D3D12 Mesh Shader 소개

유영천

https://megayuchi.com

https://youtube.com/megayuchi

들어가기 전에

- Mesh Shader란 무엇인가에 대해 설명합니다.
- Mesh Shader를 이용한 고급 주제에 대해서는 다루지 않습니다.
- Mesh Shader를 이용한 Tessellation을 아직 구현하지 않았으므로 이에 대해 설명할 수 없습니다.
- Mesh Shader를 이용한 LOD도 아직 구현하지 않았으므로 이에 대해 설명할 수 없습니다.

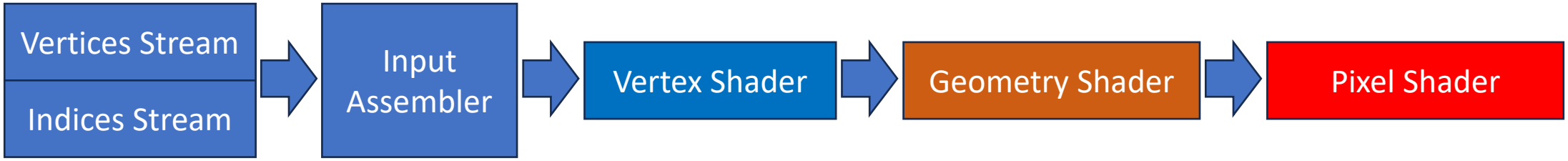
GPGPU

- 실행 주체는 마이크로 스레드
- Warp(wave)단위 스케쥴링이 비교적 투명하게 드러남.
- 읽기/쓰기 랜덤 억세스
- 기본적으로는 그래픽 파이프라인과 상관없어 보인다.

그래픽 shader의 GPGPU화

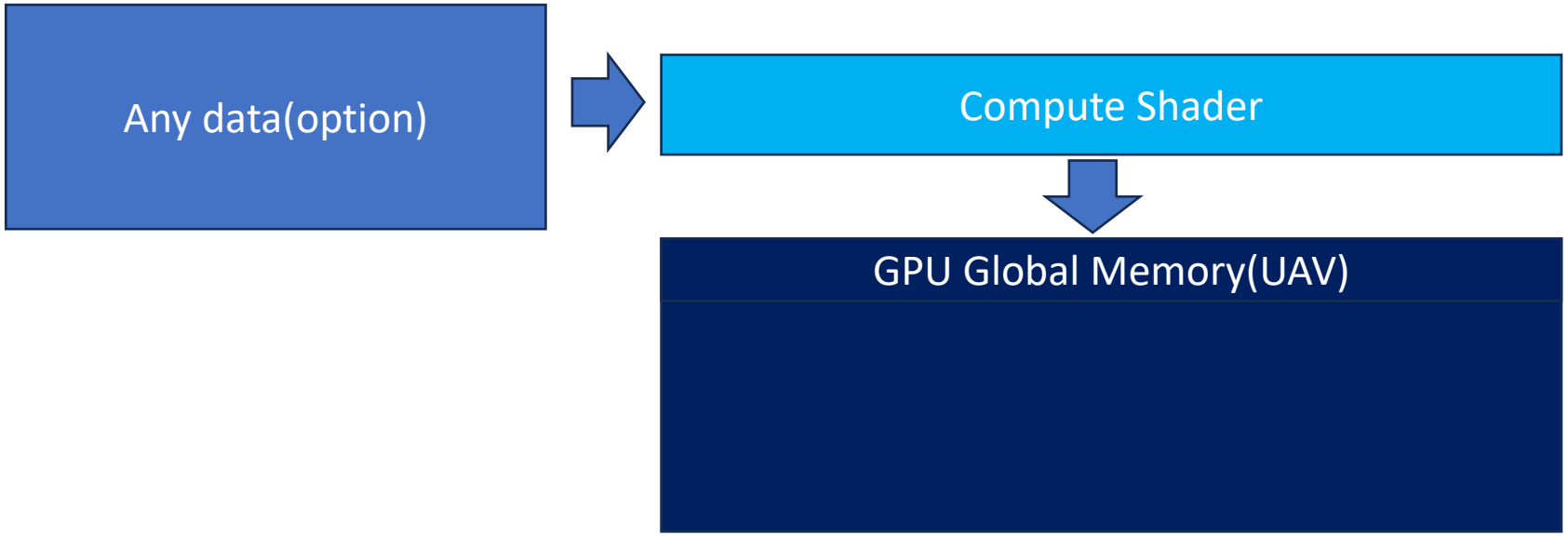
- Apple Metal이 대표적. 모든 shader가 그냥 GPGPU스타일임.
  - 입력 어셈블러 없이 raw포인터를 받아서 랜덤 억세스.
  - Geometry Shader가 없음. Mesh Shader로 구현하라고 함.
- 기존에는 Programmable Pipeline을 구현하는 것이 가장 큰 목표였다면, 이제는 Programmable Pipeline+ High Performance를 목표로 한다.
- GPU성능을 최대한으로 끌어내려면 GPU 스레드 구조에 코드를 맞춰야 한다.

Vertex Shader Pipeline



점(vertex) 과 점의 인덱스(indexed triangle)를 ‘입력 ' 받아서 최종 삼각형으로 출력.

Compute Shader Pipeline

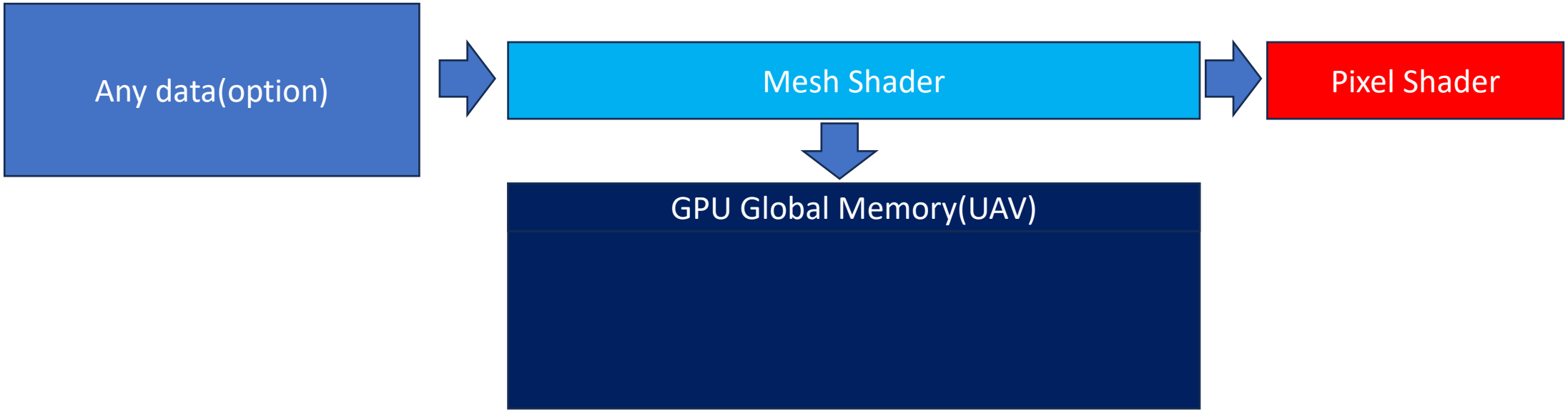


임의의 데이터를 Compute Shader코드(주체)에서 직접 억세스 해서 처리한 결과물을 임의의 메모리에 저장.

Mesh Shader?

- GPGPU(Compute Shader)에 삼각형 출력 기능을 추가
- 기본적으로 Compute Shader이다.
- Vertex와 Indexed Triangle 입력이 있어도 되고 없어도 된다.
- 입력이 있을 경우 랜덤 억세스로 필요한 데이터를 사용한다.
- 입력이 없어도 삼각형 출력을 만들어낸다!(가장 중요한 포인트)

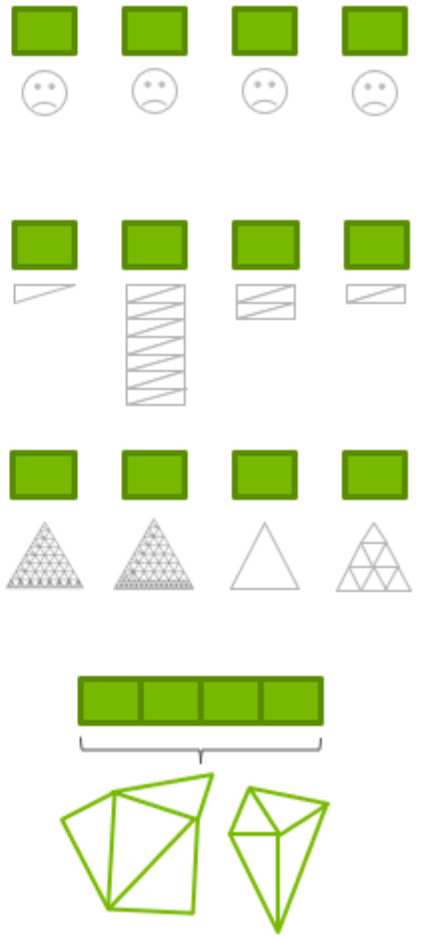
Mesh Shader Pipeline



임의의 데이터를 Compute Shader코드(주체)에서 직접 억세스 해서 처리한 결과물을 임의의 메모리에 저장, or (삼각형을) Pixel Shader로 출력.

Thread대응 관계

Shader		Thread Mapping	Topology
Vertex Shader	No access to connectivity	1 Vertex	No influence
Geometry Shader	Variable output doesn't fit HW well	1 Primitive / 1 Output Strip	Triangle Strips
Tessellation Shader	Fixed-function topology	1 Patch / 1 Evaluated Vertex	Fast Patterns
Mesh Shader	Compute shader features	Flexible	Flexible within work group allocation



https://developer.nvidia.com/blog/introduction-turing-mesh-shaders/

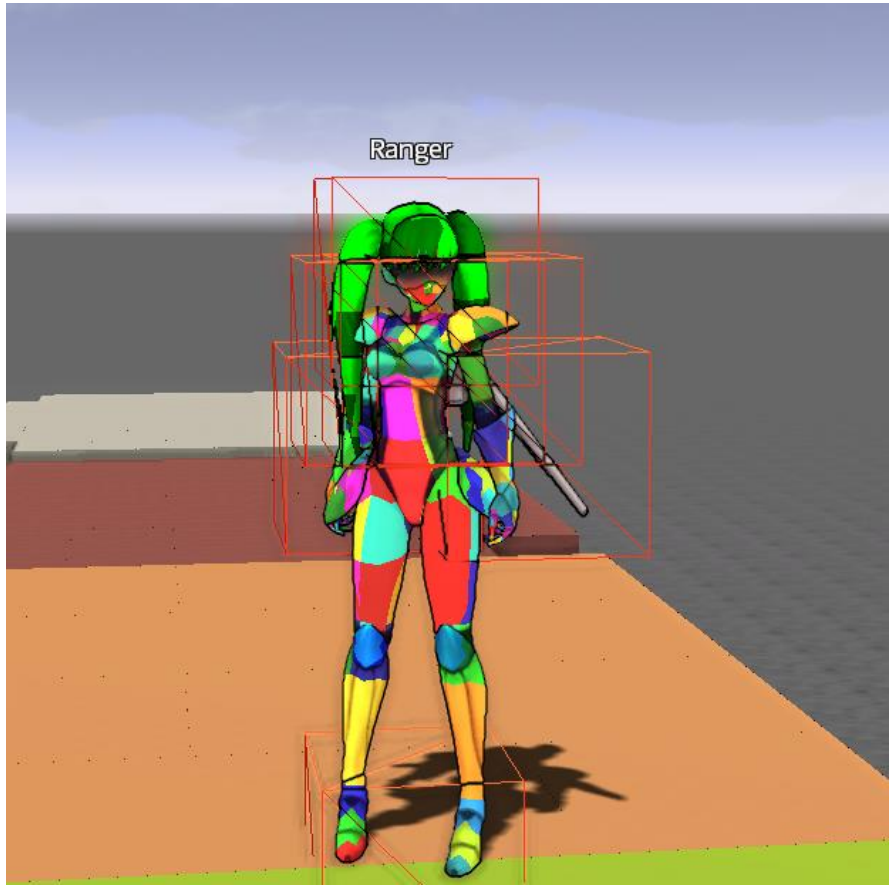
입력 vertex/index stream 없이 BOX 출력

```
static float3 g_boxVertexList[8] = {
                                    float3(-1, 1, 1), float3(-1, -1, 1),
                                    float3(1, -1, 1), float3(1, 1, 1),
                                    float3(-1, 1, -1), float3(-1, -1, -1),
                                    float3(1, -1, -1), float3(1, 1, -1)
};
groupshared float4 g_groupVertexList[8];

[NumThreads(THREAD_NUM_PER_GROUP, 1, 1)]
[OutputTopology("triangle")]
void msBox(uint threadID : SV_GroupThreadID, uint groupID : SV_GroupID, out indices uint3 pOutTriList[12], out vertices VS_OUTPUT pOutVertexList[36])
{
    SetMeshOutputCounts(36, 12);

    if (threadID < 8)
    {
        float4 PosBase = float4(0, 0, 0, 1);
        float4 PosWorld = mul(PosBase, matWorld) + float4(g_boxVertexList[threadID].xyz * 25.0, 0);
        g_groupVertexList[threadID] = PosWorld;
    }
    GroupMemoryBarrierWithGroupSync();

    if (threadID < 12)
    {
        uint3 IndexedTri = g_boxIndexList[threadID];
        float4 PosWorld[3] =
        {
            g_groupVertexList[IndexedTri.x],
            g_groupVertexList[IndexedTri.y],
            g_groupVertexList[IndexedTri.z]
        };
        float3 N = CalcNormalWithTri((float3)PosWorld[0], (float3)PosWorld[1], (float3)PosWorld[2]);
        float cosang = saturate(dot(N, (float3)(-LightDir)));
        for (uint i = 0; i < 3; i++)
        {
            uint BoundsVertexID = threadID * 3 + i;
            float4 PosOut = mul(PosWorld[i], matViewProjArray[ArrayIndex]);
            pOutVertexList[BoundsVertexID].Pos = PosOut;
        }
        pOutTriList[threadID] = uint3(threadID * 3 + 0, threadID * 3 + 1, threadID * 3 + 2);
    }
}
```



Mesh Shader 적용으로 기대되는 이득

1. 프로그래밍의 유연함
2. 더 높은 성능
3. 1,2로 인해 (결과적으로) 더 미려한 그래픽 제공

구현

- MeshShader생성 후 `D3DX12_MESH_SHADER_PIPELINE_STATE_DESC::MS` 멤버에 설정 -> ID3D12PipelineState객체 생성
- 기존 파이프 라인을 대체하려면 (실질적으로)Vertex Stream/Index Stream은 필요함.
- Vertex Stream / Index Stream을 UAV로 전달.
  - IASetIndexBuffer(), IASetVertexBuffer()사용하지 않음.
- 기존 VS파이프라인과 마찬가지로 Constant Buffer는 그대로 전달.
- Mesh Shader함수에서 UAV로 전달받은 Vertex Stream과 Index Stream과 Constant Buffer를 랜덤 억세스.

PipelineState 생성

```
D3D12_GRAPHICS_PIPELINE_STATE_DESC psoDescRef = {};
SetDefaultPipelineStateDesc(&psoDescRef);
SetDefaultPipeLineStateRenderTargets(&psoDescRef);

D3DX12_MESH_SHADER_PIPELINE_STATE_DESC psoDescMS = {};
psoDescMS.pRootSignature = m_pRootSignature;
psoDescMS.NumRenderTargets = psoDescRef.NumRenderTargets;
memcpy(psoDescMS.RTVFormats, psoDescRef.RTVFormats, sizeof(DXGI_FORMAT) * psoDescRef.NumRenderTargets);
psoDescMS.DSVFormat = psoDescRef.DSVFormat;
psoDescMS.RasterizerState = psoDescRef.RasterizerState;
psoDescMS.BlendState = psoDescRef.BlendState;
psoDescMS.BlendState.RenderTarget[0].RenderTargetWriteMask = D3D12_COLOR_WRITE_ENABLE_ALL;
psoDescMS.DepthStencilState = psoDescRef.DepthStencilState;
psoDescMS.SampleMask = psoDescRef.SampleMask;
psoDescMS.SampleDesc = psoDescRef.SampleDesc;

psoDescMS.MS = CD3DX12_SHADER_BYTECODE(m_pMS->pCodeBuffer, m_pMS->dwCodeSize);
psoDescMS.AS = CD3DX12_SHADER_BYTECODE(m_pMS_AS->pCodeBuffer, m_pMS_AS->dwCodeSize);
psoDescMS.PS = CD3DX12_SHADER_BYTECODE(m_pPS->pCodeBuffer, m_pPS->dwCodeSize);

CD3DX12_PIPELINE_MESH_STATE_STREAM psoStream = CD3DX12_PIPELINE_MESH_STATE_STREAM(psoDescMS);

D3D12_PIPELINE_STATE_STREAM_DESC streamDesc;
streamDesc.pPipelineStateSubobjectStream = &psoStream;
streamDesc.SizeInBytes = sizeof(psoStream);

pDevice->CreatePipelineState(&streamDesc, IID_PPV_ARGS(&m_pPipelineStateMS));
```

Mesh Shader 제한 사항

- 각 Mesh Shader 함수는 최대 256개의 삼각형 집합(uint3 index stream)과 256개의 점 집합(vertex stream)만 출력 가능
- 따라서 점과 삼각형이 최대 256개 이내의 mesh로 잘라서 Mesh Shader에 전달해야 한다.
  - Ex) 1024 triangles -> DispatchMesh(1024/256, 1, 1);
- 실질적으로 Mesh Shader를 사용하기 위해선 전체 mesh를 임의의 단위로 잘라야 한다.

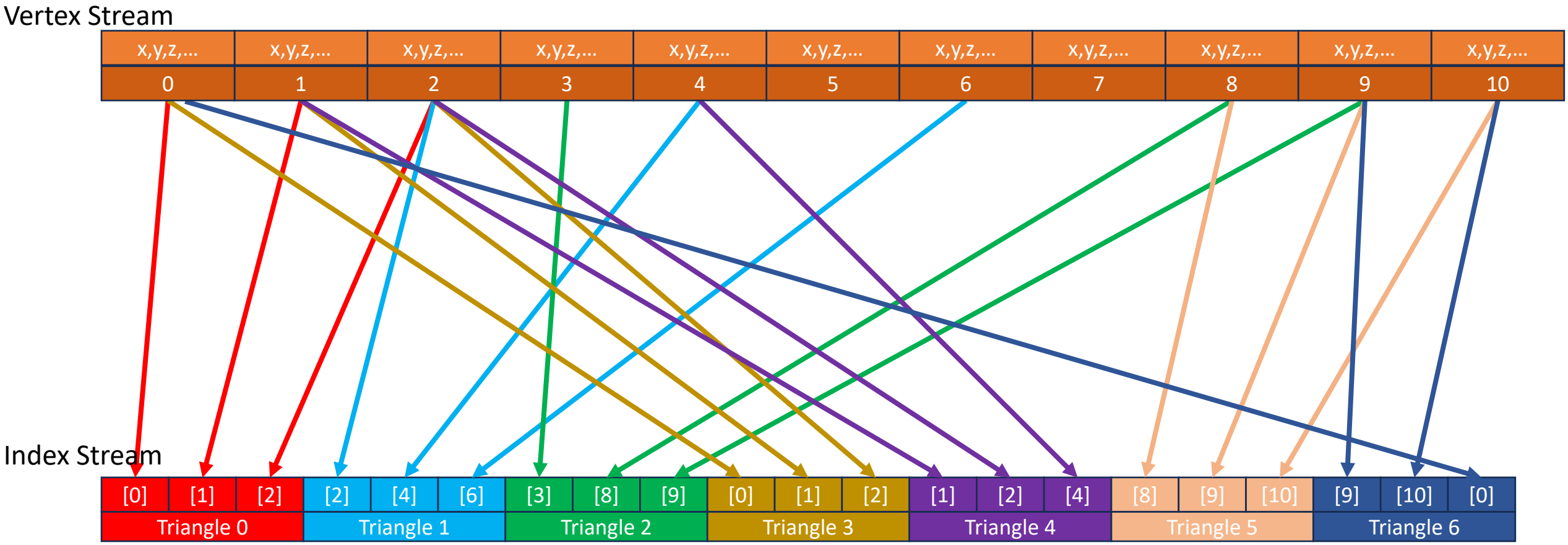
Meshlet

- 임의의 단위로 잘린 조각들을 Meshlet이라 부르기로 하자.
- 1 Meshlet -> Mesh Shader 함수 1 Group에 대응.
- Vertex data/ Index data(혹은 Vertex Data / Index Data를 참조할 수 있는)를 포함해야 한다.

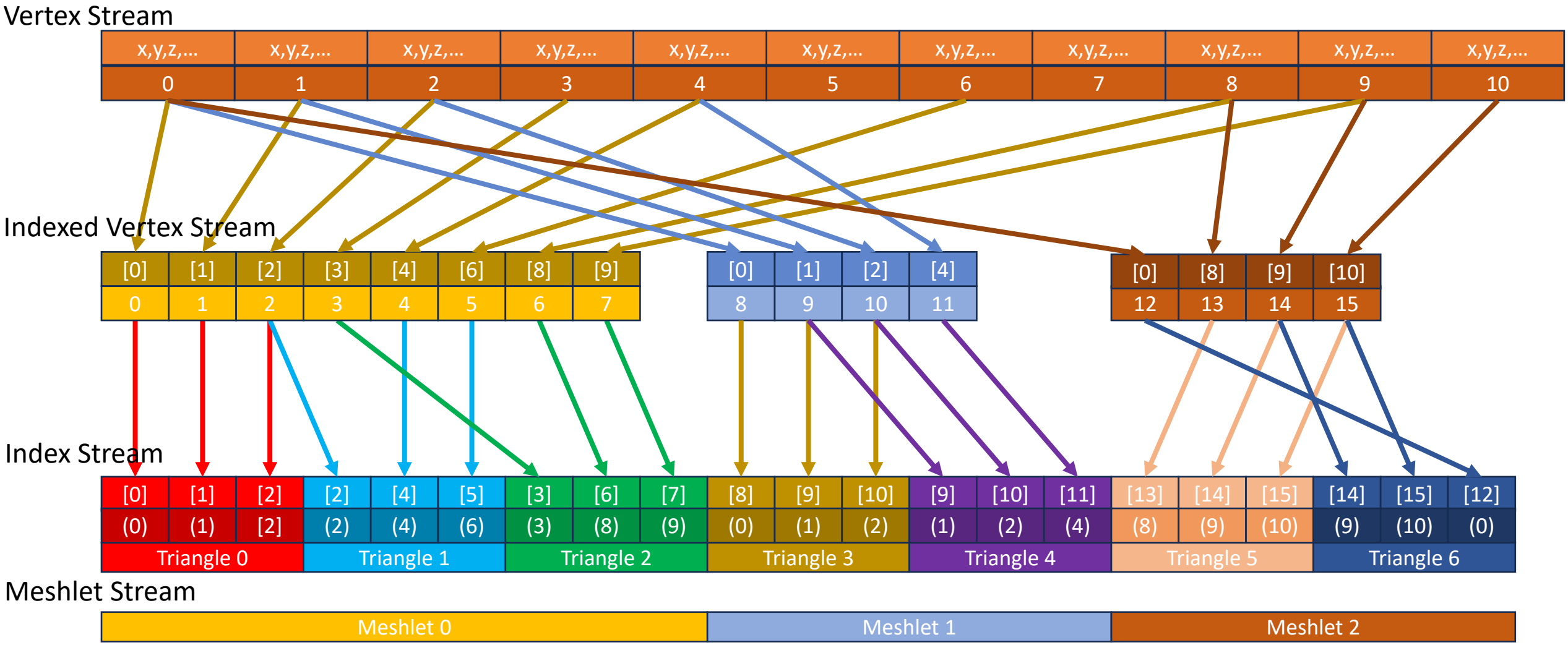
Meshlet

- Meshlet안에 직접 vertex data/Index data를 집어넣으면 GPU메모리 오버 헤드가 커진다
- VS파이프라인과 혼용할 경우 더욱 GPU메모리를 낭비하게 된다.
- 기존 Vertex Buffer는 그대로 두고
  - 일정 단위로(vertex/triangle 개수, 위치) 잘라서 삼각형 인덱스의 시작위치/ 점 인덱스의 시작 위치만 기록해 둔다.

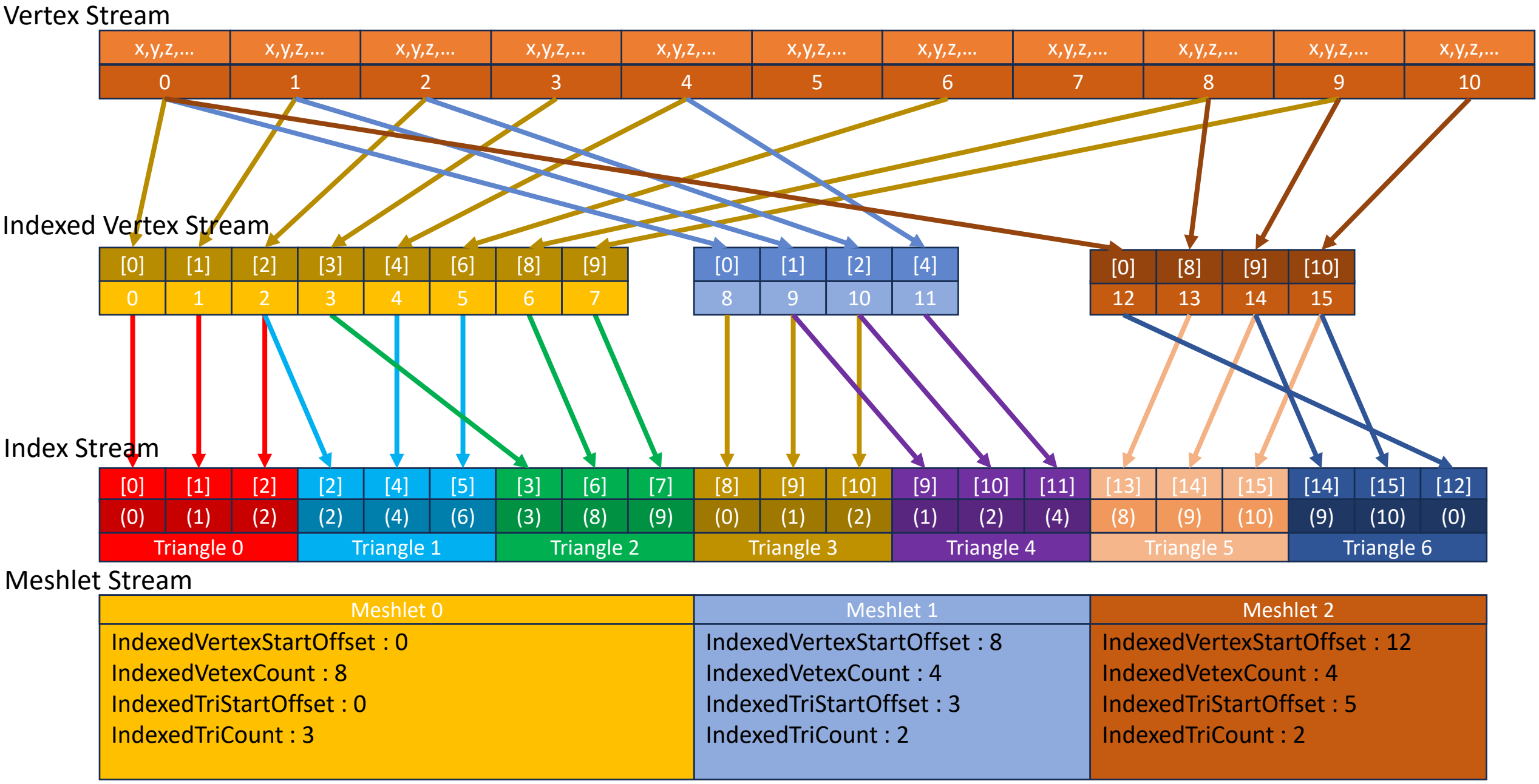
Vertex Buffer/Index Buffer

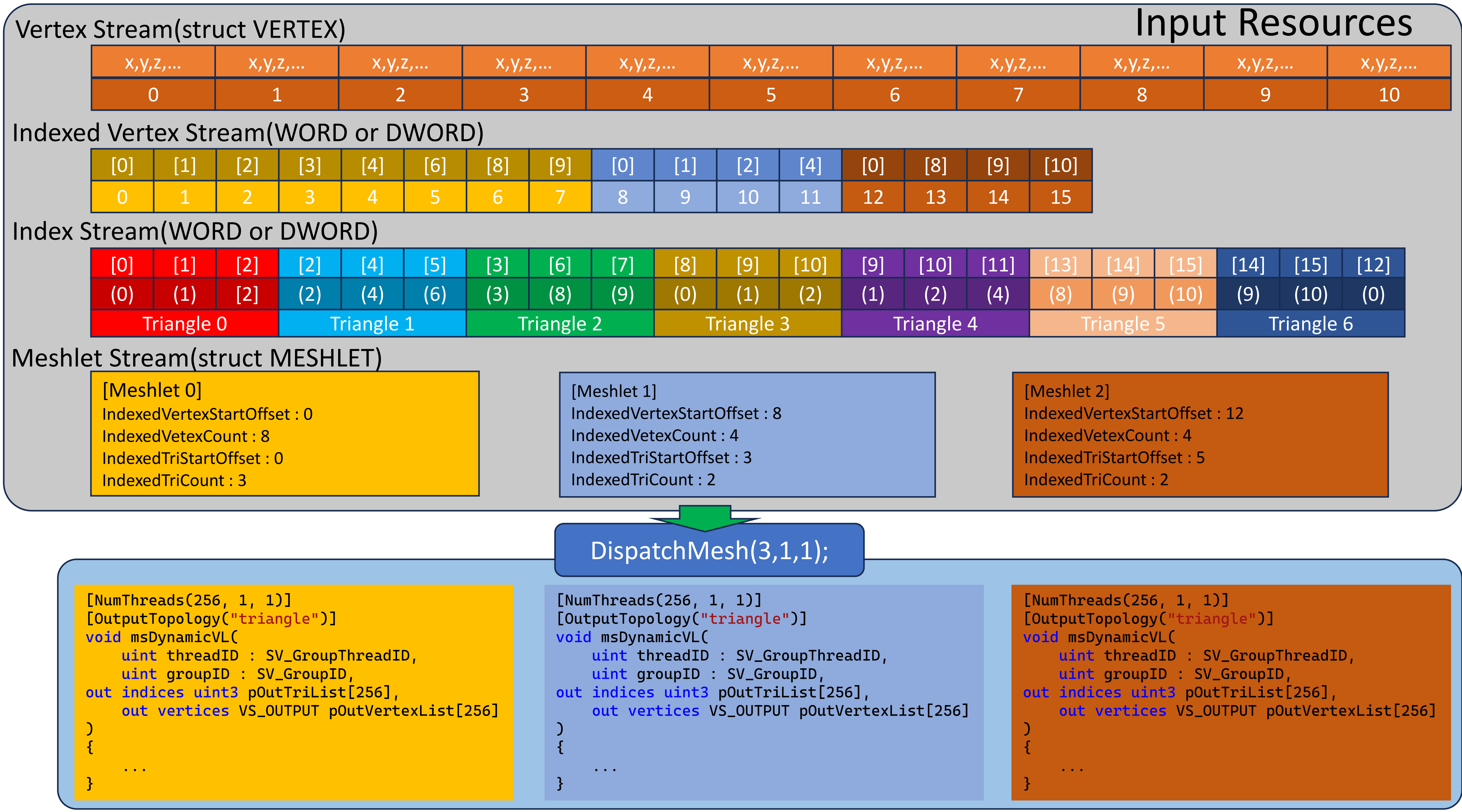


Meshlet


Vertex Stream
x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,...
0 1 2 3 4 5 6 7 8 9 10
Indexed Vertex Stream
[0] [1] [2] [3] [4] [6] [8] [9]
0 1 2 3 4 5 6 7
[0] [1] [2] [4]
8 9 10 11
[0] [8] [9] [10]
12 13 14 15
Index Stream
[0] [1] [2] [2] [4] [5] [3] [6] [7] [8] [9] [10] [9] [10] [11] [13] [14] [15] [14] [15] [12]
(0) (1) [2] (2) (4) (6) (3) (8) (9) (0) (1) (2) (1) (2) (4) (8) (9) (10) (9) (10) (0)
Triangle 0 Triangle 1 Triangle 2 Triangle 3 Triangle 4 Triangle 5 Triangle 6
Meshlet Stream
Meshlet 0 Meshlet 1 Meshlet 2

Meshlet


Vertex Stream
x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,... x,y,z,...
0 1 2 3 4 5 6 7 8 9 10
Indexed Vertex Stream
[0] [1] [2] [3] [4] [6] [8] [9]
0 1 2 3 4 5 6 7
[0] [1] [2] [4]
8 9 10 11
[0] [8] [9] [10]
12 13 14 15
Index Stream
[0] [1] [2] [2] [4] [5] [3] [6] [7] [8] [9] [10] [9] [10] [11] [13] [14] [15] [14] [15] [12]
(0) (1) (2) (2) (4) (6) (3) (8) (9) (0) (1) (2) (1) (2) (4) (8) (9) (10) (9) (10) (0)
Triangle 0
Triangle 1
Triangle 2
Triangle 3
Triangle 4
Triangle 5
Triangle 6
Meshlet Stream
Meshlet 0
IndexedVertexStartOffset : 0
IndexedVetexCount : 8
IndexedTriStartOffset : 0
IndexedTriCount : 3
Meshlet 1
IndexedVertexStartOffset : 8
IndexedVetexCount : 4
IndexedTriStartOffset : 3
IndexedTriCount : 2
Meshlet 2
IndexedVertexStartOffset : 12
IndexedVetexCount : 4
IndexedTriStartOffset : 5
IndexedTriCount : 2


Input Resources
Vertex Stream(struct VERTEX)
x,y,z,...
0 1 2 3 4 5 6 7 8 9 10
Indexed Vertex Stream(WORD or DWORD)
[0] [1] [2] [3] [4] [6] [8] [9] [0] [1] [2] [4] [0] [8] [9] [10]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Index Stream(WORD or DWORD)
[0] [1] [2] [2] [4] [5] [3] [6] [7] [8] [9] [10] [9] [10] [11] [13] [14] [15] [14] [15] [12]
(0) (1) [2] (2) (4) (6) (3) (8) (9) (0) (1) (2) (1) (2) (4) (8) (9) (10) (9) (10) (0)
Triangle 0
Triangle 1
Triangle 2
Triangle 3
Triangle 4
Triangle 5
Triangle 6
Meshlet Stream(struct MESHLET)
[Meshlet 0]
IndexedVertexStartOffset : 0
IndexedVetexCount : 8
IndexedTriStartOffset : 0
IndexedTriCount : 3
[Meshlet 1]
IndexedVertexStartOffset : 8
IndexedVetexCount : 4
IndexedTriStartOffset : 3
IndexedTriCount : 2
[Meshlet 2]
IndexedVertexStartOffset : 12
IndexedVetexCount : 4
IndexedTriStartOffset : 5
IndexedTriCount : 2
DispatchMesh(3,1,1);
[NumThreads(256, 1, 1)]
[OutputTopology("triangle")]
void msDynamicVL(
uint threadID : SV_GroupThreadID,
uint groupID : SV_GroupID,
out indices uint3 pOutTriList[256],
out vertices VS_OUTPUT pOutVertexList[256]
)
{
...
}

굳이 이중참조를 해야하는가?

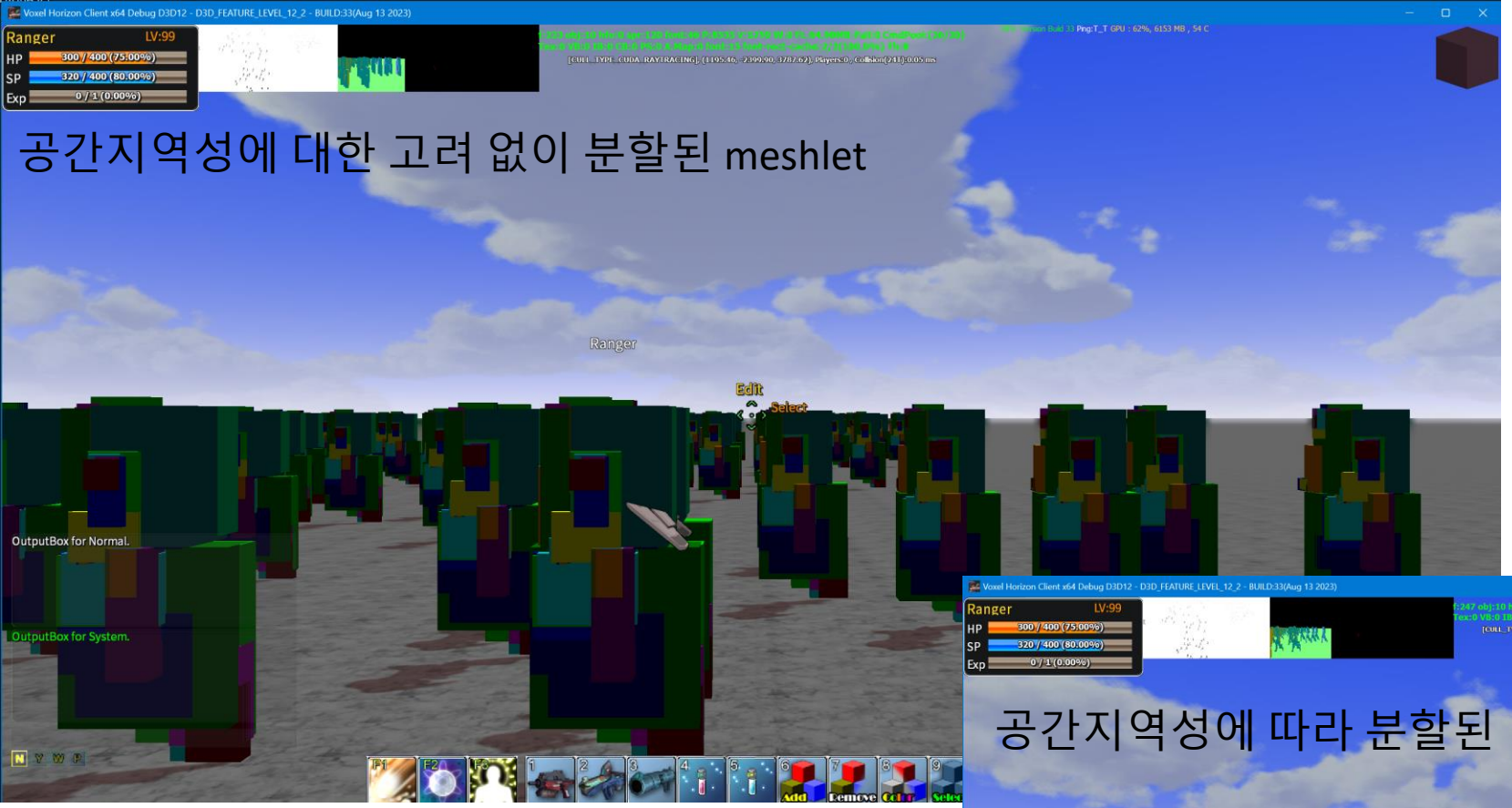
- 귀찮으면 간단하게 처리해도 됨. 그래도 어떻게든 meshlet으로 분할하긴 해야 함.
- 이중 참조를 하지 않으면…
  - 중복되는 Vertex 데이터로 GPU메모리 낭비
  - Vertex 가공(변환 등)할때 삼각형 단위로(점 3개씩) 처리해야함 -> 성능저하
  - Vertex 데이터의 변환/저장이 중복해서 발생함 -> 성능저하
- VS/GS파이프라인과 Vertex Stream을 공유할 수 있다.
- Meshlet에서 필요한 버텍스 데이터만 packing되어있으므로 별도 가공시 필요한 shared memory사이즈를 제한할 수 있음.

공간 지역성에 따라 meshlet으로 분할

- 어차피 meshlet으로 쪼갤거면 인접한 삼각형들끼리 묶어주자.
- 인접한 삼각형들끼리 묶지 않으면 meshlet들이 상호 교차하게 된다.
- Meshlet의 전체적인 모양이 균일해야 가공(테셀레이션 등)하기 좋다.
- Meshlet들 간에 상호 교차하지 않아야 culling 효율이 좋아진다.

공간지역성에 대한 고려 없이 분할된 meshlet

공간지역성에 따라 분할된 meshlet

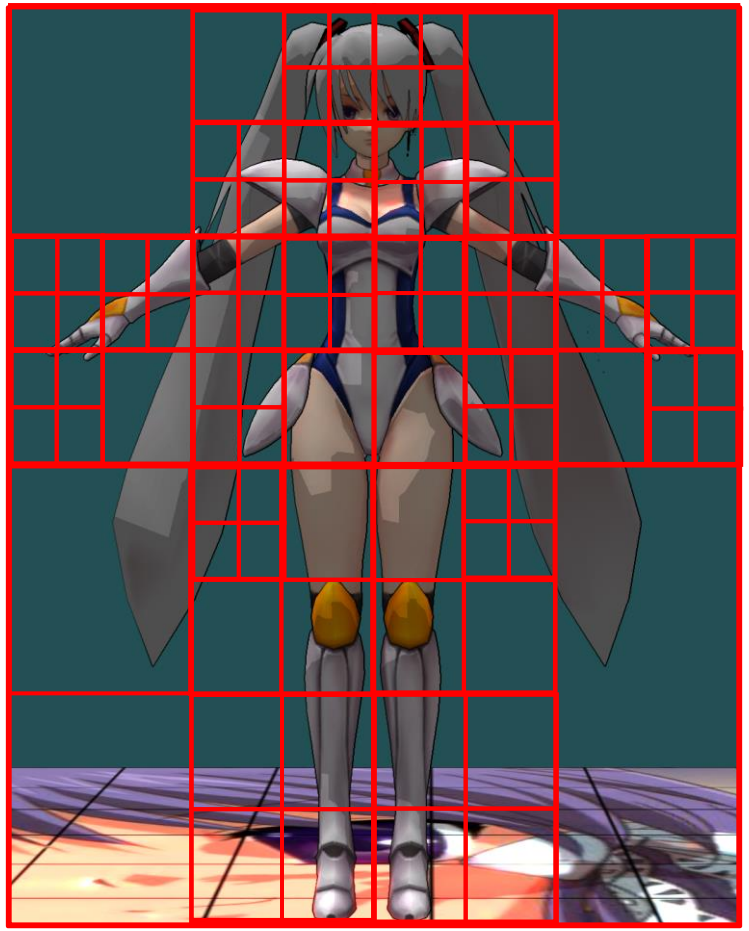


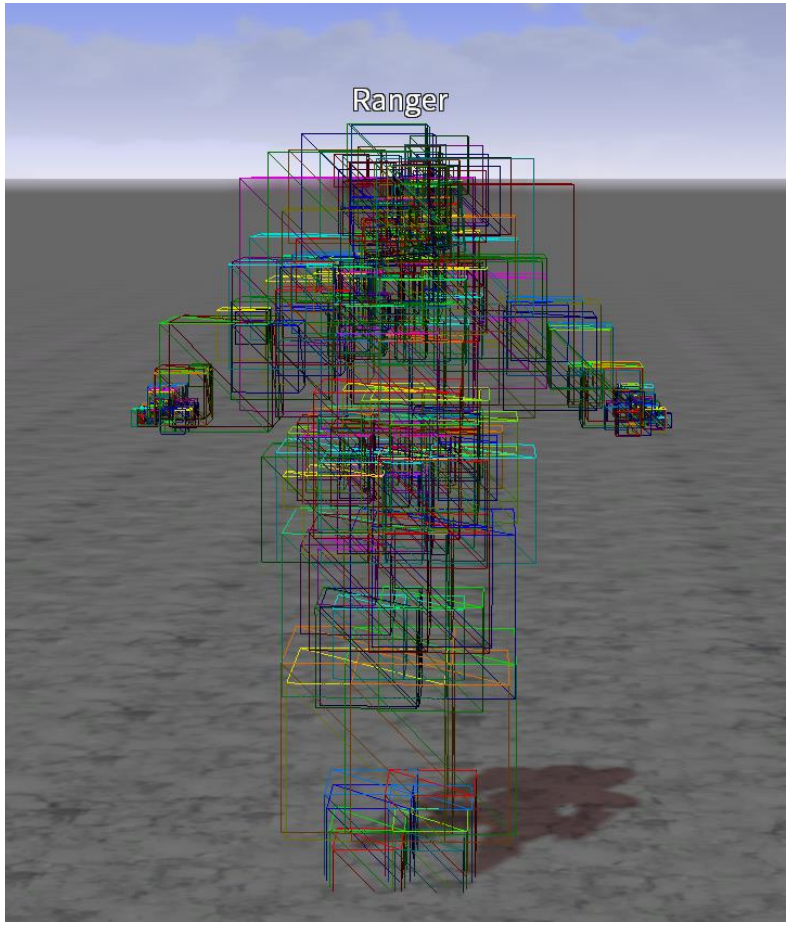
- 다른 meshlet과 겹치지 않는다.
- Culling효율이 좋다.
- 기하학적으로 가공하기에 좋다.

- 임의의 meshlet이 다른 meshlet과 겹친다.
- Culling효율이 떨어진다.
- 기하학적으로 가공하기에 좋지 않다.

Tree를 이용한 분할

1. node(leaf)에 포함된 Indexed Triangle과 Vertex의 개수가 기준치 이하가 될 때까지 공간을 잘라 나간다.
2. OCTree/ KD Tree등을 사용할 수 있다.
3. Tree빌드 후 각 leaf의 Indexed Triangle과 vertex 목록으로 meshlet을 만든다.





Instancing

- 아마도 Mesh Shader사용에 가장 적합한 기능
- Groups = Meshlet x 인스턴스 개수
  - DispatchMesh(Groups, 1, 1);
  - InstanceID = GroupID / ObjCountPerMeshlet;
  - MeshletID = GroupID % ObjCountPerMeshlet;

Mesh Shader 단독 사용의 제약

- DispatchMesh()에 전달할 group개수는 65536개가 최대이다.
- 100개의 meshlet으로 이루어진 오브젝트를 1000개 인스턴싱하면 DispatchMesh(100x1000)이 되어 호출에 실패한다.
- DispatchMesh()를 쪼개서 호출해야 되네?

Amplification Shader

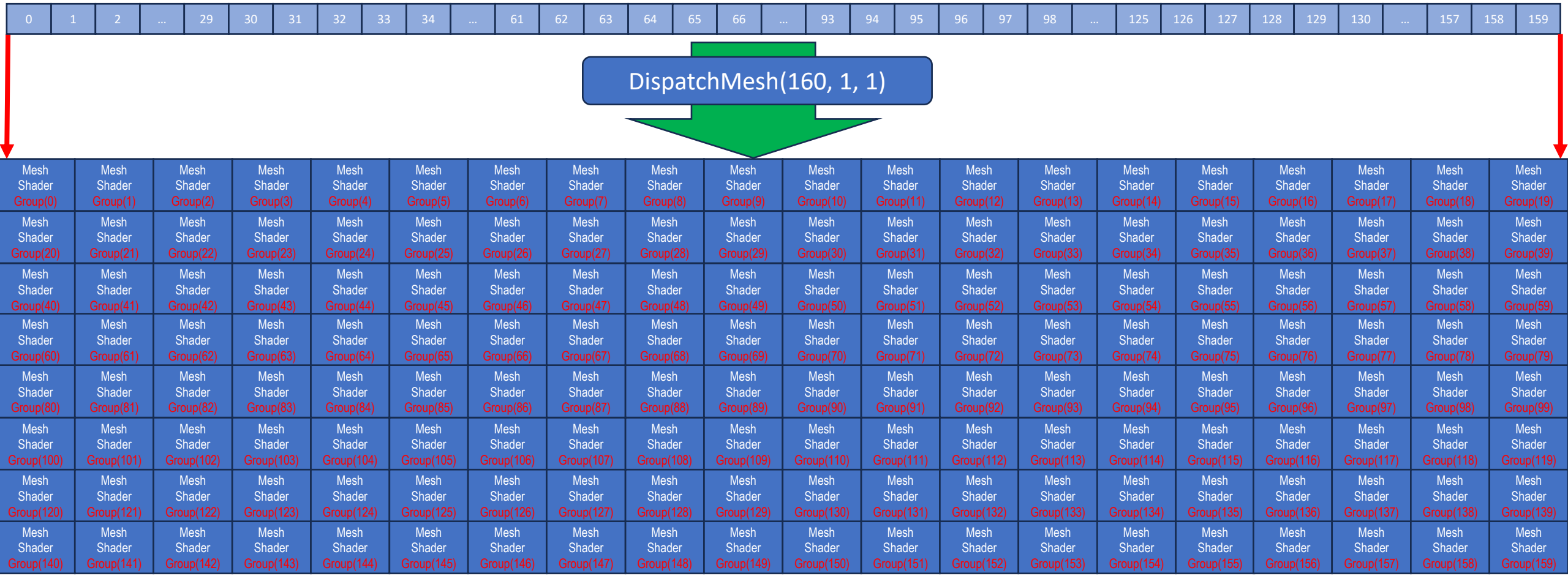
- 어차피 쪼개서 호출해야 한다면 meshlet들을 특정 조건으로 필터링해서 호출할 수 있으면 좋겠다.
- 이 작업을 GPU안에서 할 수 있다면?
  - 특정 조건을 만족하는 meshlet만 골라서 다수의 DispatchMesh()에 나눠서 호출하는 코드를 GPU에서 실행할 수 있을까?
- 각 meshlet에 대해서 어떻게 처리할지 별도의 파라미터를 전달하면 더 많은 일을 할 수 있다.

Amplification Shader

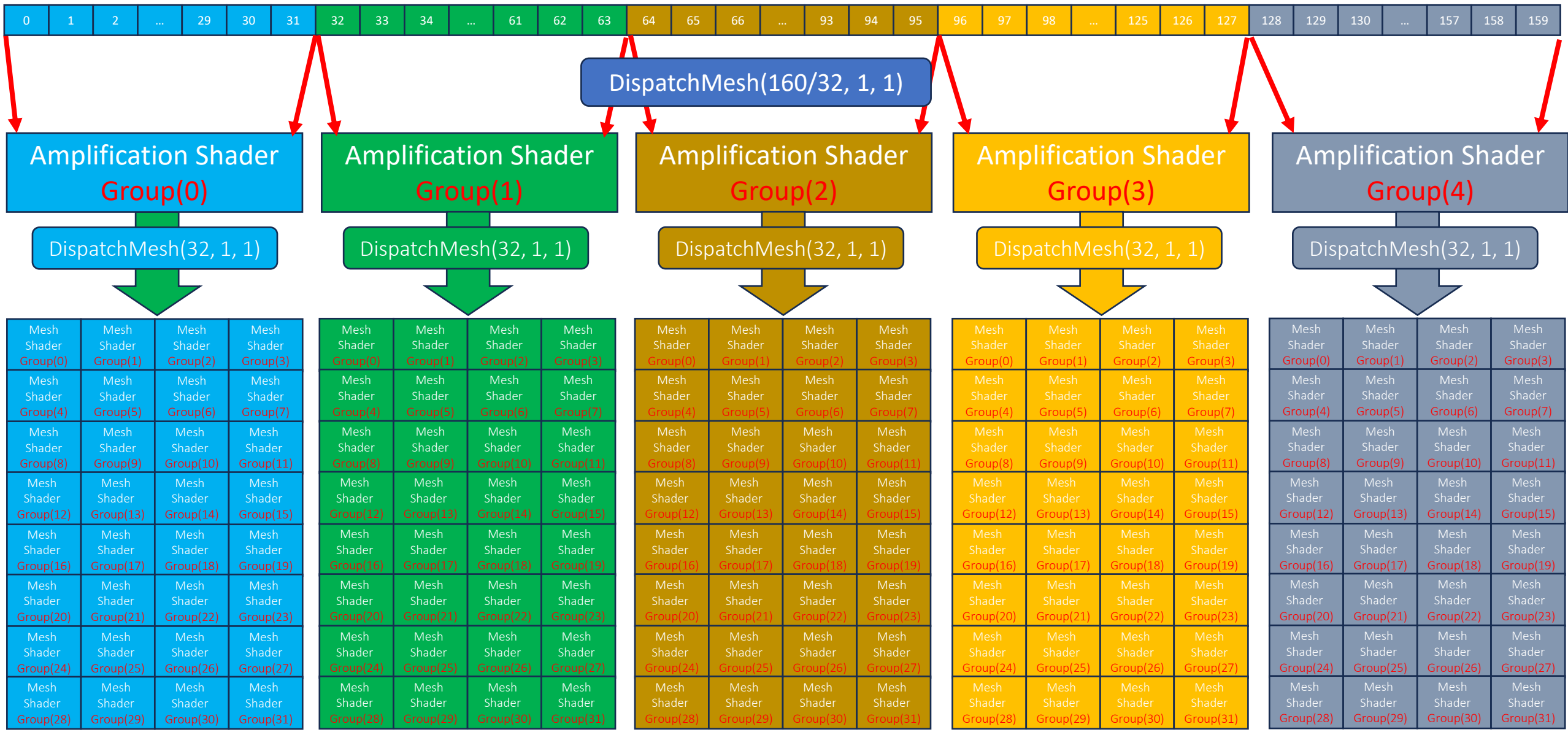
- 기본적으로 Compute Shader.
- Mesh Shader- 최종적으로 삼각형 집합을 출력.
- Amplification Shader – 최종적으로 DispatchMesh()를 호출.
- Mesh Shader에 전달할 데이터를 선별(meshlet 선별)
- 어떻게 처리할지 파라미터를 전달
- Meshlet Culling, Tessellation 등에 사용

Mesh Shader 단독 사용

Meshlet Stream



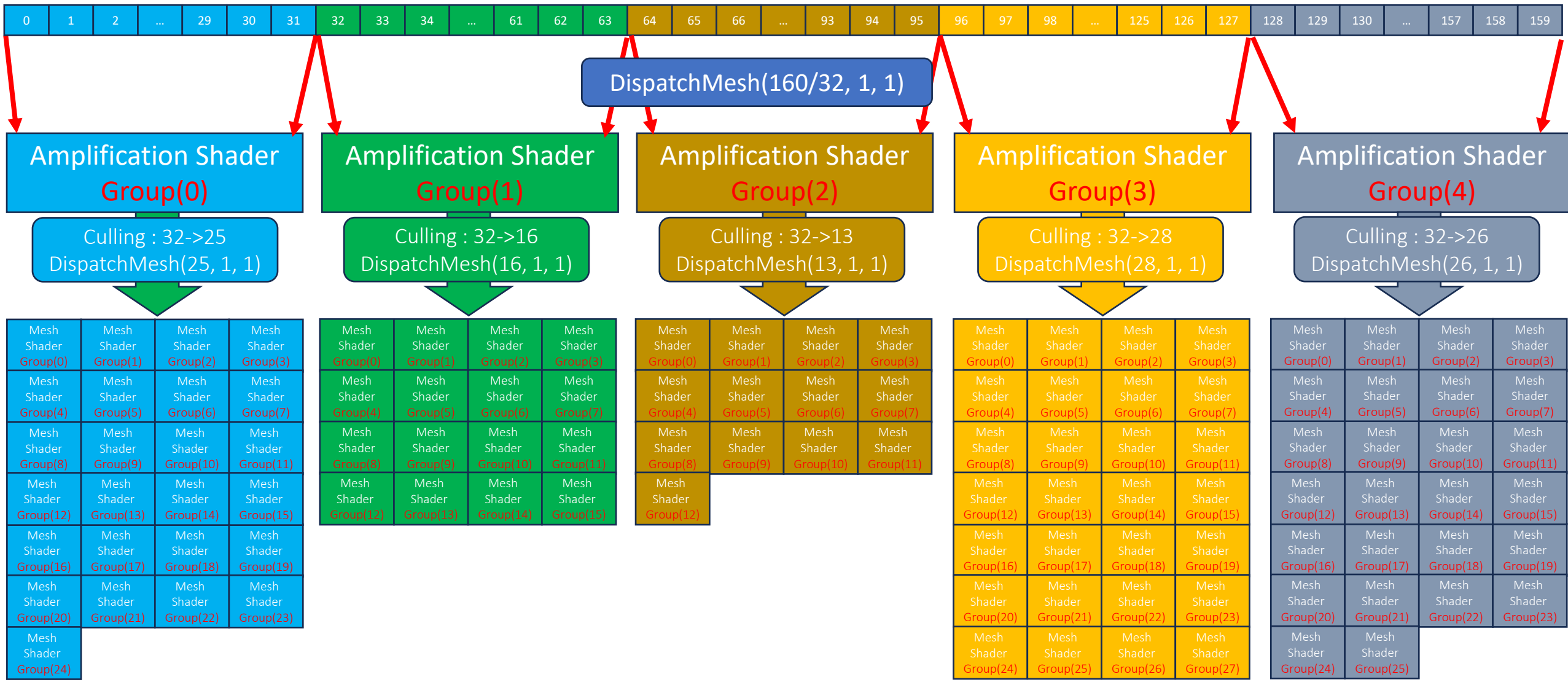
Amplification Shader + Mesh Shader



Meshlet Culling

- View Frustum을 이루는 6개의 평면을 전달
- Meshlet생성시 AABB or Bounding Sphere를 만들어 둠
- Amplification Shader에서 View Frustum – Bounding Mesh로 Culling
- 보여진다고 판단되는 meshlet의 인덱스만 payload에 저장해서 Mesh Shader로 전달

Culling with Amplification Shader

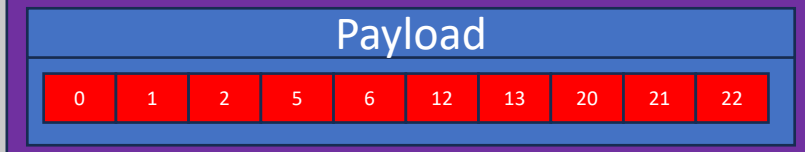


Amplification Shader

Meshlet stream

Culling Meshlet(0 ~ N-1) by thread (0 ~ N-1)

Pack & Save to Payload



DispatchMesh(10, 1, 1, Payload)

Mesh Shader

```
struct Payload
{
    uint MeshletID[AS_GROUP_SIZE];
};

groupshared Payload s_Payload;

[NumThreads(AS_GROUP_SIZE, 1, 1)]
void asDefault(uint gtid : SV_GroupThreadID, uint DispatchID : SV_DispatchThreadID, uint groupID : SV_GroupID)
{
    bool visible = false;
    uint meshltID = DispatchID % MeshletCountPerObj;
    MESHLET meshlet = g_MeshletBuffer[meshltID];

    float4 SpherePosWorld = mul(float4(meshlet.Bounds.xyz, 1), matWorld);
    visible = CullSphere(g_FrustumPlanes, SpherePosWorld.xyz, meshlet.Bounds.w) != 0;
    if (visible)
    {
        uint index = WavePrefixCountBits(visible);
        s_Payload.MeshletID[index] = meshltID;
    }

    uint visibleCount = WaveActiveCountBits(visible);
    DispatchMesh(visibleCount, 1, 1, s_Payload);
}
```

GPGPU의 warp(wave)특성을 이용한 함수

uint WavePrefixCountBits(bool bBit)
현재 lane보다 인덱스가 작은 모든 활성 lane에서 true로 설정된 지정된 모든 bool 변수의 합계를 반환합니다.

https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/waveprefixcountbytes

uint WaveActiveCountBits(bool bBit)
현재 wave의 모든 활성 lane에서 true로 평가되는 bool 변수의 수를 세고 그 결과를 wave의 모든 lane에 복제합니다.

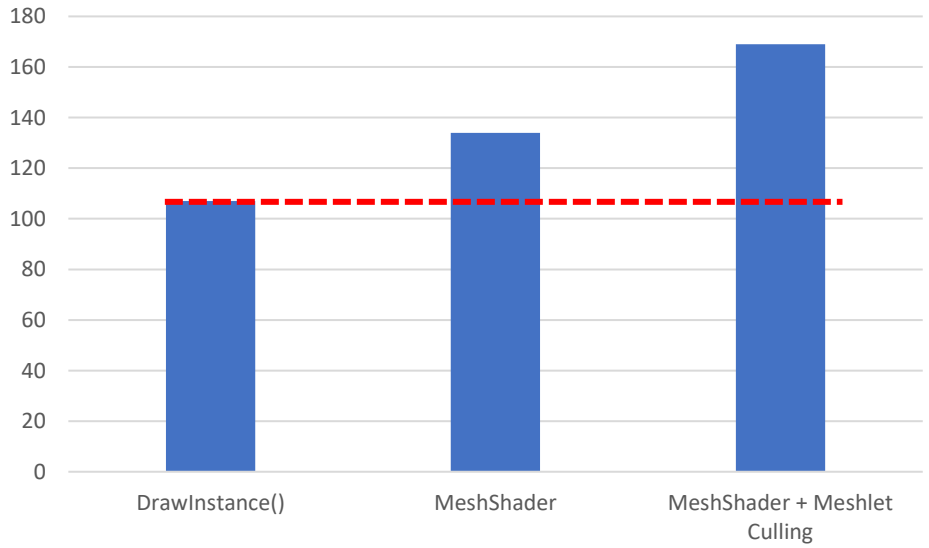
https://learn.microsoft.com/en-us/windows/win32/direct3dhlsl/waveactivecountbits

코드 최적화

- shared memory는 꼭 필요한 경우에만 사용한다.
- GPGPU의 context switching은 메모리에 backup/restore하지 않으므로 동시에 스케쥴링될 수 있는 warp수는 사용된 shared memory 사이즈와 register수에 따라 의해 결정된다.
- Vertex를 가공할 때는 삼각형 단위(점3개)가 아닌 meshlet으로 주어진 vertex(점 1개) 단위로 가공한다. 중복처리 금지.

성능비교- 캐릭터 인스턴싱 테스트

- intel i7 13700K , RTX 3070
- 해상도 2860x1537 , 캐릭터 1 -> 1024마리 인스턴싱



DrawInstance()	MeshShader	Mesh Shader + Meshlet Culling
107 FPS	134 FPS	169 FPS


Voxel Horizon Client x64 Release D3D12 - D3D_FEATURE_LEVEL_12_2 - BUILD:33(Aug 16 2023)
Ranger LV:99
HP 300 / 400 (75.00%)
SP 320 / 400 (80.00%)
Exp 0 / 1 (0.00%)
Ranger
Edit
Select
OutputBox for Normal.
OutputBox for System.
F1 F2 F3 1 2 3 4 5 6 Add 7 Remove 8 Edit 9 Select F7 HP


Voxel Horizon Client x64 Release D3D12 - D3D_FEATURE_LEVEL_12_2 - BUILD:33(Aug 16 2023)
Ranger LV:99
HP 300 / 400 (75.00%)
SP 320 / 400 (80.00%)
Exp 0 / 1 (0.00%)
Ranger
Edit
Select
OutputBox for Normal.
OutputBox for System.
Voxel Edit [Select]
Fill
[60]
Select Color: (←, →, ↑, ↓) keys
Change brush size: [,] keys or Mouse wheel L / R
Transform
Footrest
CTRL + ↑, ↓
Show
F1 F2 1 2 3 4 5 6 Add 7 Remove 8 Edit 9 Select F7 HP F8 F9

참고자료

- MS공식 영상
  - https://youtu.be/CFXKTXtil34

- MS 공식 샘플코드
  - https://github.com/microsoft/DirectX-Graphics-Samples/tree/master/Samples/Desktop/D3D12MeshShaders