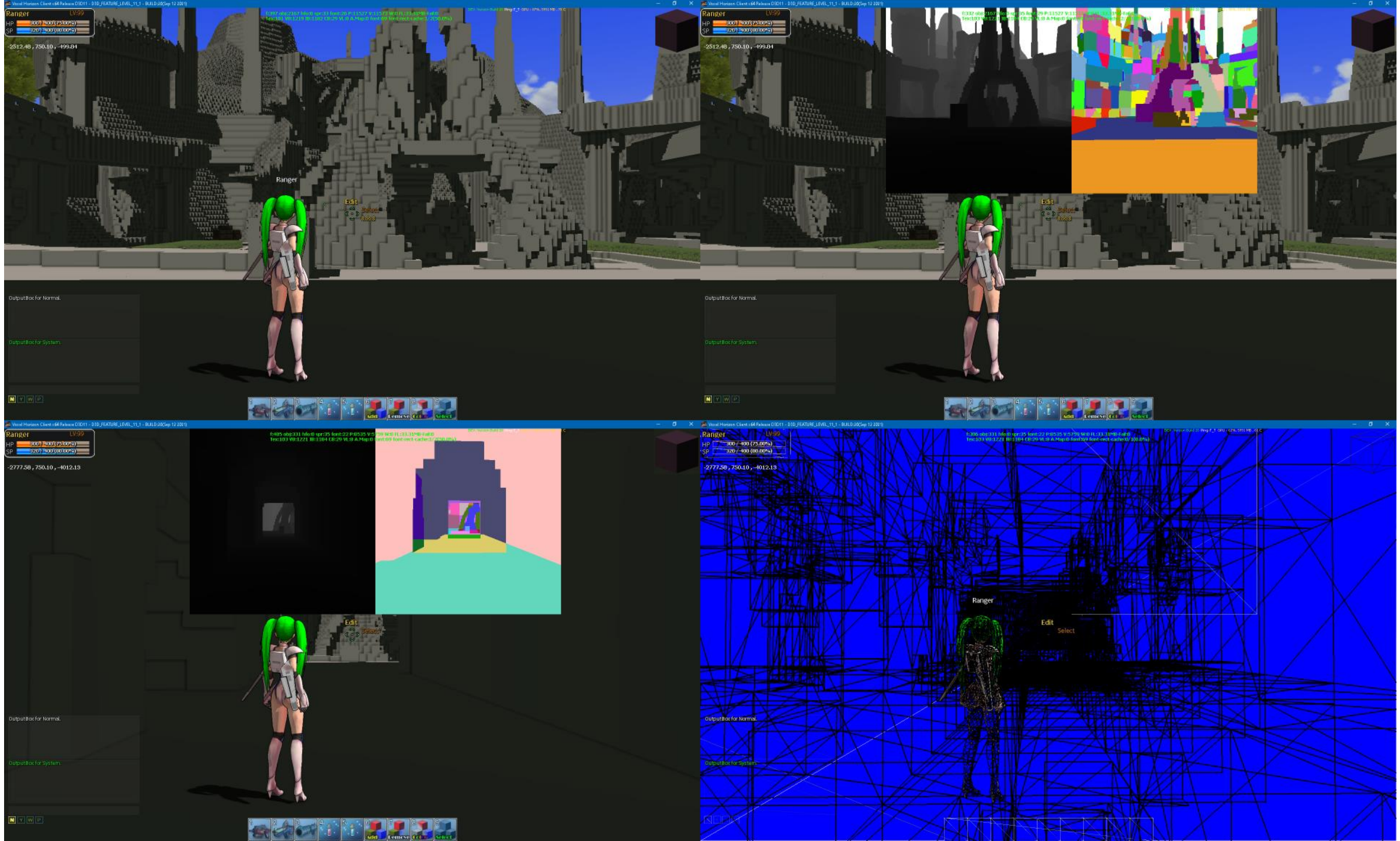
CUDA Raytracing을 이용한 voxel 오브젝트 가시성 검사

유영천

https://megayuchi.com

Tw: @dgtman

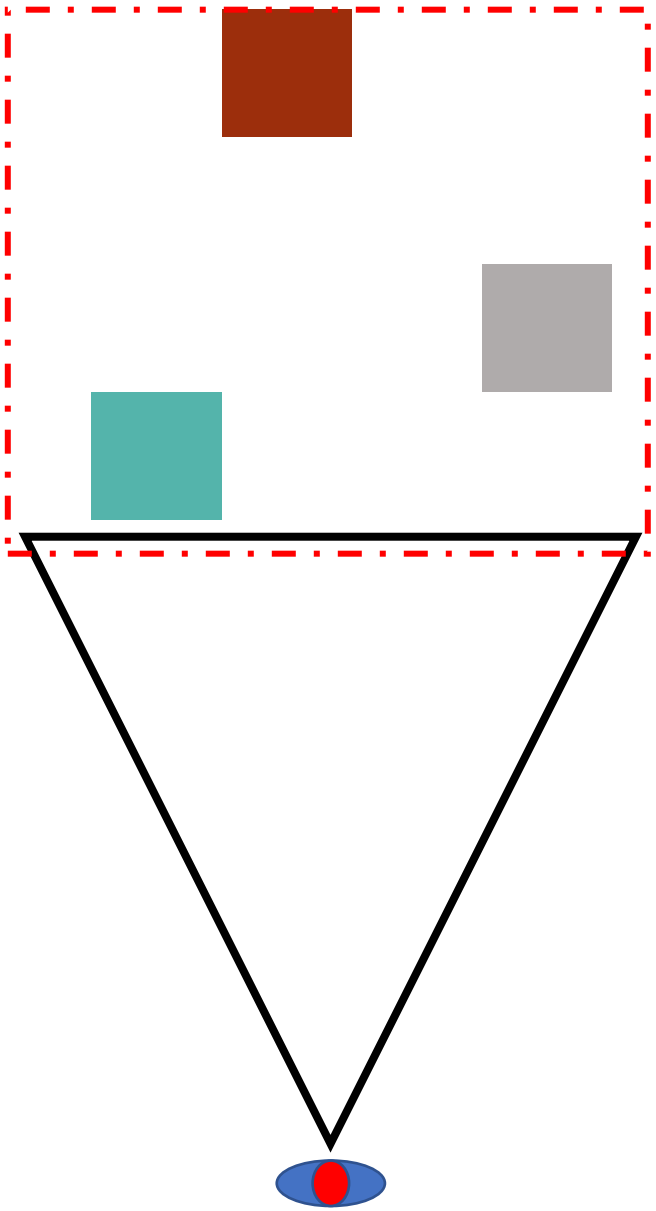

Ranger
Edit
Select
OutputBox for Normal.
OutputBox for System.

도형을 화면에 표시하는 방법

- Rasterization
- Ray Tracing


Voxel Horizon Client x64 Debug D3D11 - D3D_FEATURE_LEVEL_11_1 - BUILD:20(Sep 13 2021)
Ranger
LV:99
HP
300 / 400 (75.00%)
SP
320 / 400 (80.00%)
f:1055 obj:7 hfo:0 spr:33 font:23 P:0 V:0 W:0 FL:15.63MB-Fail:0
Tex:68 VB:1079 IB:1078 CB:29 VL:7 A.Map:0 font:15 font-rect-cache:1/1(100.0%)
[CULL_TYPE_CUDA_RAYTRACING], (-968.06, -2099.90, -4549.21)
8/8
10/10
RIFL
OutputBox for Normal.
'Ranger' used 'SMG3'.
'Ranger' used 'Rifle7'.
'Ranger' used 'SMG3'.
'Ranger' used 'Rifle7'.
N Y W P
1
2
3
4
5
6
Add
7
Remove
8
Color
9
Select

Rasterization



X

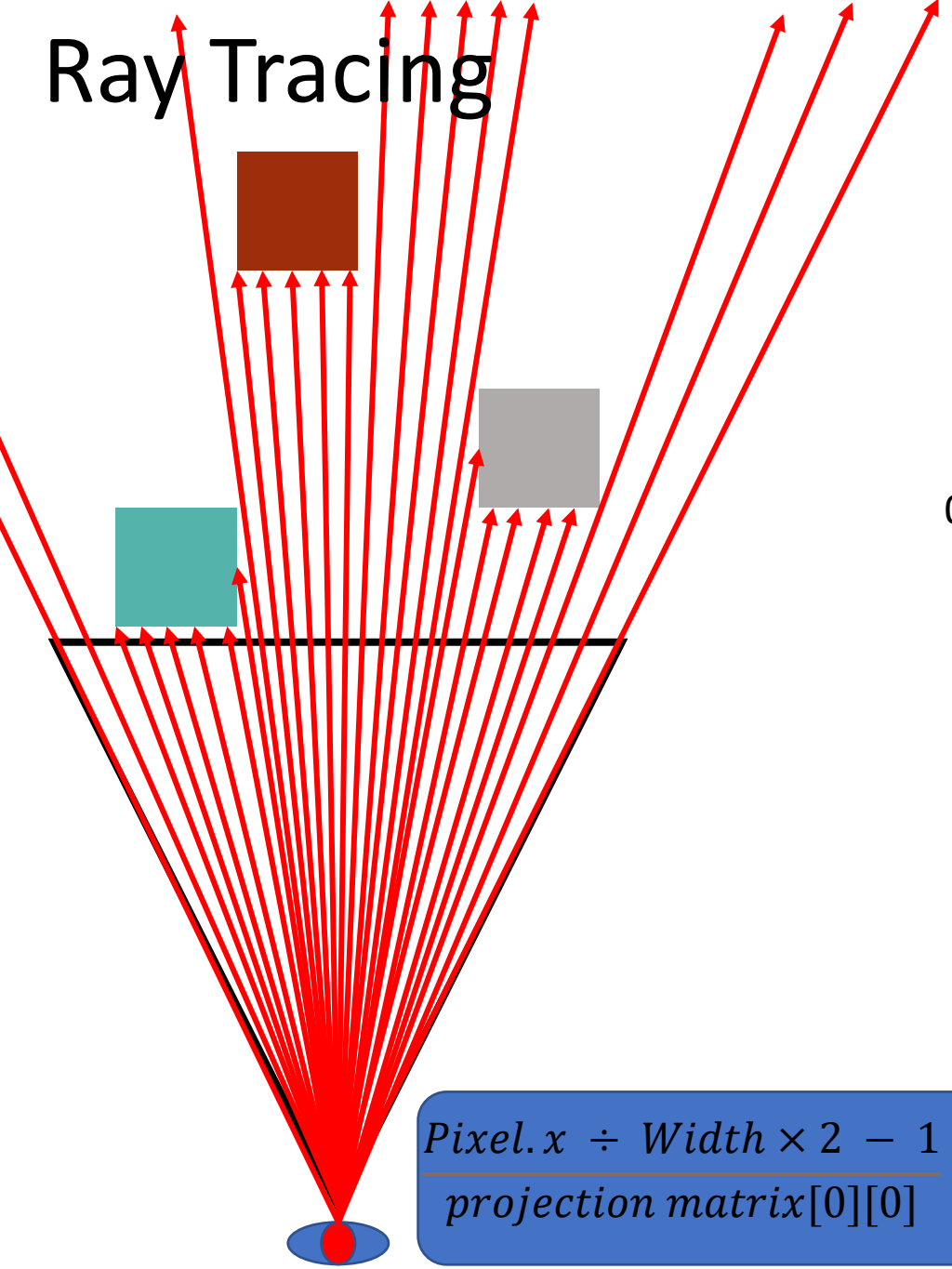
View Matrix

X

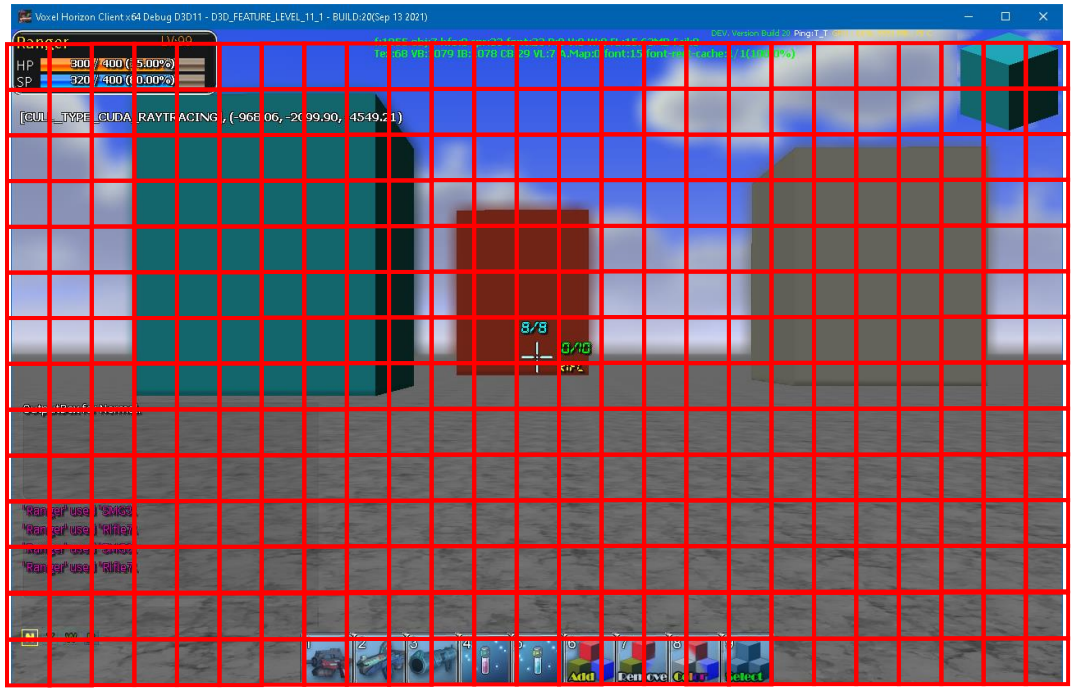
Projection Matrix

->

Ray Tracing



0



$$\frac{Pixel.x \div Width \times 2 - 1}{projection\ matrix[0][0]}, \frac{-(Pixel.y \div Height \times 2 - 1)}{projection\ matrix[1][1]}, 1, 0$$

x

$$View\ \text{matrix}^{-1}$$

가장 앞의 픽셀을 그리는 방법

Rasterization

- 기본적으로 모든 매시는 순서 없이 그린다.
- Z-buffer를 이용해서 가장 앞에 있는 픽셀로 덮어쓴다.
- 삼각형->WVP 변환 후 얻은 z값을 z-buffer에 기록. 현재 Z값이 기존 z값보다 작으면 기존 픽셀을 덮어쓴다.

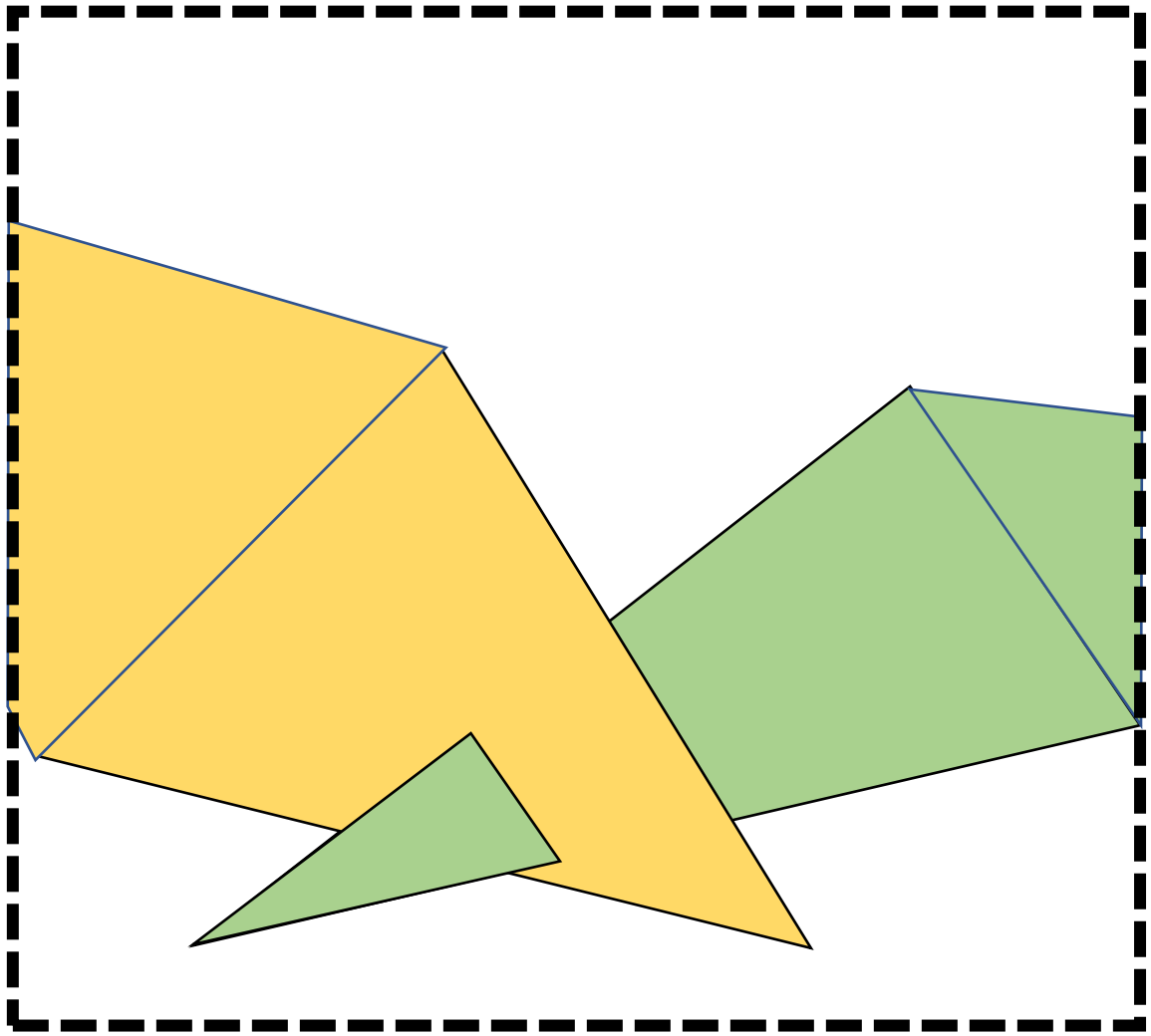
Ray Tracing

- depth버퍼 없다.
- Ray가 발사됐을 때 Ray마다 가장 앞에 있는 픽셀을 찾아낸다.

Rasterization

NDC

(-1.0, 1.0) (1.0, 1.0)



(-1.0, -1.0) (1.0, -1.0)

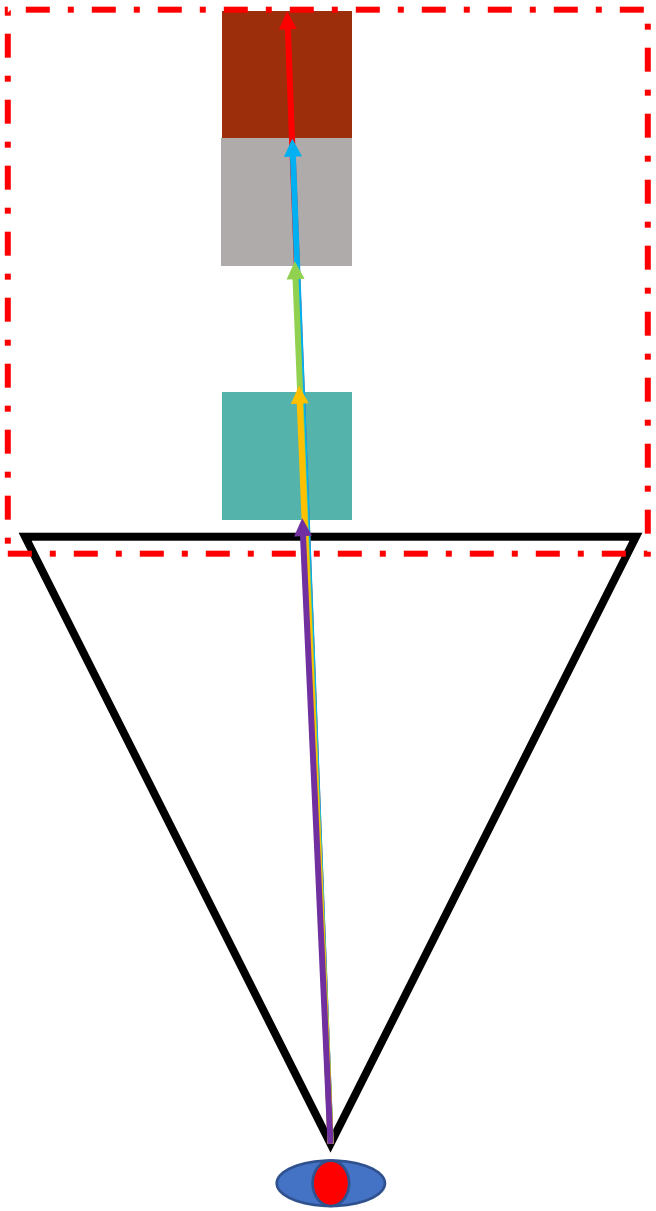
Z-Buffer

(0, 0) (Width, 0)

1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.45	0.45	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
0.44	0.44	0.45	0.46	0.46	1.0	1.0	1.0	1.0	1.0	1.0
0.46	0.45	0.46	0.46	0.45	1.0	1.0	1.0	0.48	0.48	0.48
0.46	0.46	0.46	0.45	0.44	1.0	1.0	0.47	0.48	0.48	0.48
0.47	0.46	0.46	0.45	0.44	0.43	0.46	0.47	0.47	0.48	0.48
0.47	0.46	0.45	0.44	0.43	0.43	0.46	0.46	0.47	0.48	0.48
1.0	1.0	0.42	0.43	0.43	0.43	0.43	0.45	0.46	1.0	1.0
1.0	1.0	0.41	0.42	0.43	0.43	0.42	0.41	1.0	1.0	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

(0, Height) (Width, Height)

Ray tracing에서 어느 오브젝트, 어느 삼각형이 앞에 있는지 어떻게 알지?



쉬운방법

- Ray 하나당 모든 오브젝트를 순회한다.
- Ray가 많아지면 당연히 미친듯이 느려진다.

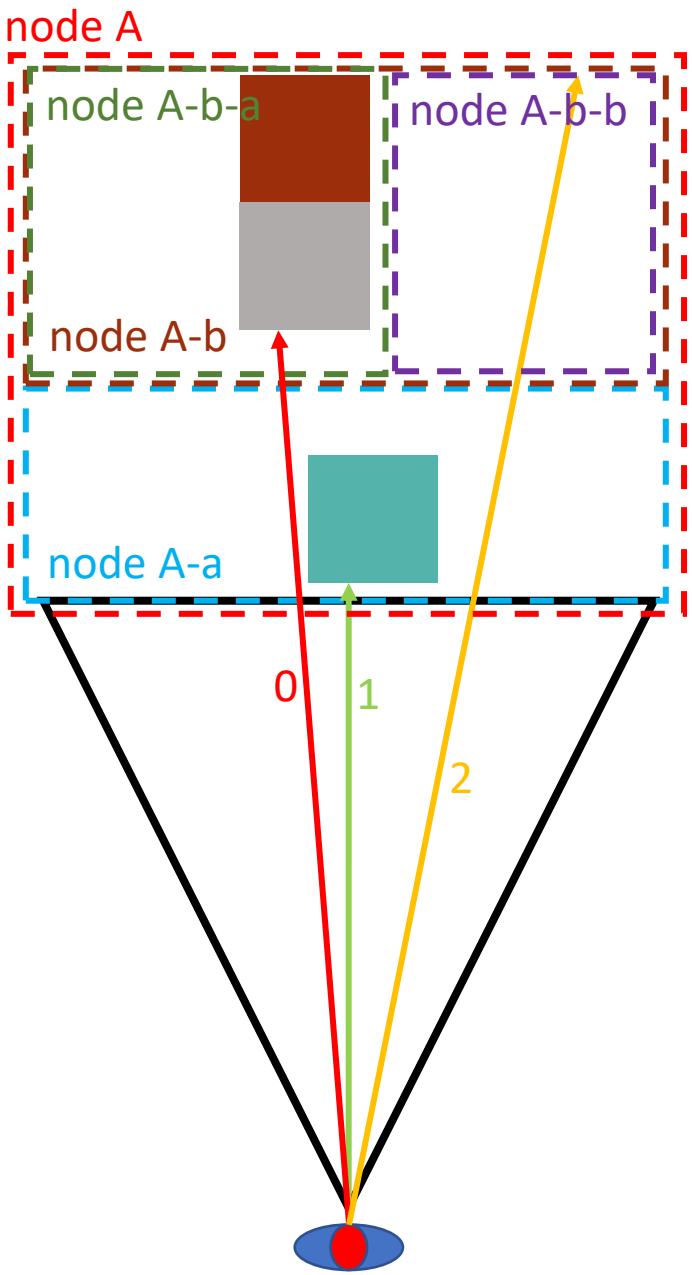
KD-Tree

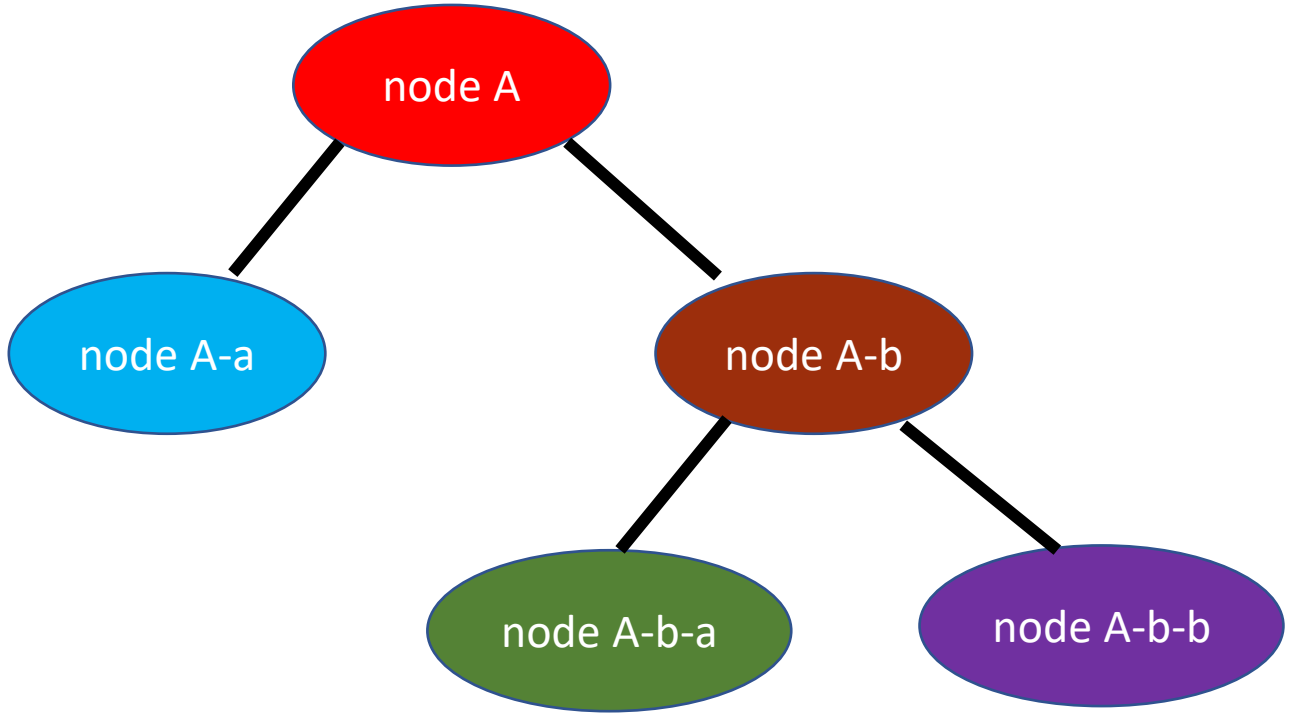
- X,Y,Z 축에 정렬된 BSP Tree.
- 각 node를 3가지 축과 축방향의 거리 D값으로 표현할 수 있다.
- KD-Tree를 탐색할 때 카메라로부터 가까운 node와 먼 node를 찾을 수 있다.
- Ray의 교차판정 등에 많이 사용한다.
- Near node와 Far node를 구분하지 않는다면 Quad-Tree와 별 차이는 없다.
- 게임의 월드를 KD-Tree로 관리하는 경우가 많다.

KD-Tree를 이용한 가장 가까운 오브젝트 탐색

- root node가 ray와 충돌하지 않으면 순회 중지.
- 현재 node의 자식 node 2개중 한 개만 ray와 충돌할 경우 충돌하는 노드를 방문.
- 현재 node의 자식 node 2개가 모두 충돌할 경우 먼 node를 stack에 push하고 가까운 node를 방문.
- 방문한 node가 leaf일 때 leaf에 들어있는 매시의 삼각형들에 대해서 ray와 충돌체크. 현재 leaf에서 충돌한 경우 t값을 저장하고 순회 중지.
- 방문한 node가 leaf일 때 어떤 삼각형도 ray와 충돌하지 않는 경우 stack에서 node포인터를 꺼내서 방문.

KD-Tree Traversal





가시성 검사에 응용

S/W Occlusion Culling

- 적당히 작은 사이즈의 버퍼에 최대한 간소화된 방법으로 매시를 그려본다. 이 과정에서 얻은 depth값을 버퍼에 기록.
- 오브젝트 또는 트리의 node/leaf를 depth버퍼에 대해 z-test. 렌더링 파이프라인에 넣기 전에 이 단계에서 가려지는 오브젝트를 걸러낸다.
- 이것을 GPU에서 처리하면 H/W Occlusion Culling.
- S/W방식(CPU에서)으로 처리하는 이유는 GPU와의 통신시간을 트리 탐색중 바로 적용할 수 있기 때문.

복셀 월드에서의 S/W Occlusion Culling

- 복셀 지형 특성상 안이 꽉 차 있고 모양이 복잡하다.
- 복셀 오브젝트간 가리고 가려지는 상황이 엄청 빈번하다.
- S/W Occlusion Culling을 시도했을때 depth를 써넣고 테스트 하는 비용이 너무 크다.
- 복셀 오브젝트 개수만 해도 5-10만개씩 되기 때문에 이미 수집한 5-10만개의 오브젝트에서 보이지 않는 오브젝트를 제거할때도 생각보다 CPU시간을 많이 소모한다.
- S/W Occlusion Culling에 적합한가?

Ray Tracing을 이용한 가시성 검사

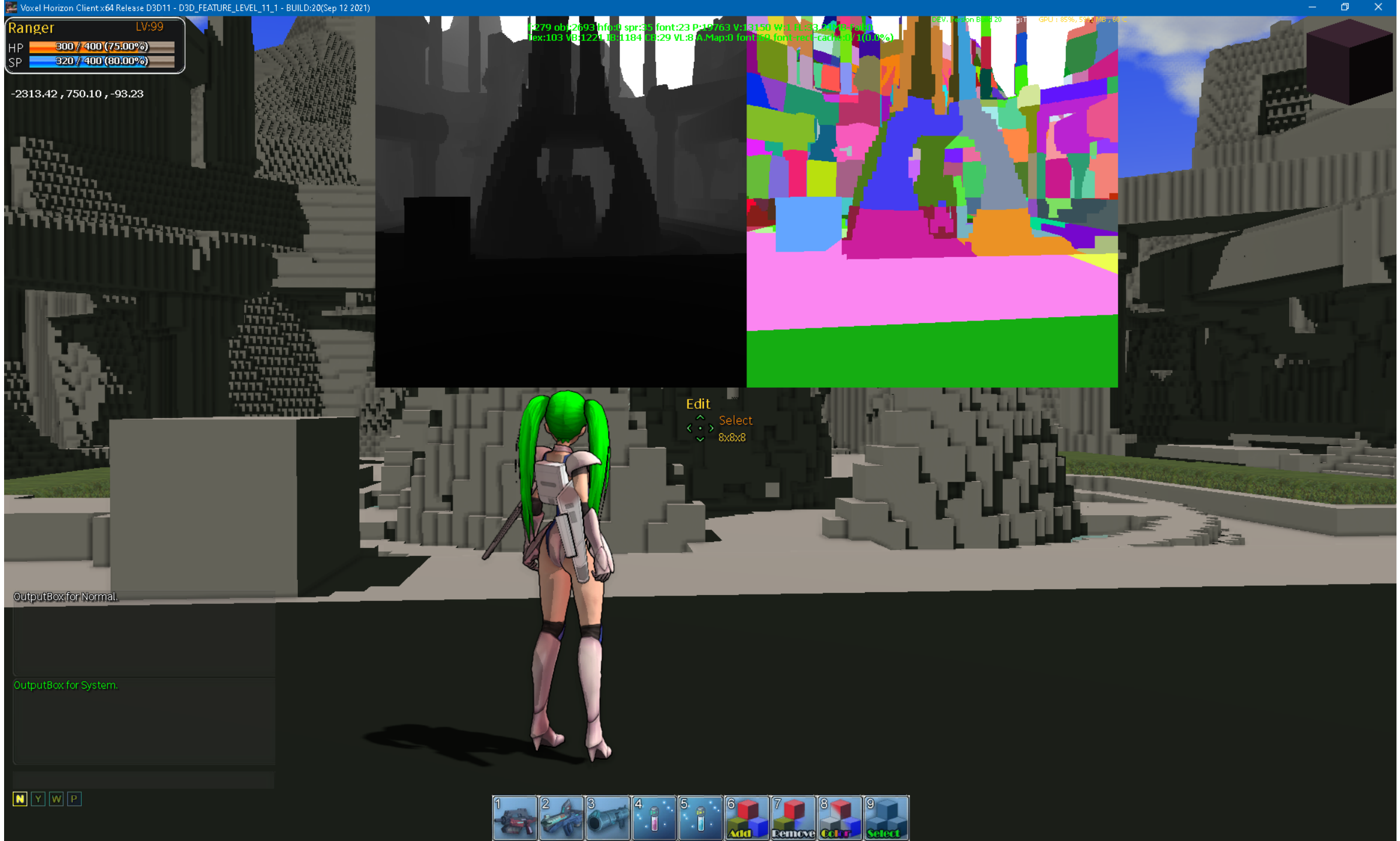
- Ray Tracing에선 처음부터 가려지는 픽셀-오브젝트는 그릴 일이 없다.
- Z-test도 필요없다.
- Ray를 쏴서 충돌하는 픽셀의 컬러값 대신 충돌한 지형지물의 오브젝트 ID 또는 leaf의 ID를 기록하면 정확히 보이는 오브젝트 목록만 얻을 수 있다.

Ray Tracing을 이용한 가시성 검사-복셀월드

- 복셀 오브젝트당 삼각형 개수는 비교적 적은 편이다.
- 변경/삭제가 빈번하지 않다.
- 복셀 오브젝트 변경시 트리를 재구축할 필요없이 leaf의 삼각형만 갱신해주면 된다.
- S/W Occlusion Culling으로 보이지 않는 오브젝트를 제외시키기보다 Ray Tracing을 이용해서 보이는 오브젝트만을 골라내는 편이 더 효율적이다.

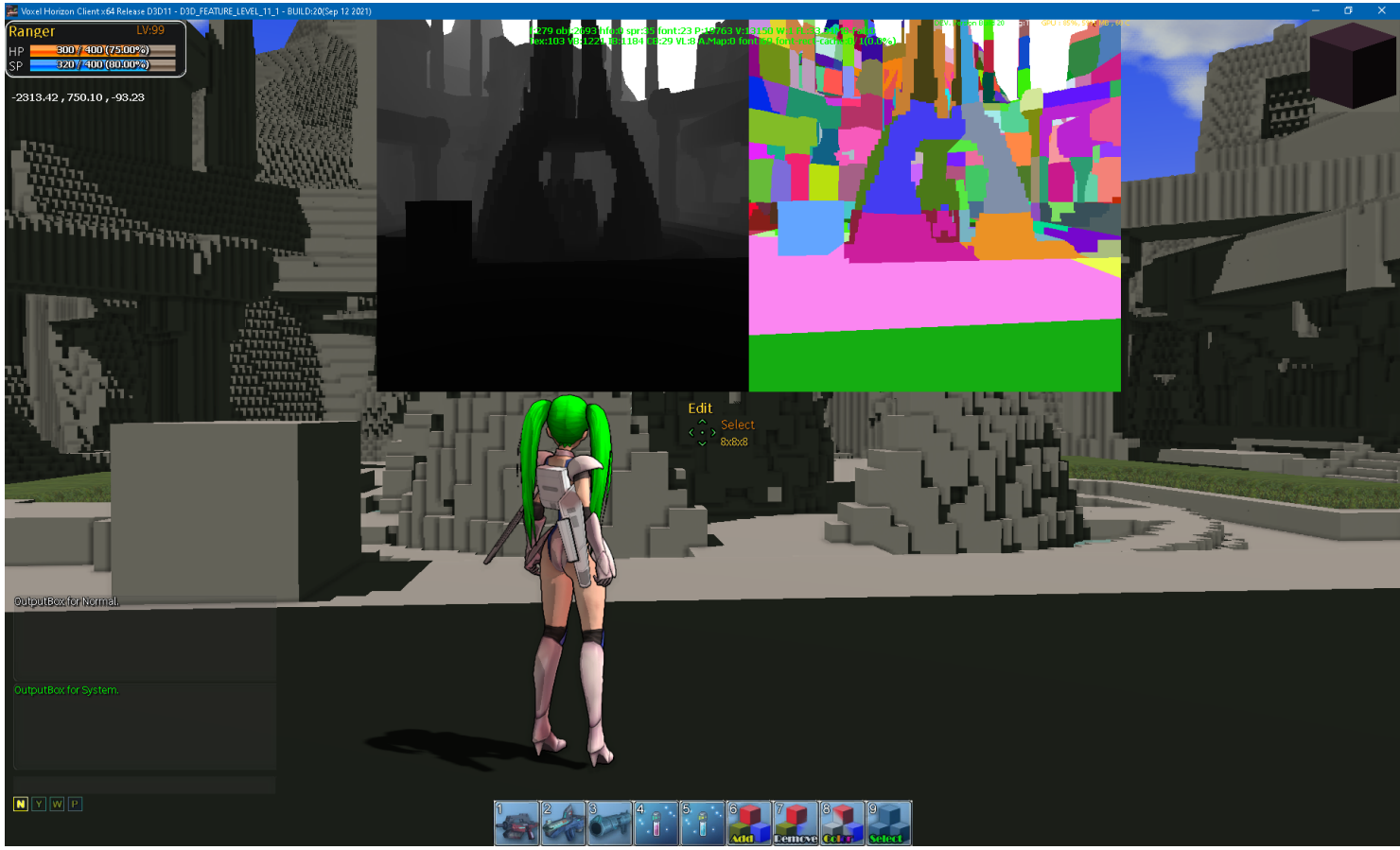
Ray Tracing을 이용한 복셀월드의 가시성 검사

1. 메모리에 적당한 사이즈의 화면 버퍼를 만든다(ex: 512×512)
2. view matrix, projection matrix를 이용해서 각 픽셀로부터 월드공간의 ray를 만든다.
3. 각 픽셀에서 출발한 ray로 복셀 오브젝트들로 구성된 KD-Tree를 순회하며 ray가 충돌하는 가장 가까운 오브젝트를 찾는다.
4. 오브젝트를 찾으면 화면 버퍼에 오브젝트의 ID를 기록한다.(테스트 시에는 depth값-ray가 충돌했을 때의 t값)
5. 버퍼의 기록된 오브젝트 ID를 가지고 렌더링 될 복셀 오브젝트 목록을 만든다.


Voxel Horizon Client x64 Release D3D11 - D3D_FEATURE_LEVEL_11_1 - BUILD:20(Sep 12 2021)
Ranger
LV:99
HP 300 / 400 (75.00%)
SP 320 / 400 (80.00%)
-2313.42 , 750.10 , -93.23
Edit
Select
8x8x8
OutputBox for Normal.
OutputBox for System.
N Y W P
1
2
3
4
5
6 Add
7 Remove
8 Color
9 Select

CPU vs GPU

- CPU로 구현하면 안되나?
- CPU로 구현해보자.



[i7 8700k 6 cores – 12 threads]

Single Thread - 2408.62 ms
Multi Thread – 12 thread(HT enabled) - 353.59 ms

60프레임을 위해서는 16ms를 초과하면 안된다! CPU로는 무리!

CUDA로 구현

CUDA를 이용한 구현

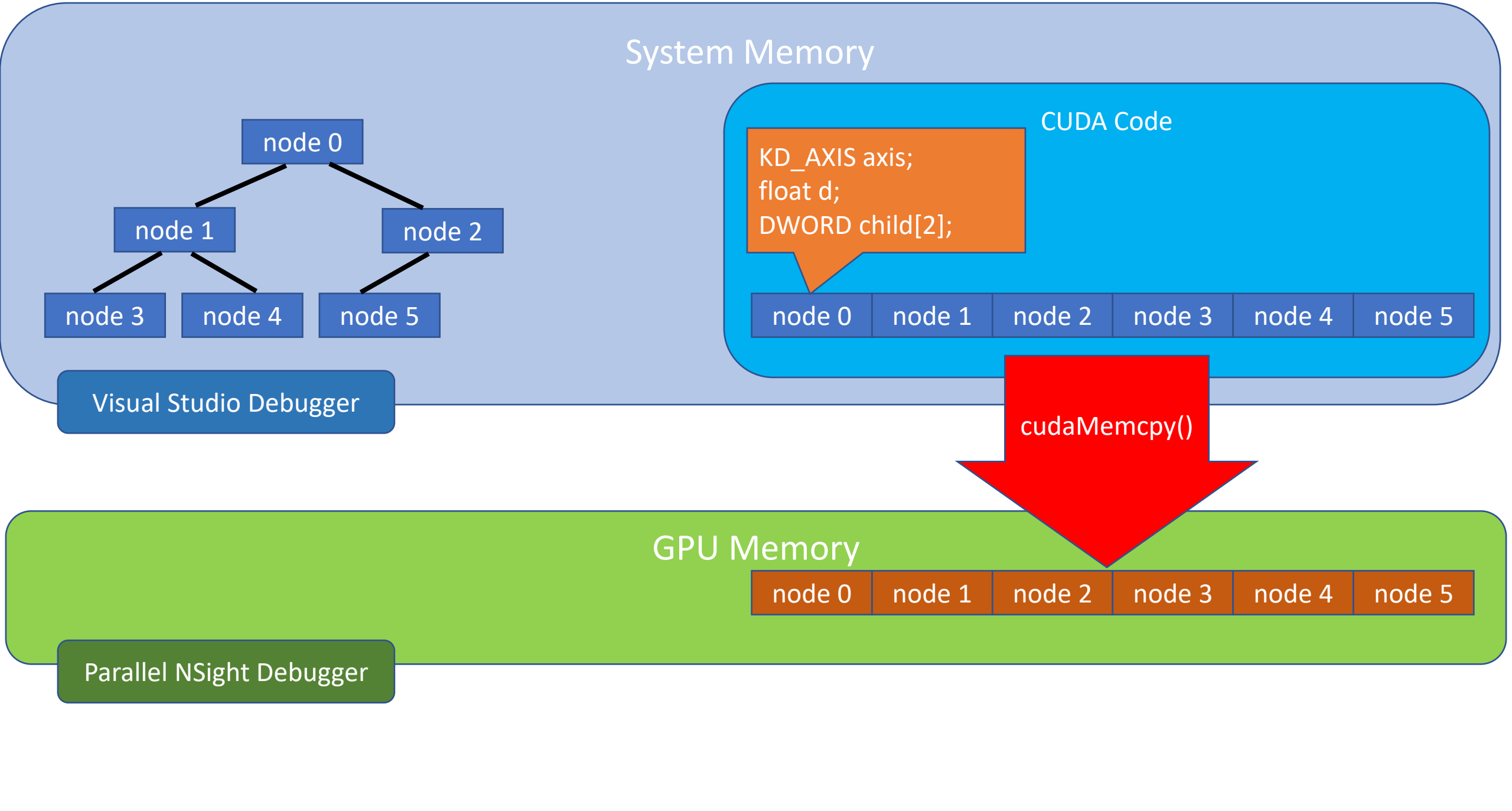
- CPU로는 너무 느리다. GPU를 사용하자.
- DirectX Raytracing API를 사용하면 간단하게 구현할 수 있다.
  그러나…
- GTX 900시리즈 같은 DXR 지원 안하는 GPU에서도 성능 향상을
  보고 싶다.
- 그래픽 API를 사용할 수 없는 환경에서도 이 기능을 사용하고 싶다.
  예를 들면 서버라든가.
- 그렇다면 CUDA를 사용한다.

기본 구현

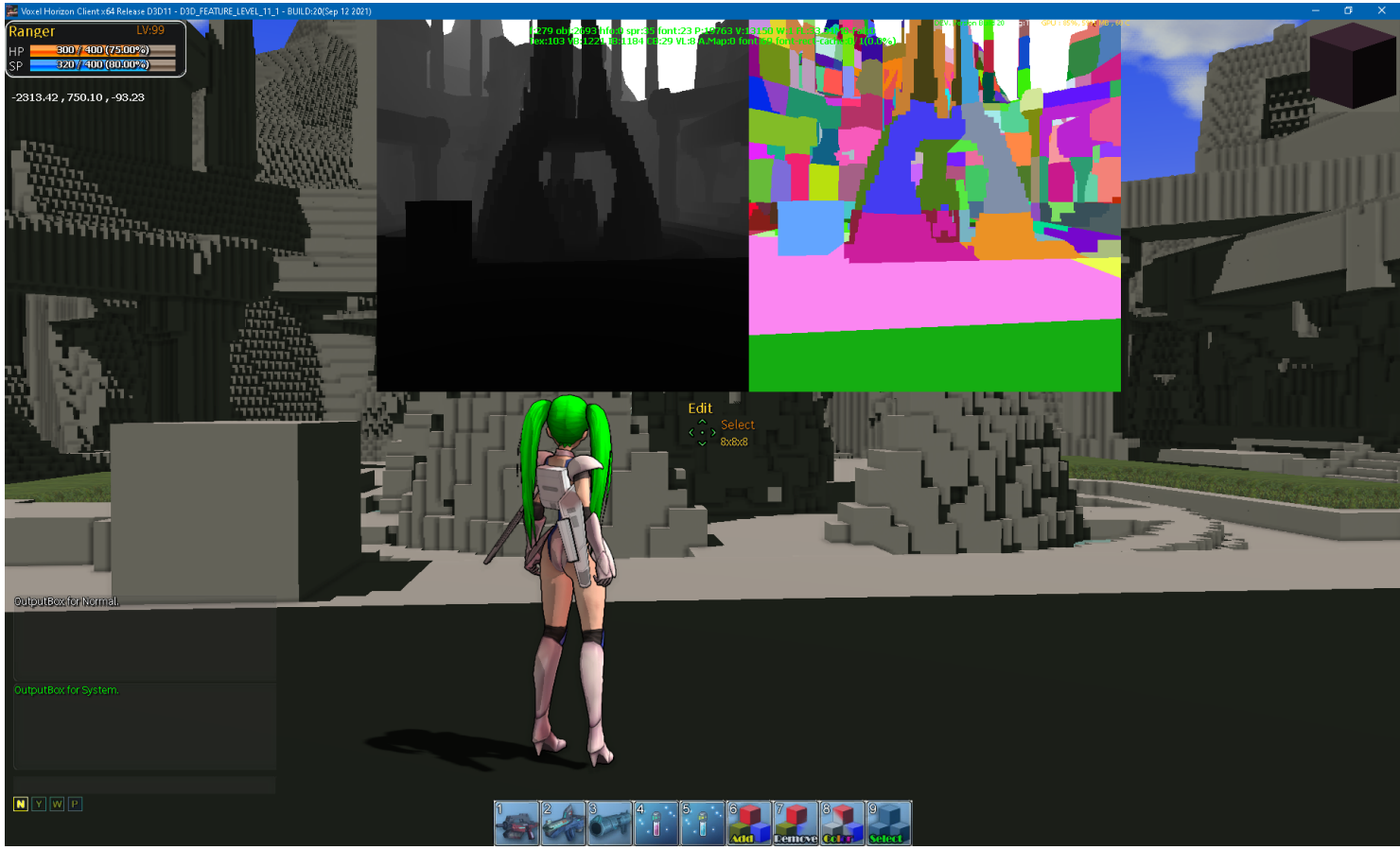
- 트리 재구성 비용을 줄이기 위해 트리는 최대 depth만큼 미리 빌드해 둔다.
- 트리는 어차피 CPU측에 먼저 구현해야 한다. (picking, CPU를 이용한 오브젝트 수집 등)
- GPU측(CUDA memory)에 트리 전체를 업데이트.
- 복셀 오브젝트가 추가/삭제/변경 될 경우 CUDA측 트리의 leaf내용만 업데이트.
- Raytracing을 이용한 depth, Object ID를 저장하는 코드는 CPU 코드로 먼저 작성한다. 1)검증, 2) 포팅 용이함을 위해서다.

GPU측에 KD-Tree구현하기

- KD-Tree 순회 코드는 CPU코드를 거의 그대로 사용할 수 있다.
- CUDA는 device memory와 host memory를 구분하므로 트리를 구축할때는 간단하지 않다. 팁을 제시하자면…
  1. cudaMallocManaged()를 사용한다. – 가장 쉽다. 하지만 느리다.
  2. cudaMalloc()/cudaMallocHost()를 이용해서 구현한다.
     a. Host메모리에 먼저 구현. 트리를 배열로 표현.
     b. 각 노드는 child node를 저장할때 포인터 대신 node 배열에서의 index로 저장.
     c. 똑같은 사이즈의 Device메모리를 만들어서 host->device로 한번에 카피.


System Memory
node 0
node 1
node 2
node 3
node 4
node 5
CUDA Code
KD_AXIS axis;
float d;
DWORD child[2];
node 0
node 1
node 2
node 3
node 4
node 5
Visual Studio Debugger
cudaMemcpy()
GPU Memory
node 0
node 1
node 2
node 3
node 4
node 5
Parallel NSight Debugger

성능 테스트



[i7 8700k 6 cores – 12 threads]

Single Thread - 2408.62 ms
Multi Thread – 12 thread(HT enabled) - 353.59 ms

CUDA(GTX970) - 12.2468 ms

60프레임 가능! 사용 가능!

성능 테스트

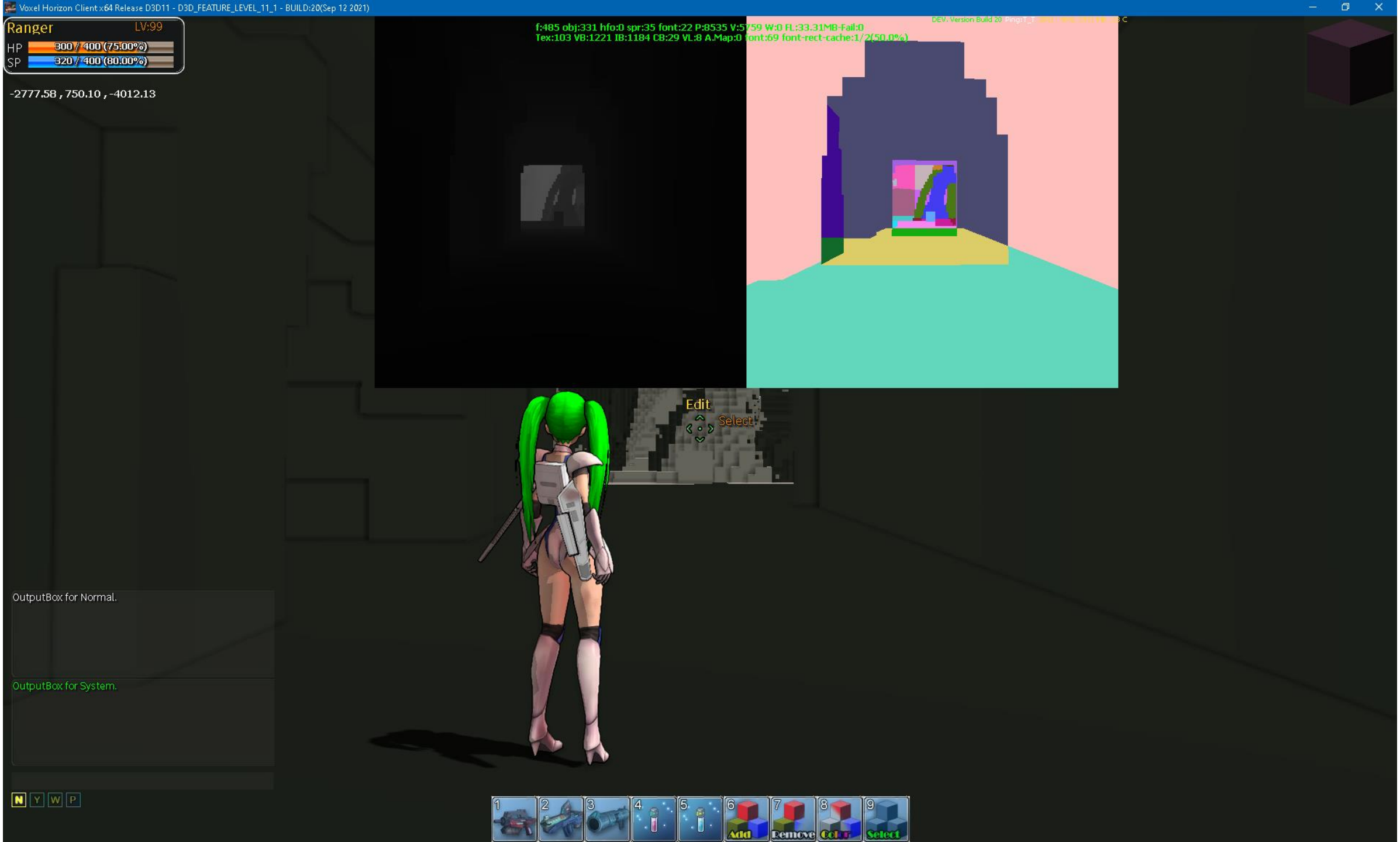
	SW Occlusion Culling	CUDA Raytracing (GTX970)	delta
DX11 (1920x1200)			
렌더링된 오브젝트	3580	2428	-32%
카메라 정지시 FPS	294	345	+17%
카메라 이동시 FPS	220	246	+11%
DXR (1920x1200)			
렌더링된 오브젝트	2675	1873	-29%
카메라 정지시 FPS	20	21	+5%
카메라 이동시 FPS	19	19	0%

오차 문제

- Ray Tracing버퍼의 해상도가 512x512정도면 충분한가?
  - 일반적으로 게임 해상도는 1920x1080이상이므로 512x512버퍼의 Raytracing테스트로는 완벽하지 않을 수 있다.
  - 멀리 있는 오브젝트들의 앞뒤 관계에서 오차가 발생할 수 있다.
  - 그렇다고 게임의 네이티브 해상도와 일치시키면 너무 느리다.

오차 문제 - 해결안

- Ray Tracing버퍼의 해상도가 512x512정도면 충분한가?
  - 일반적으로 게임 해상도는 1920x1080이상이므로 512x512버퍼의 Raytracing테스트로는 완벽하지 않을 수 있다.
  - 멀리 있는 오브젝트들의 앞뒤 관계에서 오차가 발생할 수 있다.
  - 그렇다고 게임의 네이티브 해상도와 일치시키면 너무 느리다.
  - 오브젝트의 ID대신 KD-Tree의 leaf의 ID를 저장한다.
  - 다소의 불필요한 렌더링을 감수한다. 오차를 줄인다.


Voxel Horizon Client x64 Release D3D11 - D3D_FEATURE_LEVEL_11_1 - BUILD:20(Sep 12 2021)
Ranger
LV:99
HP 300 / 400 (75.00%)
SP 320 / 400 (80.00%)
-2777.58 , 750.10 , -4012.13
f:485 obj:331 hfo:0 spr:35 font:22 P:8535 V:5759 W:0 FL:33.31MB-Fail:0
Tex:103 VB:1221 IB:1184 CB:29 VL:8 A.Map:0 font:69 font-rect-cache:1/2(50.0%)
Edit
Select
OutputBox for Normal.
OutputBox for System.
N Y W P
1 2 3 4 5 6 Add 7 Remove 8 Color 9 Select

제약사항

- nvidia GPU 한정
- GPU 성능 요구사항

생각해볼 주제

- CUDA 최적화

- OpenCL로의 포팅
  - 디버깅 환경이 거지같지만 포인터가 있으니 '의지'가 있다면 가능.
- Compute Shader로의 포팅
  - 이론상 불가능하진 않지만 포인터가 없기 때문에 꽤~~~~ 빡실듯.

참고자료

- https://graphics.stanford.edu/papers/gpu_kdtree/kdtree.pdf