DirectX Raytracing (DXR) 프로그래밍 소개

유영천

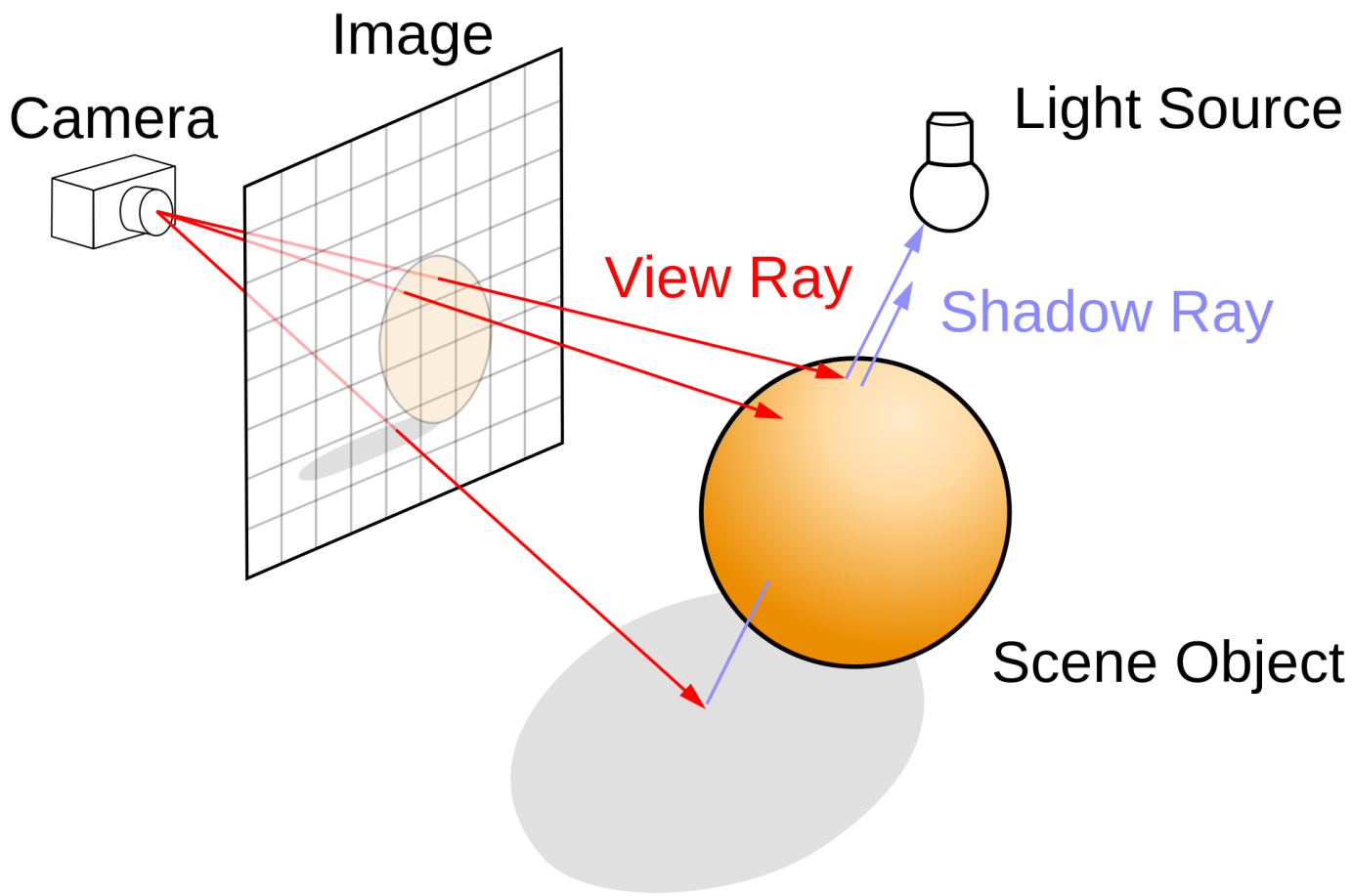
megayuchi.com

Tw: @dgtman

Raytracing?

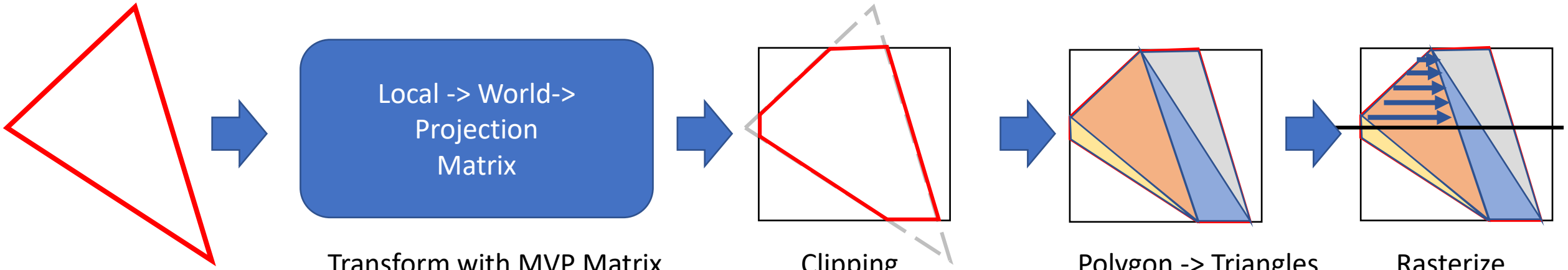
- **광선 추적**(Ray Tracing)은 가상적인 광선이 물체의 표면에서 반사되어, 카메라를 거쳐 다시 돌아오는 경로를 계산하는 것이다. 적게는 물체 하나가 반사하는 빛만 계산하면 되지만 많게는 물체를 구성하는 입자 하나하나의 빛을 전부 계산해야 되기 때문에 렌더링을 하는 데 있어 시간이 많이 소요되는 기술이다. (from wiki백과)

Raytracing으로 매시 그리기



https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

Rasterization으로 삼각형 그리기



실시간 raytracing

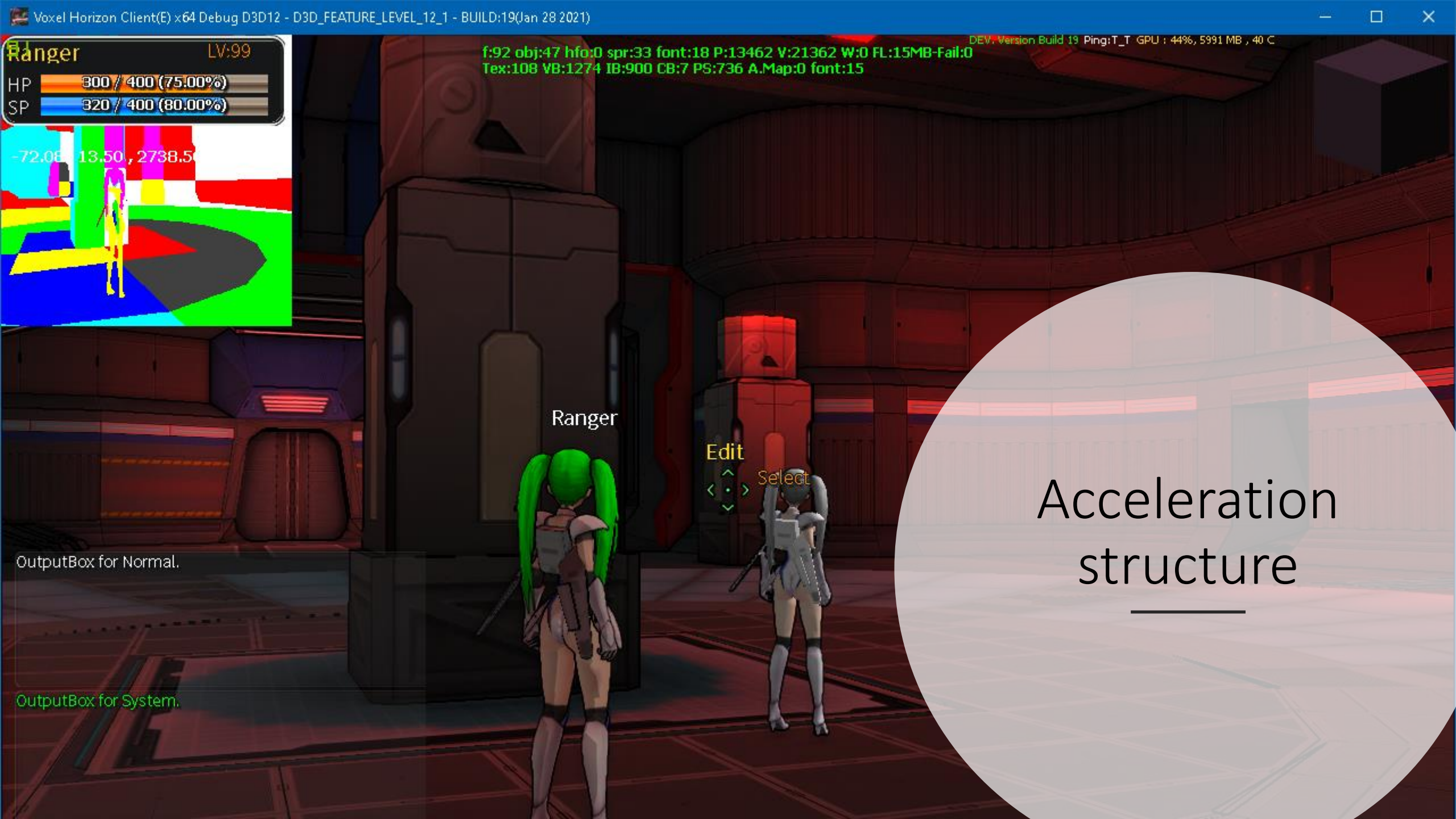
- Raytracing을 수행함에 있어서 가장 핵심적인 기능은 광선과 지형 지물간의 충돌 검출.
- KD-Tree, BVH등의 자료구조가 필요하다.
- 동적인 오브젝트들의 모양과 위치 변경을 빠르게 반영할 수 있어야 한다.
- CPU로도 가능. 하지만 연산능력에 한계가 있다.
- 대단위 병렬 연산을 사용하면 빠르게 처리할 수 있다.
- GPU로 해보자 -> CUDA로 진행했던 사례가 있다.

Why raytracing?

- 그림자 , 반사 처리가 너무 간편하게 된다!
- 오브젝트 특성에 따라 처리 방법이 달라지는 그림자와 반사 처리를 일관된 방법으로 처리할 수 있다. 따라서 전체 렌더링 코드가 단순해진다.
  - 반사 전처리 단계 -> Raytracing에선 필요없음.
  - 그림자 전처리 단계 -> Raytracing에선 필요없음.
- RTAO처리는 '결과적(denosing때문에)'으로 간편하지는 않지만 품질이 뛰어나다.
- 렌더링이 외에도 충돌처리, picking, Culling등 응용할 수 있는 범위가 넓다.

DirectX Raytracing – 빠르게 읽어보기

- GDC_DXR_deck.pdf (microsoft.com)

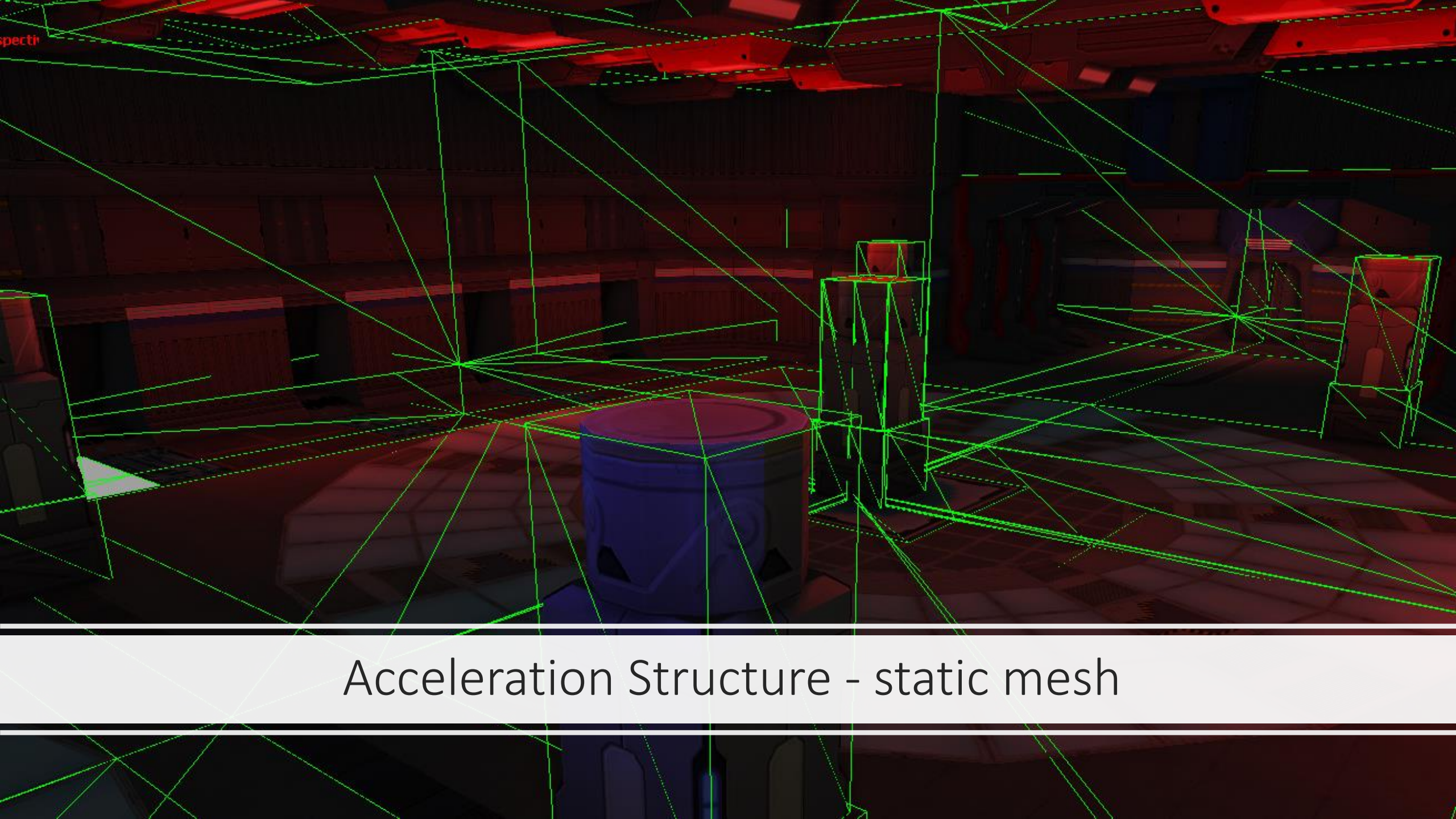

Voxel Horizon Client(E) x64 Debug D3D12 - D3D_FEATURE_LEVEL_12_1 - BUILD:19(Jan 28 2021)
Ranger
LV:99
HP
300 / 400 (75.00%)
SP
320 / 400 (80.00%)
DEV. Version Build 19 Ping:T_T GPU : 44%, 5991 MB , 40 C
f:92 obj:47 hfo:0 spr:33 font:18 P:13462 V:21362 W:0 FL:15MB-Fail:0
Tex:108 VB:1274 IB:900 CB:7 PS:736 A.Map:0 font:15
Ranger
Edit
Select
OutputBox for Normal.
OutputBox for System.
Acceleration structure

Acceleration Structure

- GPU 메모리에 위치하는 공간분할 구조.
- Bottom level, Top level의 2단계 구조.
- 빠르게 ray의 지형 지물의 충돌처리를 수행하기 위한 구조.
- 빠르게 지형 지물의 변경사항을 반영하기 위한 구조.

Acceleration Structure

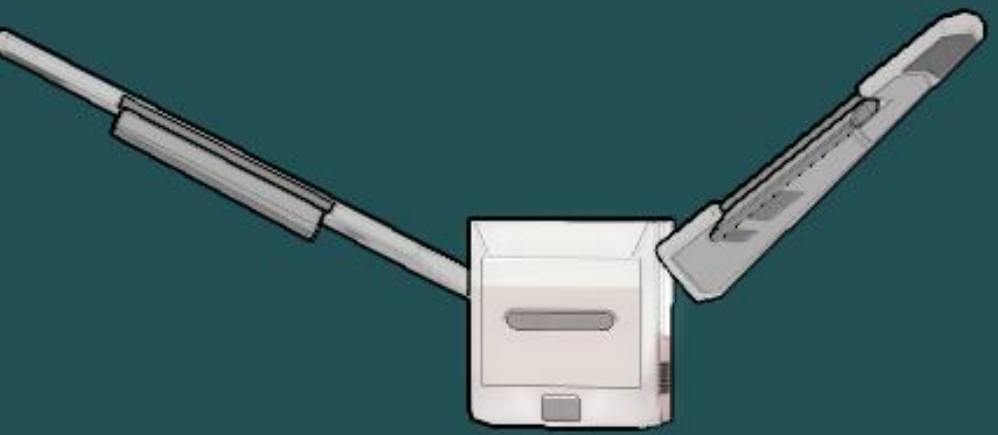
- Introduction to DirectX Raytracing Part 2 – the API (cwyman.org)
  - 7페이지부터
- DX12 Raytracing tutorial - Part 1 | NVIDIA Developer

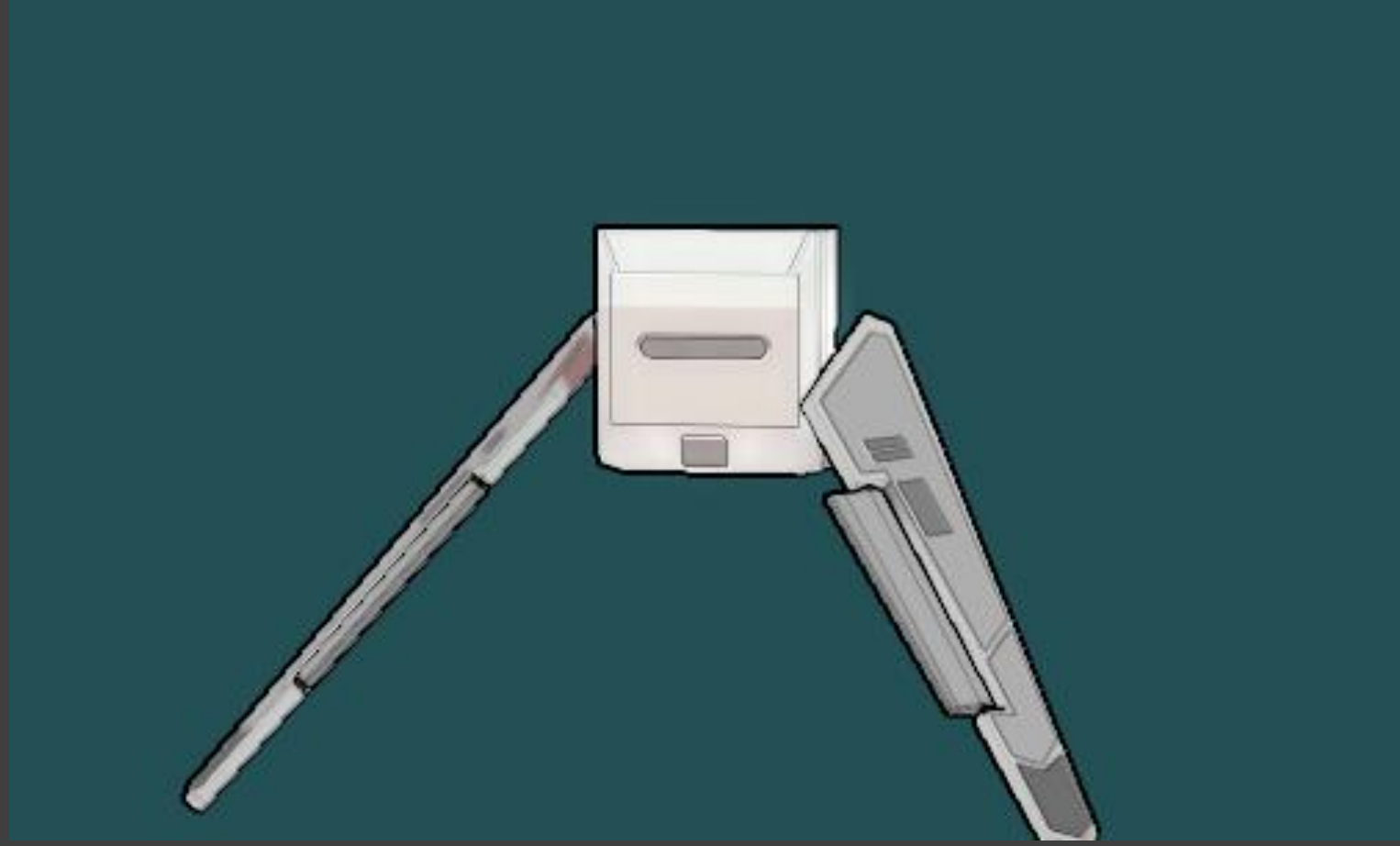


Acceleration Structure - static mesh

Acceleration Structure - static mesh

- 건물 등 움직이지 않는 지형 지물
- Index buffer -> BLAS에 그대로 사용 가능
- Vertex Buffer -> BLAS에 그대로 사용 가능
- BLAS는 최초 한번만 빌드하고 나면 손댈 필요가 없다.





Acceleration Structure – dynamic mesh(rigid)

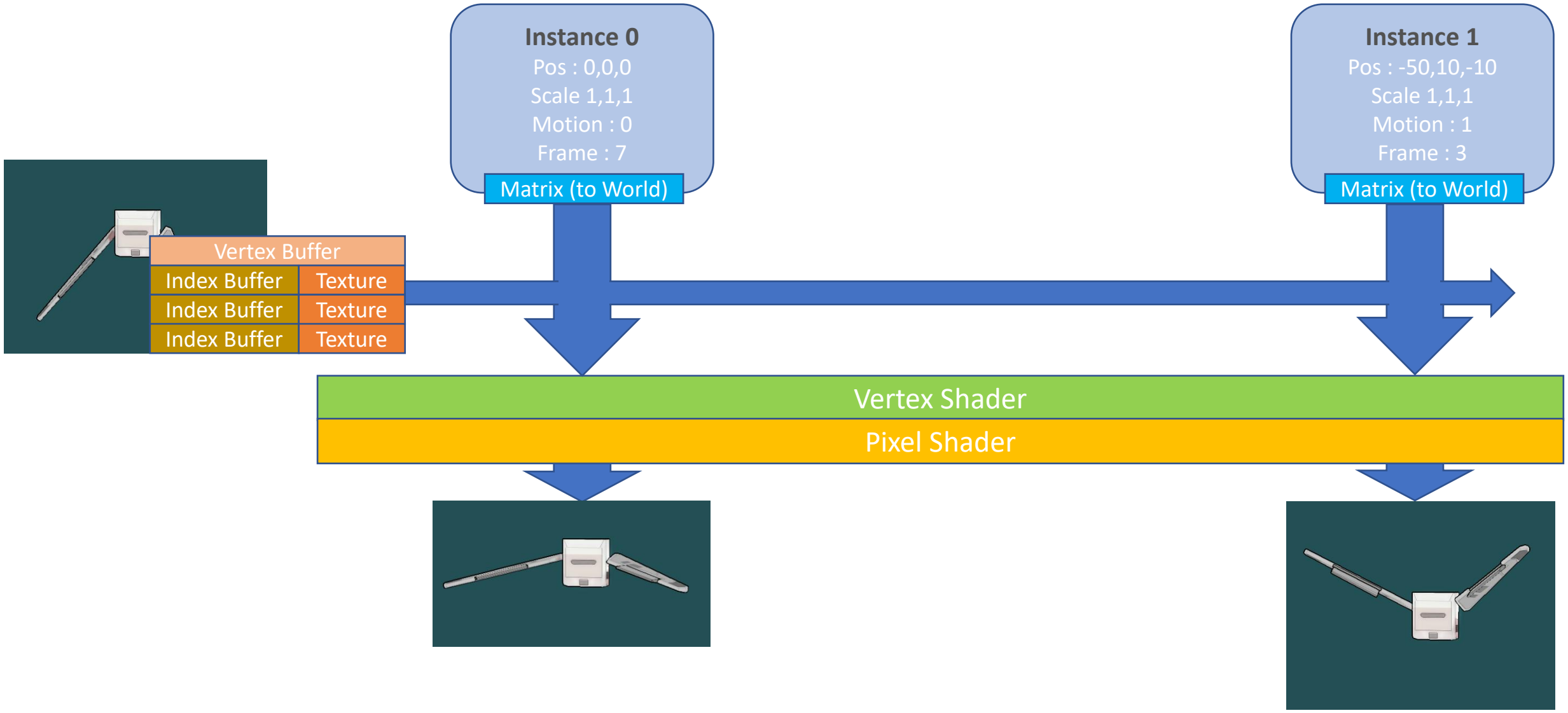
Acceleration Structure – dynamic mesh(rigid)

- 하나의 리소스로 여러 인스턴스 렌더링 가능
- 위치/스케일/회전 가능
- Index buffer -> BLAS에 그대로 사용 가능
- Vertex Buffer -> BLAS에 그대로 사용 가능
- BLAS업데이트는 필요 없다.
- 위치, 회전, 스케일 변환이 있을 경우 강체 변환이므로 변환 매트릭스만 수정해서 TLAS를 업데이트 하면 된다.
- World변환 매트릭스를 TLAS빌드/업데이트 시에 전달

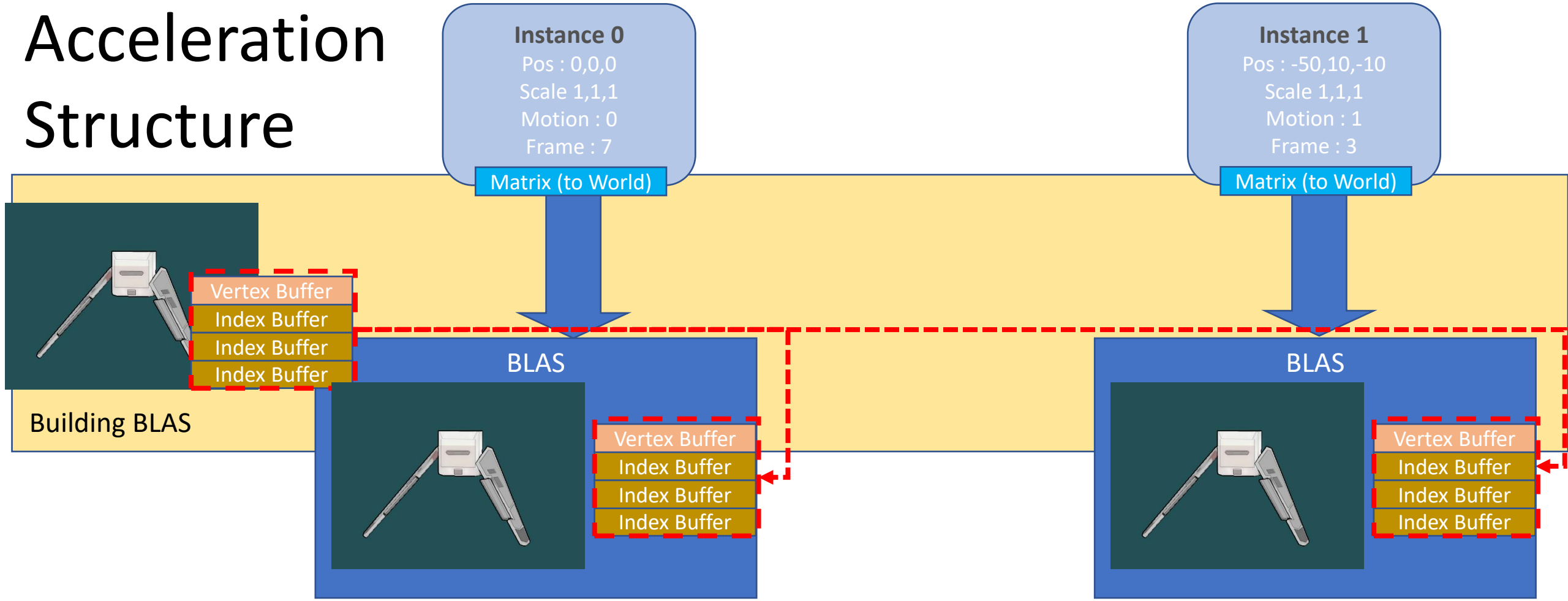
Static mesh / Dynamic mesh(rigid)

- 강체 변환이 가능한지 여부만 다르다.
- 둘 다 BLAS는 업데이트할 필요가 없다.
- Dynamic Mesh인 경우 변환 매트릭스에 변화가 생기면 TLAS를 업데이트하거나 다시 빌드해야 한다.

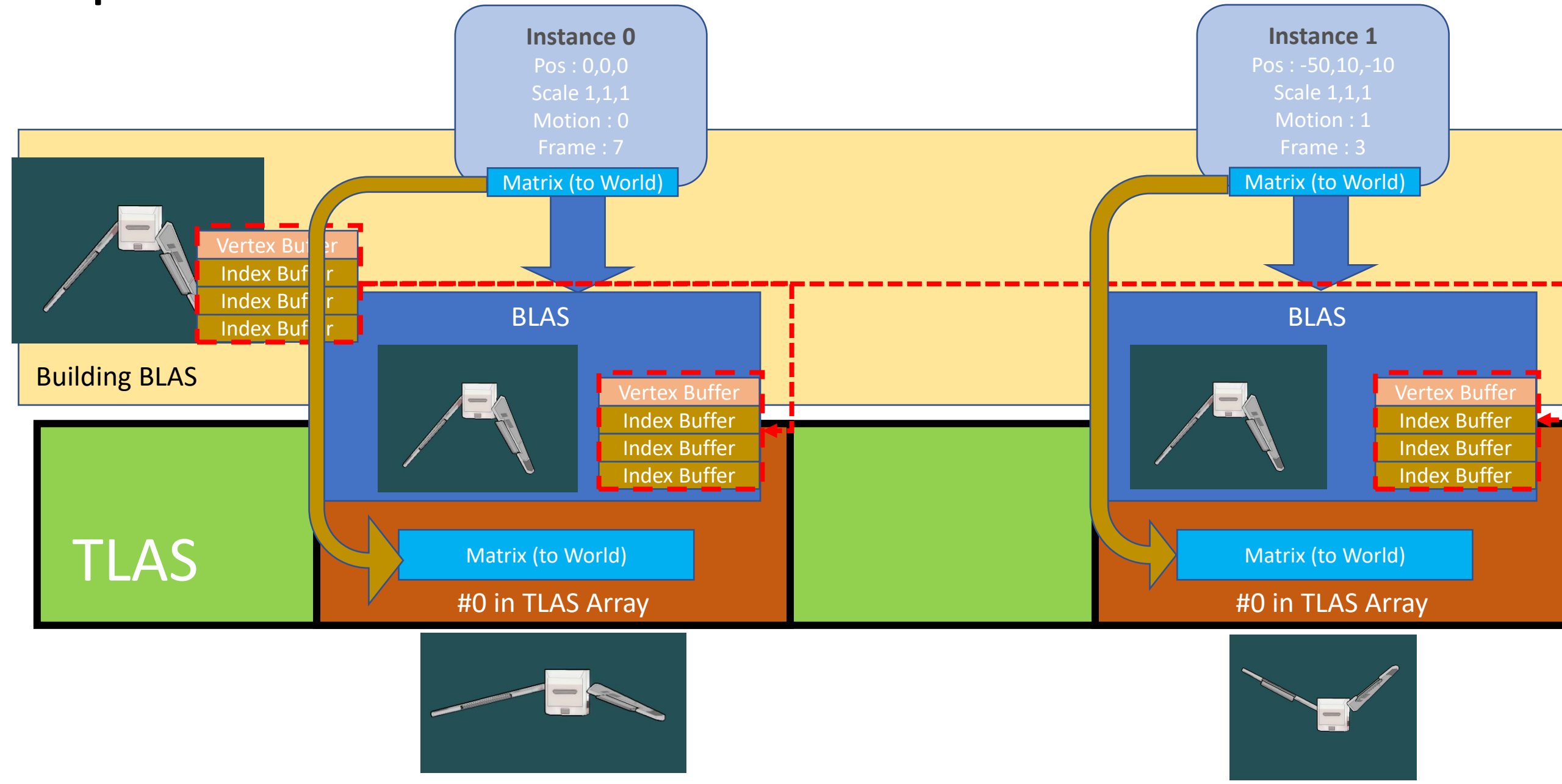
Rasterization



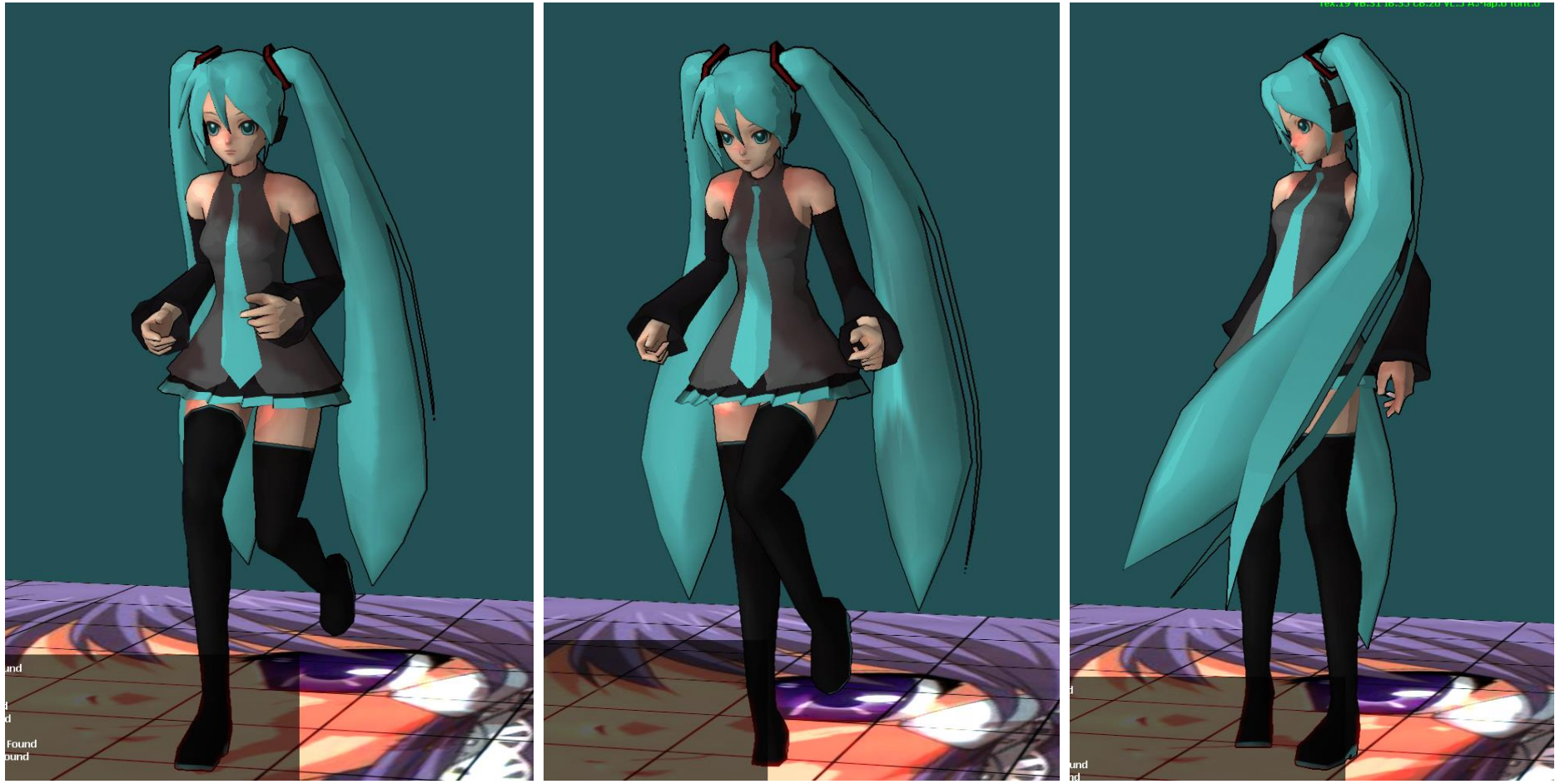
Bottom Level Acceleration Structure



Top Level Acceleration Structure



Acceleration Structure – dynamic mesh(skinned)



Acceleration Structure – dynamic mesh(skinned)

- 하나의 리소스로 여러 인스턴스 렌더링 가능
- 위치/스케일/회전 가능
- 로컬 좌표계에서의 버텍스 위치 변경 가능(모양이 변형된다)
- Index buffer -> BLAS에 그대로 사용 가능
- Vertex Buffer -> 변형되므로 BLAS Instance마다 하나씩 새로 생성
- Skin적용 애니메이션을 BLAS에 반영하기 위해 VertexBuffer를 업데이트 -> compute shader사용.
- 이때 world로는 변환하지 말고 local좌표인 상태로 버퍼에 저장
- World변환 매트릭스를 TLAS빌드/업데이트 시에 전달

Rasterization

Vertex Buffer

Index Buffer	Texture
Index Buffer	Texture
Index Buffer	Texture

Instance 0
Pos : 0,0,0
Scale 1,1,1
Motion : 1
Frame : 7

- Matrix (to World)
- Matrix (bone #0)
- Matrix (bone #1)
- Matrix (bone #2)
- Matrix (bone #n)

Instance 1
Pos : 100,0,-50
Scale 1,1,1
Motion : 4
Frame : 15

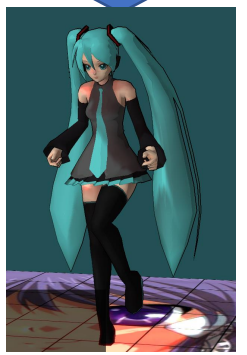
- Matrix (to World)
- Matrix (bone #0)
- Matrix (bone #1)
- Matrix (bone #2)
- Matrix (bone #n)

Instance 2
Pos : -50,10,-10
Scale 1,1,1
Motion :15
Frame : 3

- Matrix (to World)
- Matrix (bone #0)
- Matrix (bone #1)
- Matrix (bone #2)
- Matrix (bone #n)

Vertex Shader

Pixel Shader



Bottom Level Acceleration Structure

Instance 0
Pos : 0,0,0
Scale 1,1,1
Motion : 1
Frame : 7

- Matrix (to World)
- Matrix (bone #0)
- Matrix (bone #1)
- Matrix (bone #2)
- Matrix (bone #n)

Instance 1
Pos : 100,0,-50
Scale 1,1,1
Motion : 4
Frame : 15

- Matrix (to World)
- Matrix (bone #0)
- Matrix (bone #1)
- Matrix (bone #2)
- Matrix (bone #n)

Instance 2
Pos : -50,10,-10
Scale 1,1,1
Motion :15
Frame : 3

- Matrix (to World)
- Matrix (bone #0)
- Matrix (bone #1)
- Matrix (bone #2)
- Matrix (bone #n)

Vertex Buffer
Index Buffer
Index Buffer
Index Buffer

Updating BLAS

Compute Shader
Local 좌표 기준으로 vertex list를 변환(매시의 모양을 변경)

BLAS

참조
Index Buffer
Index Buffer
Index Buffer
Vertex Buffer

BLAS

참조
Index Buffer
Index Buffer
Index Buffer
Vertex Buffer

BLAS

참조
Index Buffer
Index Buffer
Index Buffer
Vertex Buffer

Top Level Acceleration Structure

Instance 0
Pos : 0,0,0
Scale 1,1,1
Motion : 1
Frame : 7

Instance 1
Pos : 100,0,-50
Scale 1,1,1
Motion : 4
Frame : 15

Instance 2
Pos : -50,10,-10
Scale 1,1,1
Motion :15
Frame : 3

Matrix (to World)
Matrix (bone #0)
Matrix (bone #1)
Matrix (bone #2)
Matrix (bone #n)

Vertex Buffer
Index Buffer
Index Buffer
Index Buffer

Compute Shader
Local 좌표 기준으로 vertex list를 변환(매시의 모양을 변경)

Updating BLAS

BLAS
참조
Index Buffer
Index Buffer
Index Buffer
Vertex Buffer
Matrix (to World)
#0 in TLAS Array

BLAS
참조
Index Buffer
Index Buffer
Index Buffer
Vertex Buffer
Matrix (to World)
#1 in TLAS Array

BLAS
참조
Index Buffer
Index Buffer
Index Buffer
Vertex Buffer
Matrix (to World)
#2 in TLAS Array

TLAS

Raytracing shaders

Raytracing shaders

- PowerPoint Presentation (cwyman.org)
  - 35페이지부터
- GDC_DXR_deck.pdf (microsoft.com)
  - 27페이지부터

Raytracing shaders

- Ray Generation Shader
  - 월드 공간에 최초로 뿌릴 ray를 생성하는 shader. 화면 좌표계 기준으로 생성한다.
- Closest-hit shader
  - TraceRay()를 호출했을 때 ray가 매시에 충돌하면 호출된다.
- Any-hit shader
  - Non opaque 매시에 충돌했을 때 호출된다. 주로 텍스처 알파를 얻어서 버릴지(완전 투명처리) 충돌로 인정할지를 판별한다.(굴절 등)
- Miss shader
  - 어떤 매시에도 충돌하지 않았을 때 호출되는 shader.
- 기본적으로 모두 Compute shader

Global Root parameter

- 모든 shader가 볼 수 있는 global 상수
- View, projection 매트릭스, material table , light table등을 설정
- Root Signature를 만드는 방법, shader로 전달하는 방법은 기존 D3D12 rasterization과 동일하다.

Global Root parameter

CPU Side

```
const UINT MAX_RTLIGHT_NUM = 32;
struct CONSTANT_BUFFER_RAY_TRACING
{
    MATRIX4     matViewInvArray[2];
    DECOMP_PROJ DecompProj[2];
    VECTOR4     v4CameraPosition;
    UINT MaxRadianceRayRecursionDepth;
    UINT MaxEffectRadianceRayRecursionDepth;
    UINT MaxShadowRayRecursionDepth;
    UINT RTLightNum;
    RT_LIGHT    RTLightList[MAX_RTLIGHT_NUM];
};
```

GPU Side(hlsl)

```
// Global Root Parameter
RWTexture2D<float4> RenderTarget : register(u0);
Texture2D       texGBufferDiffuse : register(t1);
Texture2D       texGBufferNormal : register(t2);
Texture2D       texGBufferDepth : register(t3);
Texture2D       texGBufferProperty : register(t4);
StructuredBuffer<PrimitiveMaterialBuffer>   g_MtlTable : register(t5);
RaytracingAccelerationStructure Scene : register(t0, space0);
SamplerState    samplerWrap         : register(s0);
```

Local Root parameter

- Ray가 임의의 geometry에 충돌시 closest hit shader, any hit shader가 호출된다.
- Root Signature를 만드는 방법은 기존 D3D12 rasterization과 동일하다.
- Hit group shader table을 통해 shader에 전달한다.
- Shader 호출시 규칙에 따라 각 geometry는 자신에게 맵핑된 hit group record의 local root parameter를 보게 된다.
- Geometry 각각의 Index Buffer, texture, global material table의 index등을 설정.

Local Root parameter

CPU Side

```
struct ROOT_ARG
{
    CONSTANT_BUFFER_RAY_GEOM cb;
    D3D12_GPU_DESCRIPTOR_HANDLE srvVB;
    D3D12_GPU_DESCRIPTOR_HANDLE srvVBTexCoord;
    D3D12_GPU_DESCRIPTOR_HANDLE srvIB;
    D3D12_GPU_DESCRIPTOR_HANDLE srvTexDiffuse;
};
```

GPU Side(hlsl)

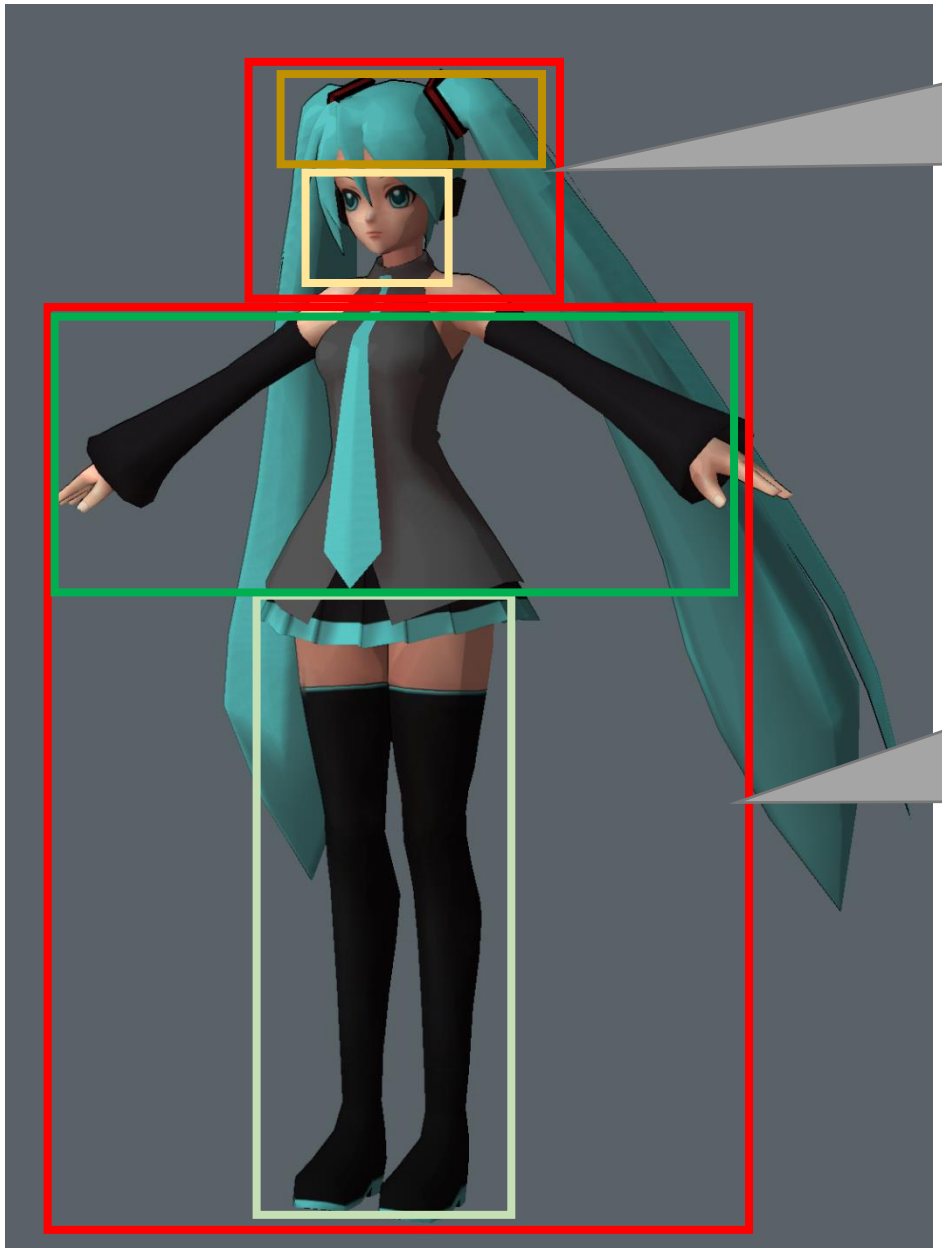
```
// Local Root Parameter
ConstantBuffer<CONSTANT_BUFFER_RAY_GEOM> l_rayGeomCB : register(b0, space1);
StructuredBuffer<D3DVLVERTEX> l_Vertices : register(t0, space1);
StructuredBuffer<TVERTEX> l_TVertices : register(t1, space1);
ByteAddressBuffer l_Indices : register(t2, space1);
Texture2D<float4> l_texDiffuse : register(t3, space1);
```

Hit Group Shader Table

Hit Group Shader Table

- Introduction to DirectX Raytracing Part 2 – the API (cwyman.org)
  - 19페이지부터

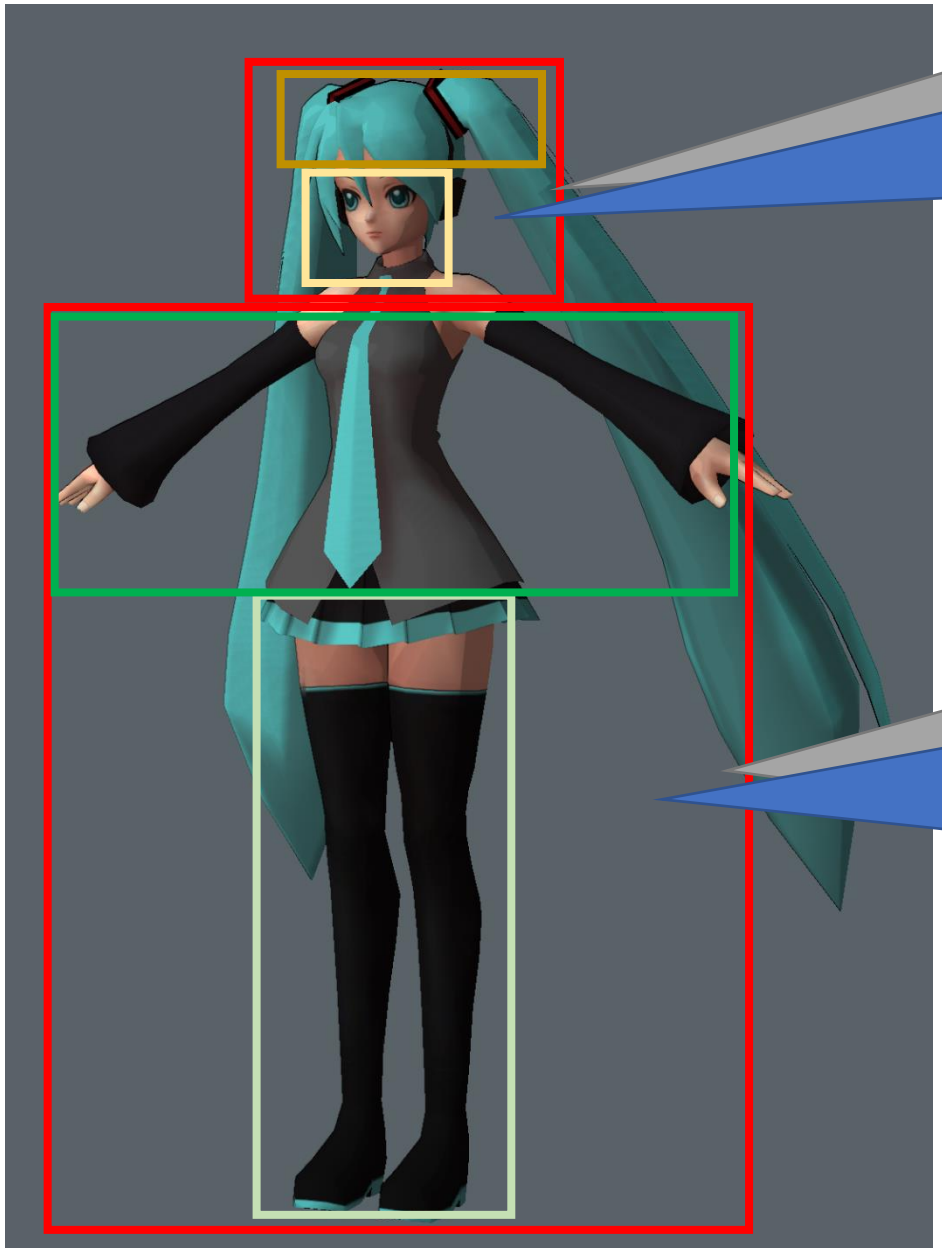
3dsmax



Object : “head”
Sub materials
+ #1: hair.dds
+ #2: face.dds

Object : “body”
Sub materials
+ #1: body_lower.dds
+ #2: body_upper.dds

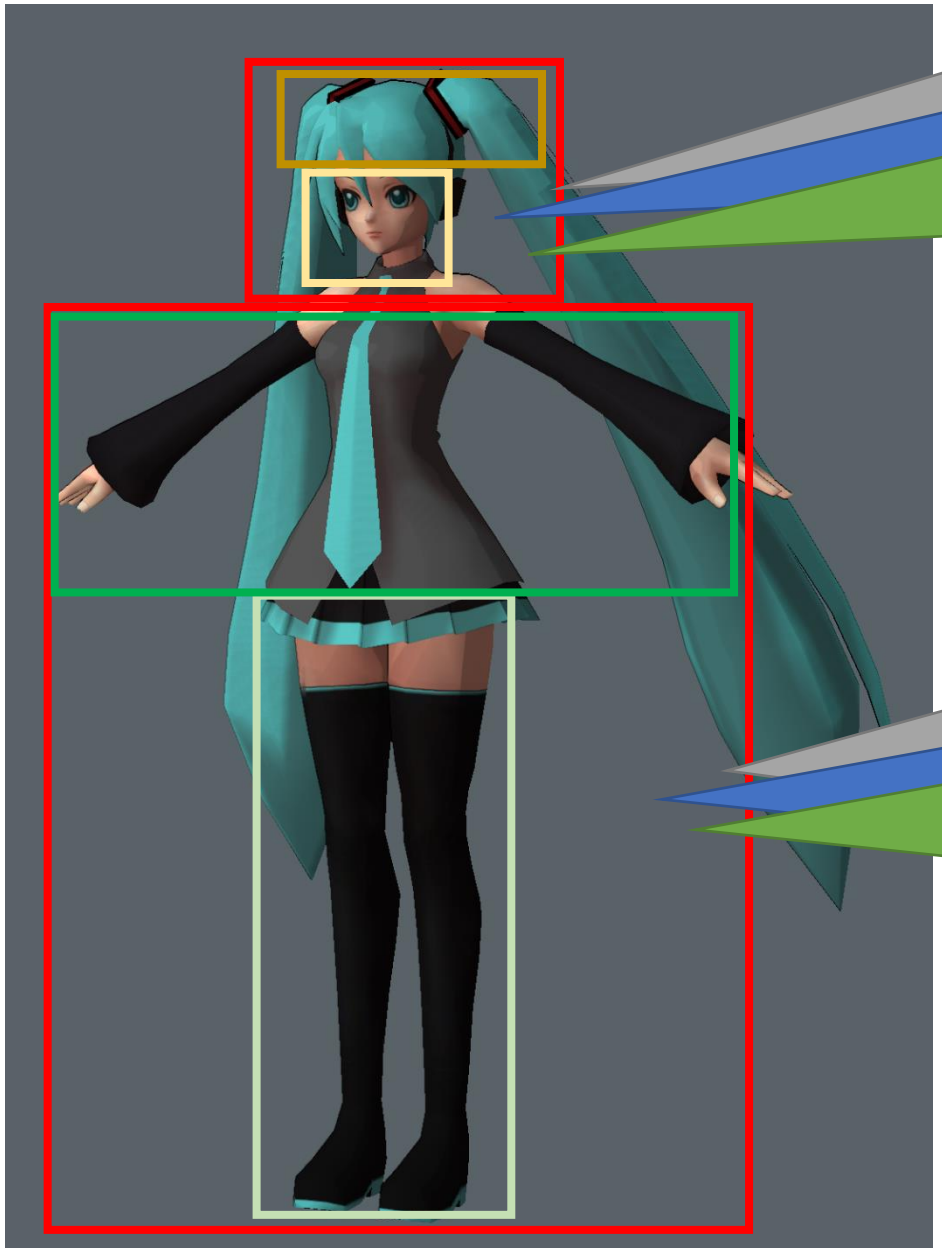
Megayuchi Engine



Object : “head”
+ Vertex Buffer
FaceGroup List
+ IndexBuffer , Mtl (hair.dds)
+ IndexBuffer, Mtl (face.dds)

Object : “body”
+ Vertex Buffer
FaceGroup List
+ IndexBuffer , Mtl (body_lower.dds)
+ IndexBuffer, Mtl (body_upper.dds)

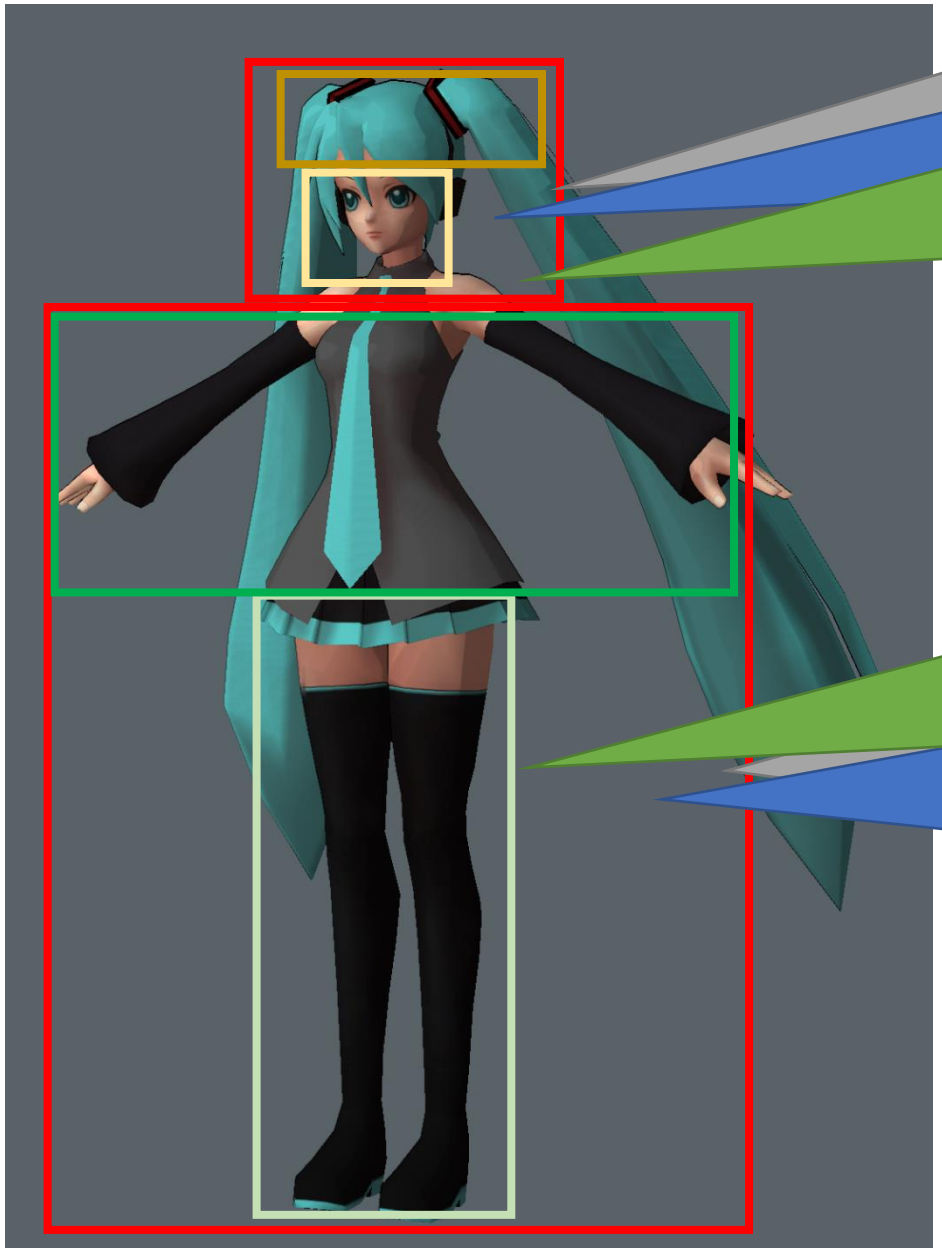
DXR - Acceleration Structure



BLAS instance #0
- Vertex Buffer – per BLAS
Geometry List
+ IndexBuffer – shared
+ IndexBuffer - shared

BLAS instance #1
- Vertex Buffer – per BLAS
Geometry List
+ IndexBuffer – shared
+ IndexBuffer - shared

DXR - Hit Group Shader Table



BLAS instance #0
- Vertex Buffer – per BLAS
- ContributionToHitGroupIndex : 0

Geometry List
- #0 - IndexBuffer(shared)
- #1 - IndexBuffer(shared)

BLAS instance #1
- Vertex Buffer – per BLAS
- ContributionToHitGroupIndex : 4

Geometry List
- #0 - IndexBuffer(shared)
- #1 - IndexBuffer(shared)

Hit group #0 – radiance
- Closest shader (radiance)
- AnyHit shader (radiance)

Local Root Param

Hit group #1 – shadow
- Closest shader (shadow)
- AnyHit shader (shadow)

Local Root Param

Hit group #2 – radiance
- Closest shader (radiance)
- AnyHit shader (radiance)

Local Root Param

Hit group #3 – shadow
- Closest shader (shadow)
- AnyHit shader (shadow)

Local Root Param

Hit group #4 – radiance
- Closest shader (radiance)
- AnyHit shader (radiance)

Local Root Param

Hit group #5 – shadow
- Closest shader (shadow)
- AnyHit shader (shadow)

Local Root Param

Hit group #6 – radiance
- Closest shader (radiance)
- AnyHit shader (radiance)

Local Root Param

Hit group #7 – shadow
- Closest shader (shadow)
- AnyHit shader (shadow)

Local Root Param

꼭 봐야할 샘플코드

- GitHub - microsoft/DirectX-Graphics-Samples: This repo contains the DirectX Graphics samples that demonstrate how to build graphics intensive applications on Windows.
- 최초로 시작할때
  - DirectX-Graphics-Samples\Samples\Desktop\D3D12Raytracing\src\D3D12RaytracingHelloWorld
- 실제 렌더링에 적용할때
  - DirectX-Graphics-Samples\Samples\Desktop\D3D12Raytracing\src\D3D12RaytracingRealTimeDenoisedAmbientOcclusion

TIP

- 최우선적으로 AS를 정상적으로 구축할 수 있도록 할 것
- G-Buffer를 미리 구축해두면 성능을 높일 수 있다.
- Raytracing은 Rasterization을 완전히 대체하지는 못한다. Rasterization은 필요하다.

참고자료

- GDC_DXR_deck.pdf (microsoft.com)
- Introduction to DirectX RayTracing (cwyman.org)
  - Introduction to DirectX Raytracing Part 2 – the API (cwyman.org)
  - PowerPoint Presentation (cwyman.org)
- DX12 Raytracing tutorial - Part 1 | NVIDIA Developer
- DX12 Raytracing tutorial - Part 2 | NVIDIA Developer
- DirectX Raytracing (DXR) Functional Spec | DirectX-Specs (microsoft.github.io)
- Tips and Tricks: Ray Tracing Best Practices | NVIDIA Developer Blog