-이용한-4샘플-무분기-치환-for-sw-occlusion-culling/

- 삼각형이 화면에서 차지하는 면적이 클수록 SIMD처리가 유리.
- 그러나 대부분의 경우 작은 삼각형의 개수가 많다.
- 실질적인 성능향상은 10%이내.
- 그나마 node를 test하는 경우는 화면상에서 차지하는 면적이 크므로 SIMD를 사용하는 쪽이 확실히 빠르다.

Multi Thread

- 삼각형을 Rasterize하거나 Test하려면 두가지 공정으로 나눌 수 있다.
  1. 세 점을 트랜스폼해서 NDC좌표계로 바꾸고 클리핑 하는 과정.
  2. 세 점으로부터 빗변의 기울기를 구하고 스캔라인 따라가며 버퍼에 그리거나 테스트하는 과정.

- 2번은 멀티스레드로 처리하려면 버퍼에 lock을 걸어야한다. 하지만 1번은 depth 버퍼를 억세스하지 않으므로 완전히 병렬로 처리할 수 있다.

- node를 방문했을때 이 node가 leaf이고 오브젝트들을 가지고 있다면 그 즉시 오브젝트들의 삼각형 리스트를 비동기로 transform(1번작업)시킨다. 이때부터 백그라운드에서 다수의 워커스레드들이 입력받은 삼각형 데이터를 클리핑된 NDC좌표 삼각형들로 바꿔나간다. 완료된 오브젝트에 대해서는 intelock변수 하나로 표시만 해둔다.
- 메인 스레드는 워커 스레드들이 완료하기를 기다리지 않고 Rasterize작업을 시작한다. 앞서 요청한 트랜스폼 작업의 interlock변수가 '완료됨'으로 표시된 경우 트랜스폼된 삼각형 데이터를 가지고 바로 버퍼에 그린다. interlock변수가 완료되지 않음으로 표시된 경우 비동기로 요청한 작업은 무시하고 메인 스레드가 직접 트랜스폼 -> 그리기 작업을 진행한다.

- 이것은 일종의 prefetch cache다. 메인스레드가 비동기로 걸어놓은 트랜스폼 작업이 때마침 완료되어서 트랜스폼을 생략할 수 있으면 hit, 완료되지 않아서 메인스레드가 트랜스폼부터 다시 하는 경우가 miss다. 현재까지 테스트한 바로는 Nehalem 아키텍처의 구형 머신에선 80%이상, 샌디브릿지 이상 i7 4코어에선 95%이상 히트한다.
- 하지만 실제 게임 처리 상황에서 큰 도움이 안되는 경우가 많다.

SW Occlusion Culling 시간제한

- SW Occlusion Culling효율이 안나오는건 사실 큰 문제는 아니다. 그보다는 SW Occlusion Culling자체로 시간을 너무 잡아먹어서 전체적인 프레임 레이트를 왕창 떨어뜨리는게 훨씬 큰 문제다.
- 따라서 최악의 상황을 막기 위해 시간제한을 건다.

SW Occlusion Culling 시간제한

- 트리 탐색중 Rasterize와 Test작업 각각에 일정 시간을 소모하면 작업을 중단한다. 즉 SW Occlusion Culling을 끄는 것이다.
- 일반적으로 Rasterize에 총시간이 더 많이 들어가므로 Rasterize와 Test 각각 다르게 시간제한을 둔다. Rasterize를 중단해도 이미 depth버퍼를 약간이라도 구성해놓은 상태이므로 Test를 더 진행할 수 있게 하기 위함이다.
- 적어도 SW Occlusion Culling때문에 느려지는 사태는 막을 수 있다.

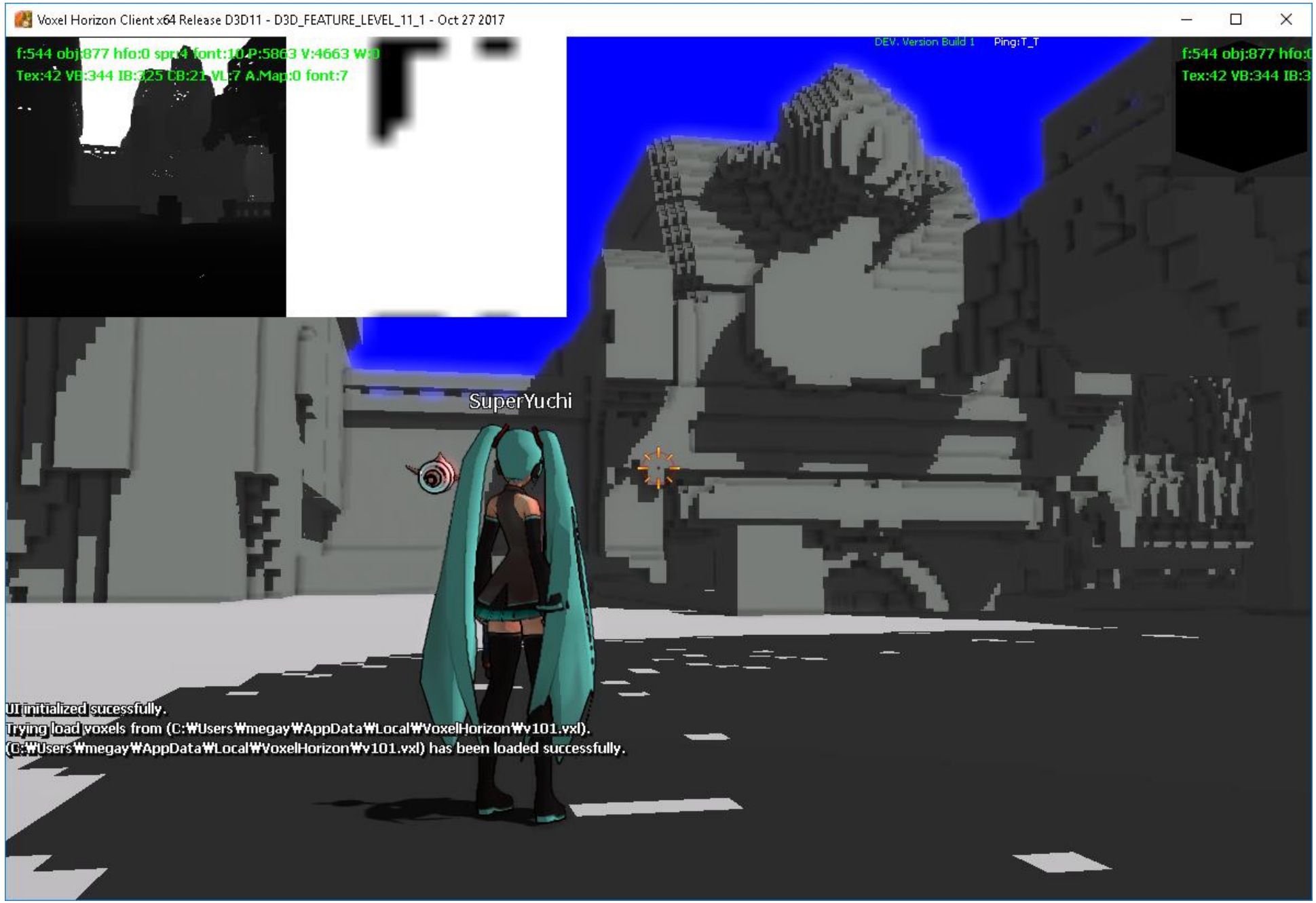
Tile Buffer

- Occluder로 사용하는 삼각형들을 충분히 그려놔야 이후 node를 구성하는 삼각형들을 Test하는 작업이 의미가 있다.
- 시간제한에 걸려서 Occluder삼각형들을 충분히 depth버퍼에 Rasterize 하지 못하는 경우가 있다.
- 시간제한에 걸려서 depth버퍼를 충분히 채우지 못한 경우, depth버퍼의 빈 영역에 test를 시도하게 된다.
- 초기에 탐색하는 node일수록 화면상에서 차지하는 사이즈가 큰데 빈 depth 버퍼에 대고 깊이 테스트하는 것은 굉장한 낭비다.
- 불필요한 z-test로 더 느려지는 결과를 초래할 수 있다.

Tile Buffer

- 16×16짜리 타일버퍼를 만든다. 화면을 16×16으로 나눈것이다.
- depth 버퍼 클리어할때 타일버퍼도 0으로 클리어한다.
- Occluder 삼각형을 rasterize할때 트랜스폼후 NDC좌표계 나오면 이 삼각형이 차지하는 타일버퍼 영역에 1을 써넣는다.
- 이후 node를 테스트할때 node의 삼각형이 차지하는 타일버퍼의 값을 읽어서 0이면 depth test까지 가지 않고 그대로 true를 리턴한다. depth 버퍼가 비어있으니 이 노드를 가리는 삼각형은 당연히 전혀 존재하지 않고, 이 node는 무조건 그려진다.
- 불필요한 test를 건너뛰는만큼 남는 시간에 정말 가려질 가능성이 있는 node를 몇 개라도 더 테스트할 수 있다.

시간제한을 걸지 않았을때 depth버퍼와 타일버퍼. 검정색 영역이 depth값이 존재하지 않는 영역이다.

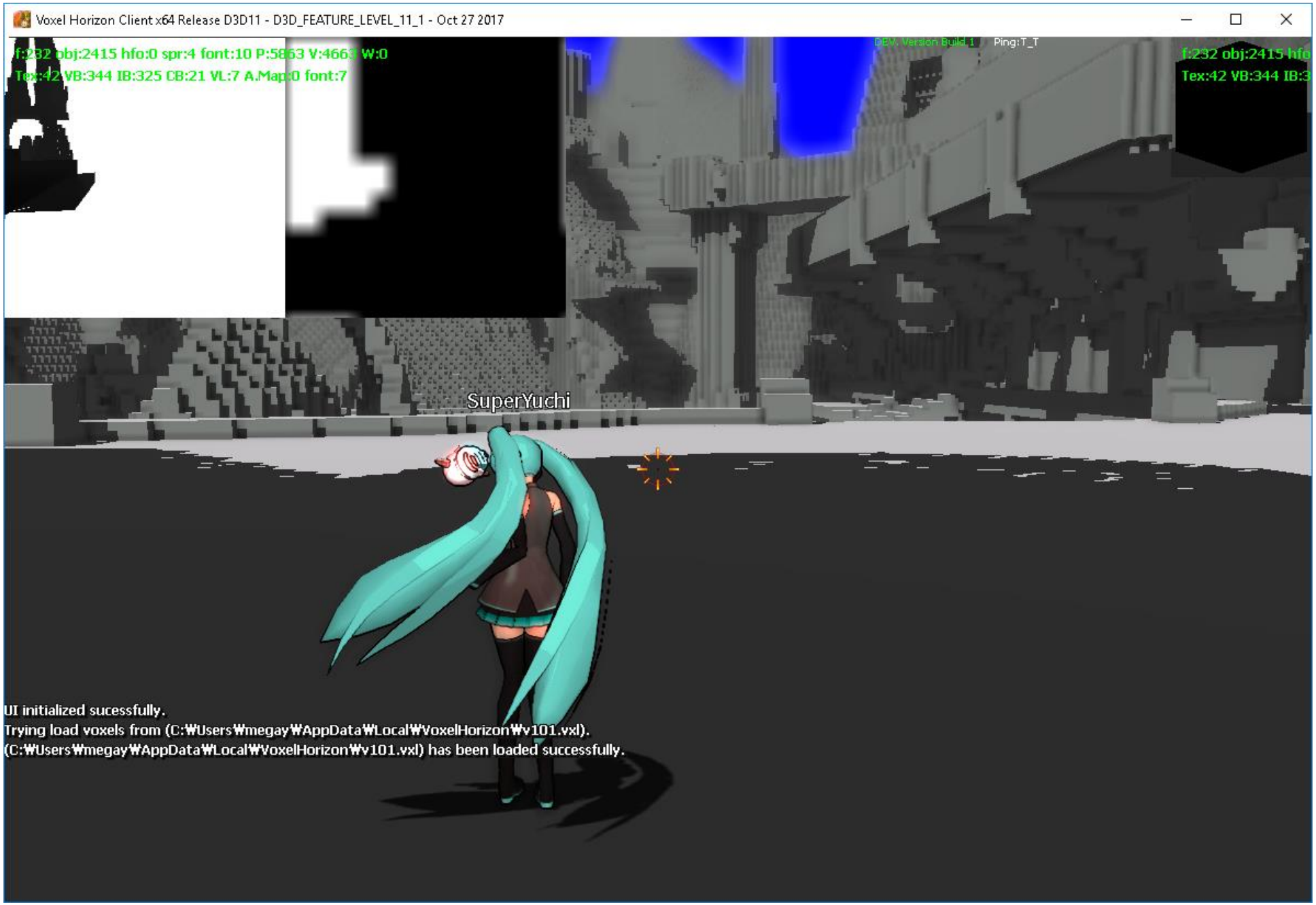


시간제한을 걸었을때의 depth버퍼와 타일버퍼.

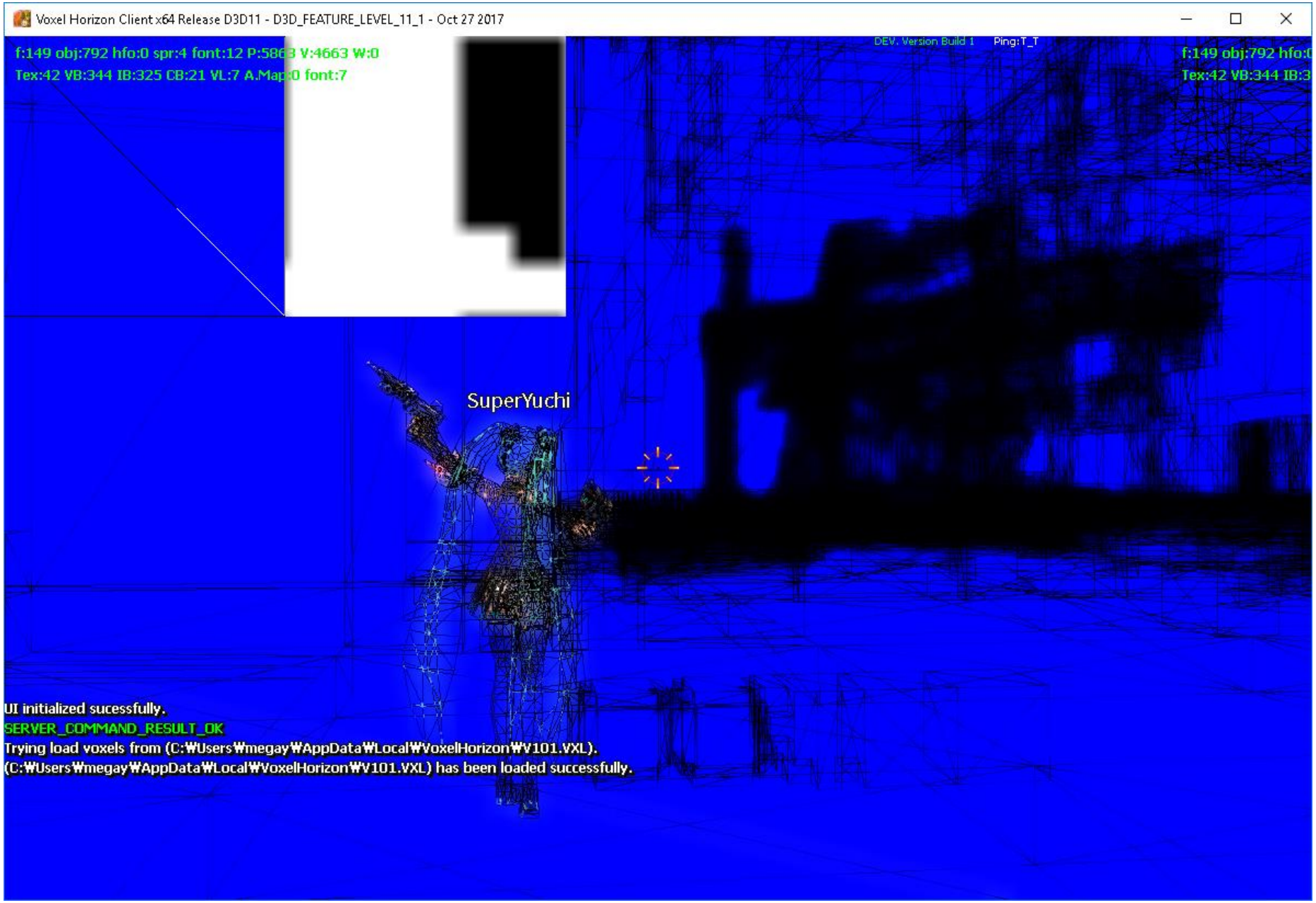
시간제한에 걸려서 충분히 Occluder를 그리지 못했고 그래서 빈 공간이 많다.

타일버퍼에서 이에 해당하는 영역이 0으로 채워져 있고 검정색으로 표시되고 있다.

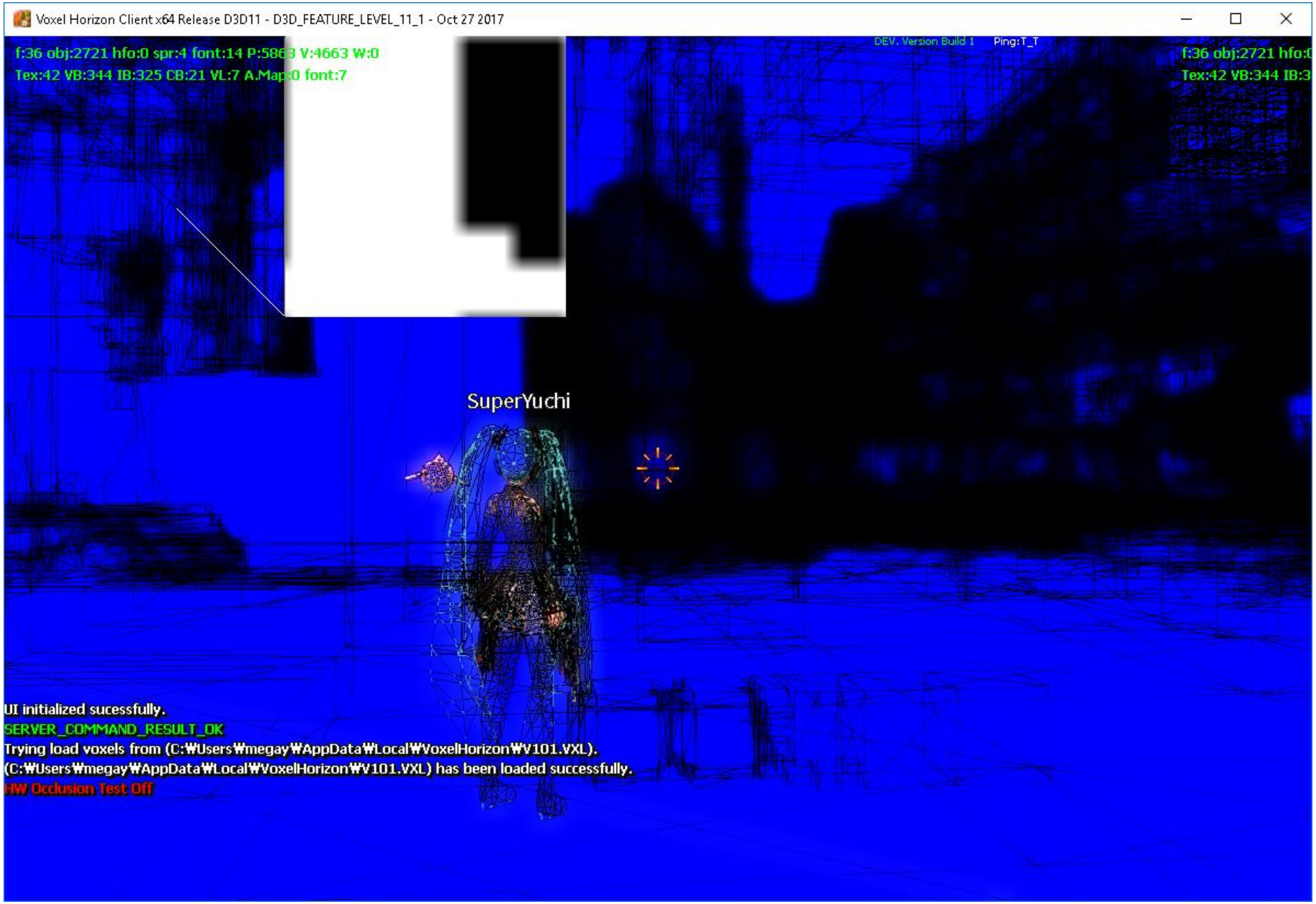
이 영역에 대해서 node의 삼각형들을 트랜스폼했을때 이 영역에 들어간다면 test를 수행하지 않고 화면에 렌더링될 node로 간주한다.



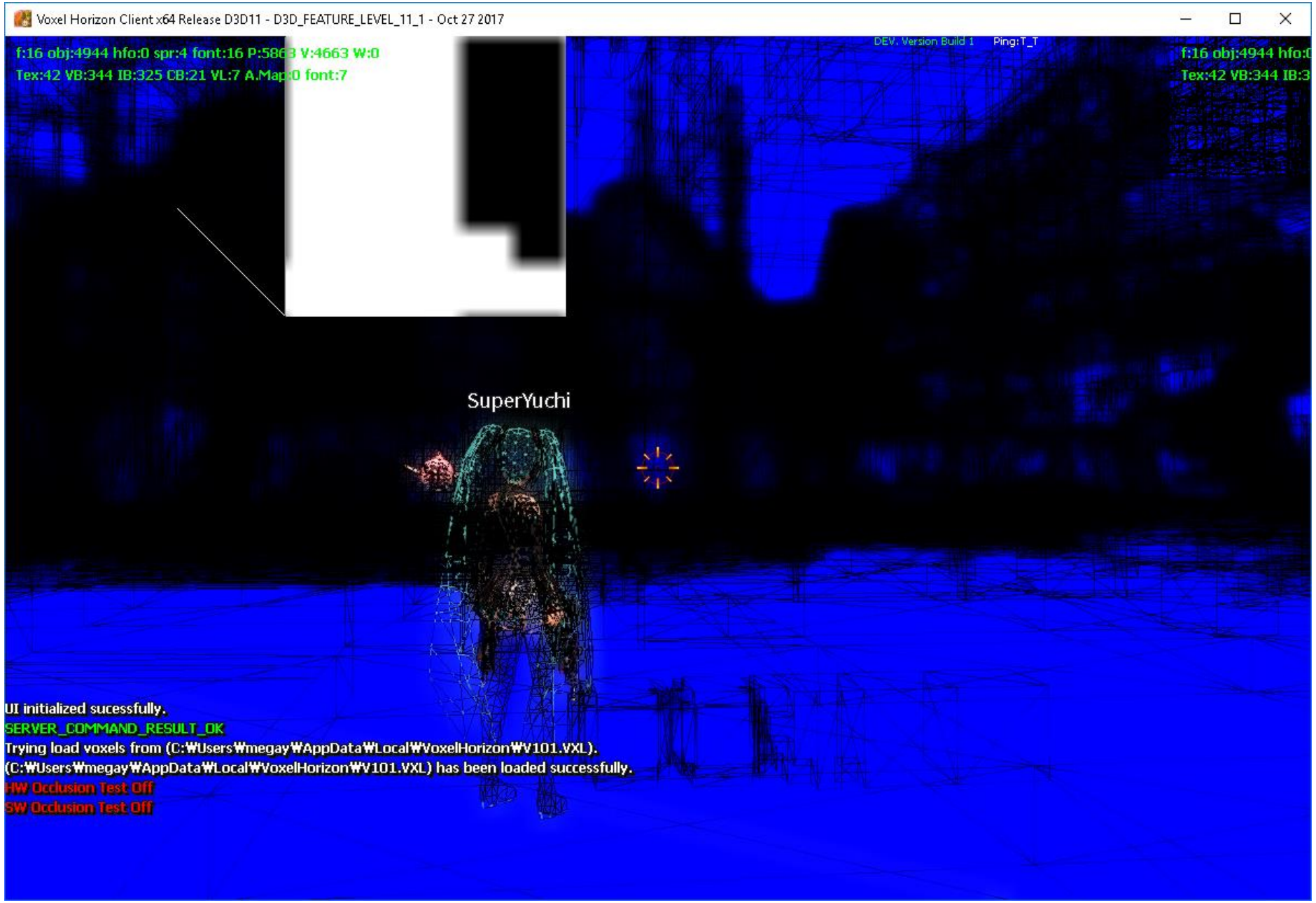
HW Occlusion Culling on / SW Occlusion Culling on (rasterize 2ms, test 1ms 제한)



Only SW Occlusion Culling (rasterize 2ms, test 1ms 제한)



HW Occlusion Culling off / SW Occlusion Culling off (rasterize 2ms, test 1ms 제한)



추후 연구해볼 과제

- Simd 추가 적용 – avx512
- 다양 멀티 스레드 방식 시도
- GPU상에서의 KD-Tree + SW Occlusion Culling
- VoxeI기반이 아니어도 적용 가능.
  - 오브젝트를 공간의 잘린 면에 맞춰서 잘라내거나
  - SW Occlusion Culling에 사용할 오브젝트와 면들을 미리 선별해주거나
  - 원 오브젝트로부터 voxeI데이터 생성
- 어차피 SW Occlusion Culling에 사용할 시간을 제한하면 되므로 적용해서 문제될것은 없다.

참고자료

- https://megayuchi.com/2016/07/24/engine-dev-sw-occlusion-culling/
- https://megayuchi.com/2016/08/01/engine-dev-clipping-triangle-for-sw-occlusion-culling/
- https://www.slideshare.net/dgtman/hierachical-z-map-occlusion-culling
- https://megayuchi.com/2017/10/27/5068/
- https://megayuchi.com/2016/10/23/sse를-이용한-4샘플-무분기-치환-for-sw-occlusion-culling/