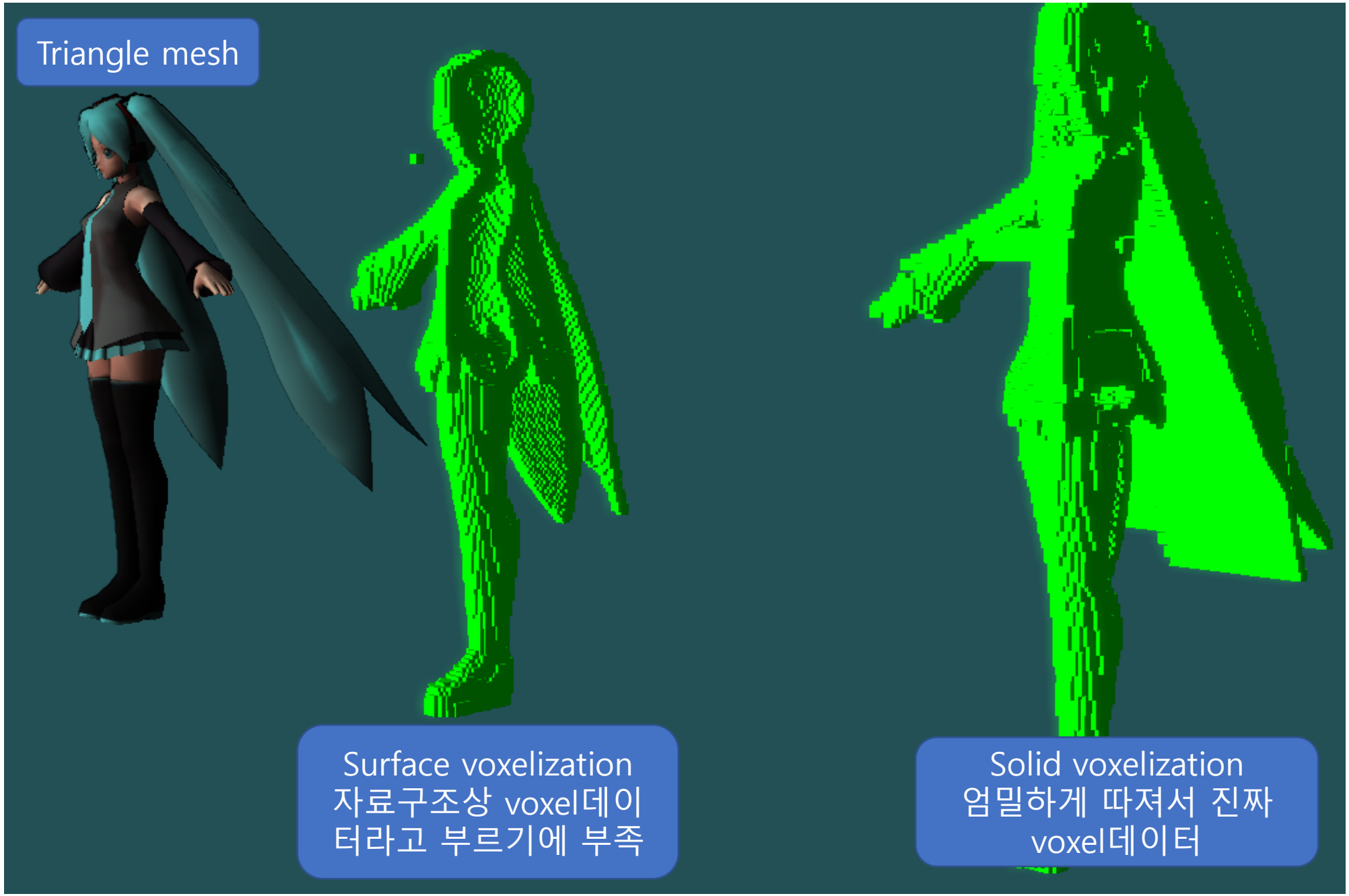
캐릭터 모델 실시간 Voxelization

유영천

Tw:@dgtman

https://megayuchi.com


Triangle mesh
Surface voxelization
자료구조상 voxel데이
터라고 부르기에 부족
Solid voxelization
엄밀하게 따져서 진짜
voxel데이터

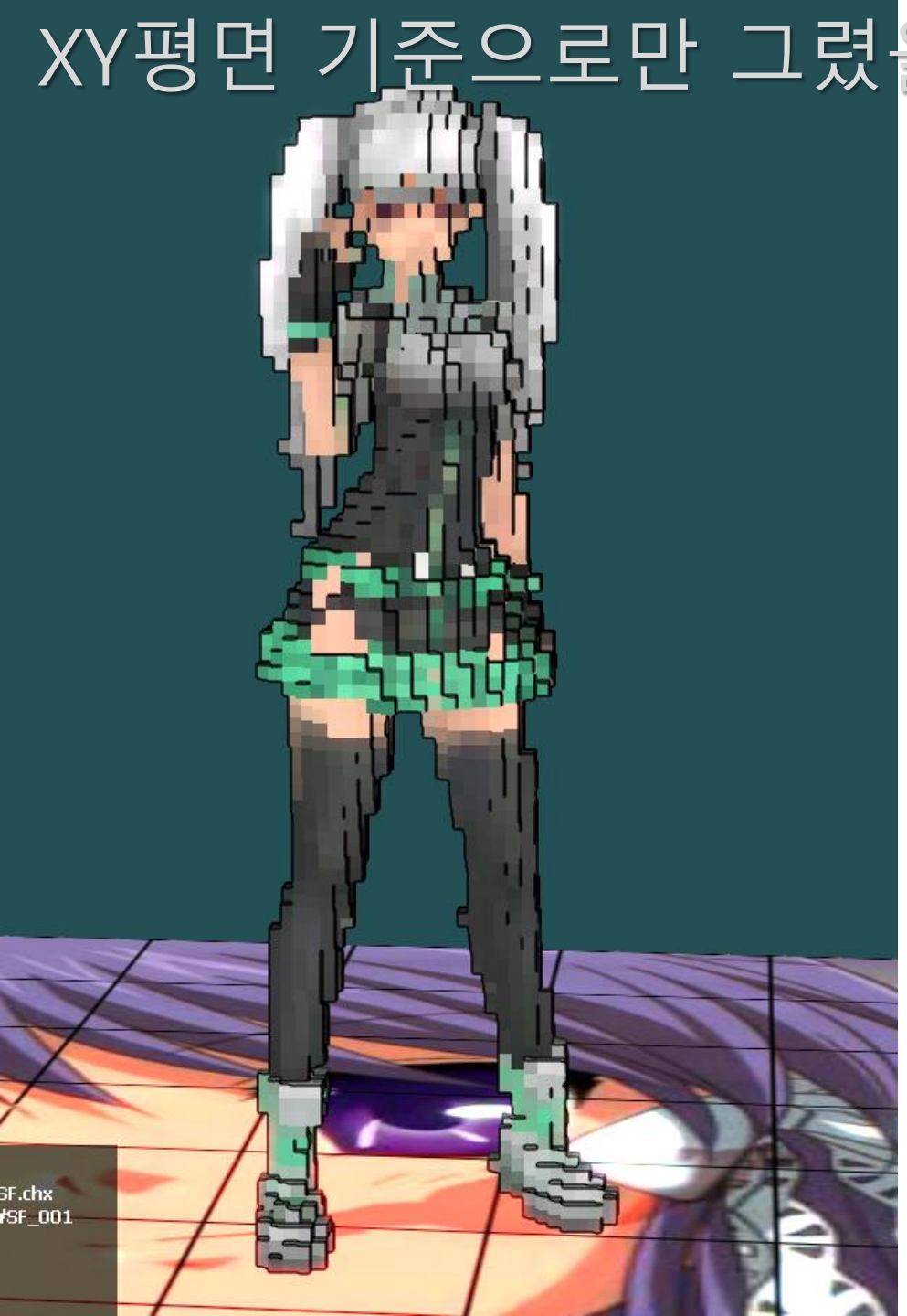
실시간 voxelization

- 실시간으로 정확하게 삼각형 매시의 안쪽을 채우기는 어렵다.
- 3D상에서 레트로 느낌 구현.
- 정확히는 voxel매시처럼 '보이게' 한다.
- Surface Voxelization에 해당한다.

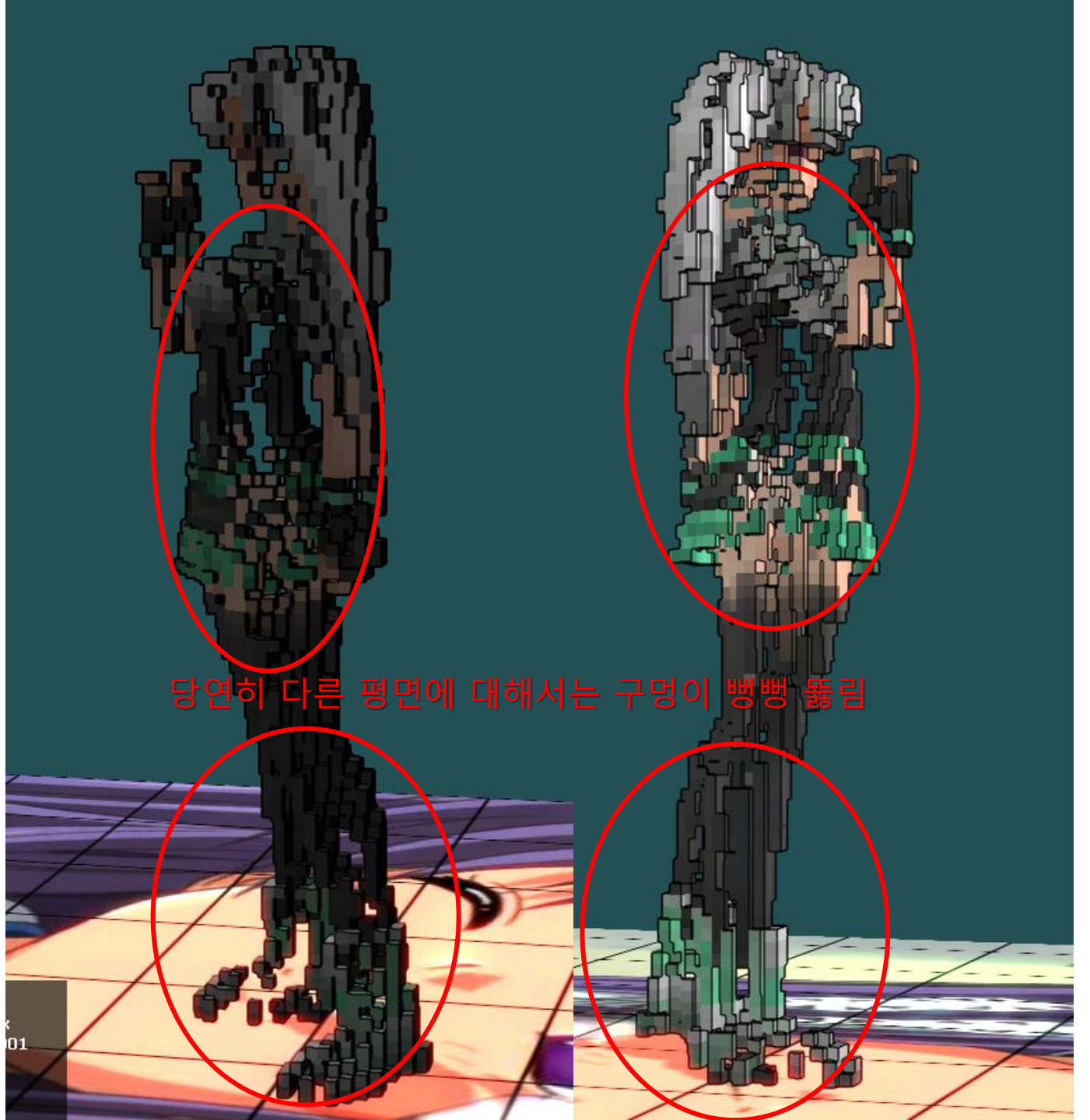
기본원리

- 모자이크화 – 2D에서의 양자화 – 해상도를 떨어뜨림
- 복셀화 – 3D에서의 양자화 – 해상도를 떨어뜨림
- 컴퓨터 그래픽스 자체가 이미 양자화된 화면을 보여주고 있다. 해상도가 충분히 높을 뿐이지
- 카메라가 고정되어 있다면 2D나 3D나 똑같다.
- 그러나 3D에서 카메라가 고정되어 있을리가 없으므로 입체를 구성해야 한다.
- 2D Render Target대신 3D Texture에다 그린다.
- z값은 3D텍스처의 w(u,v,w중 w)좌표로 사용한다.

XY평면 기준으로만 그렸을때



당연히 다른 평면에 대해서는 구멍이 뻥뻥 뚫림



ZY평면 기준으로만 그렸을때

XY,XZ 평면에 대해서는 구멍이 뻥뻥 뚫림

(xy plane -> 3D Texture) + (zy plane -> 3D Texture)
+ (xz plane -> 3D Texture) = full voxelized mesh

완전한 렌더링을 위해서

- xy, xz, zy평면에 대해 총 3번 렌더링 하면 구멍이 뚫리지 않는다. RenderTarget으로 2D텍스처를 사용하는 대신, UAV로 3D텍스처를 사용한다.
- 3번의 draw call은 꽤 낭비가 크다. Geometry Shader에서 한번에 3개 평면에 대해 렌더링 하자.

잠깐 사전 지식 점검

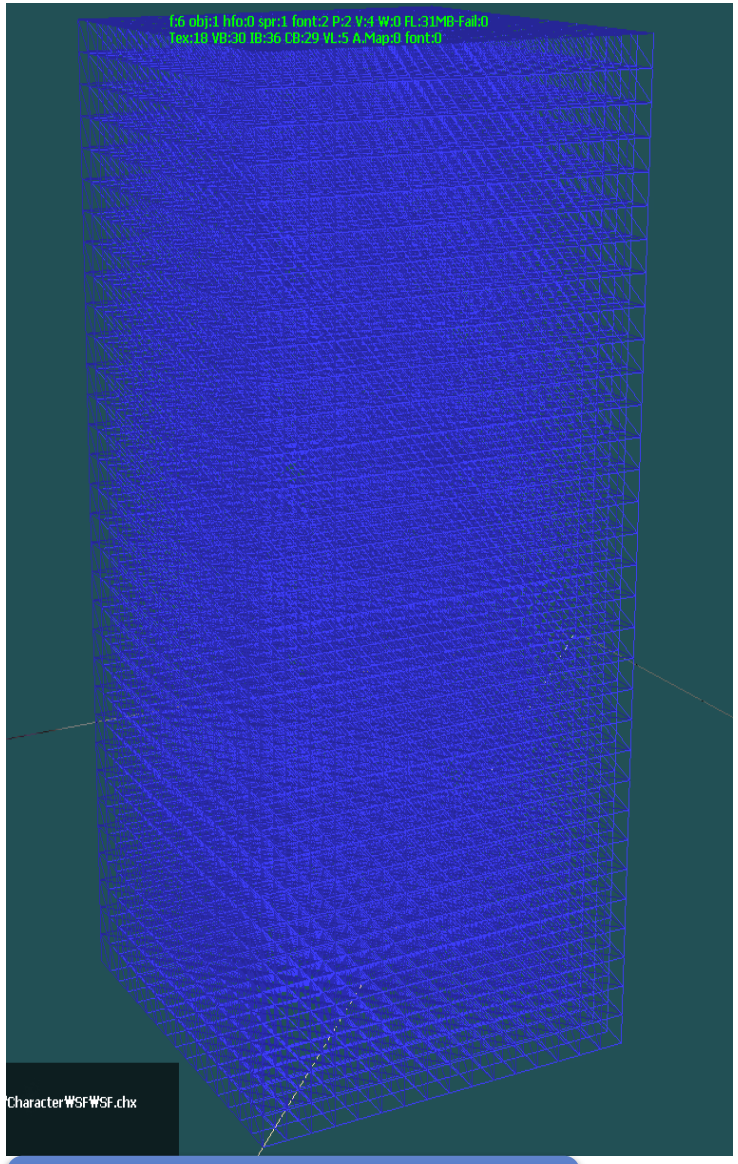
- SRV – Shader Resource View
  - Texture메모리(물리적으로는 그냥 GPU측 메모리)
  - Vertex Shader, Pixel Shader, Compute Shader
  - 읽기 전용
- UAV – Unordered Access View
  - Texture or 범용메모리(물리적으로는 그냥 GPU측 메모리)
  - Vertex Shader, Pixel Shader, Compute Shader
  - 읽기/쓰기 가능
- RTV – Render Target View
  - Texture메모리
  - Pixel Shader
  - 쓰기 전용
  - 랜덤 억세스 불가능 – 정확히는 Pixel Shader에서 기록할 위치(주소)를 컨트롤할 수 없다.

직교 투영 행렬

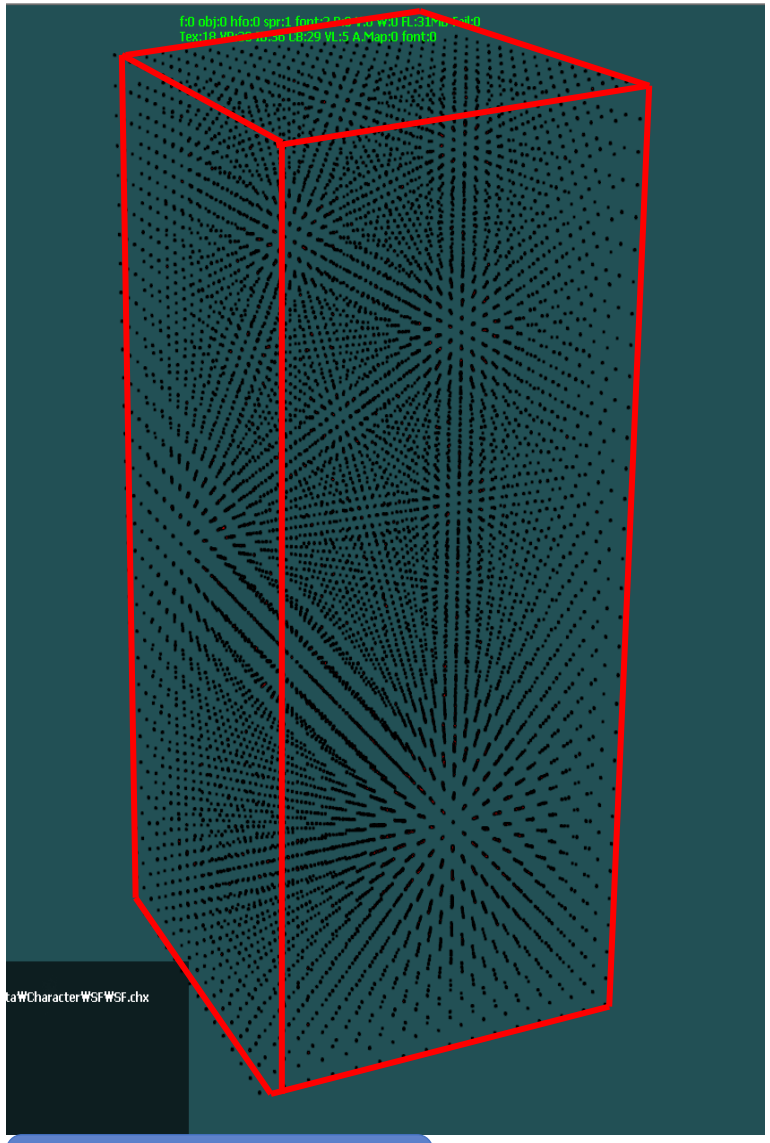
- 당연한 얘기지만 3개 축 방향에 대해서 렌더링한 결과를 모두 합산할 것이므로 원근이 있어서는 안된다.
- 변환 후 화면 안에 들어가는 점의 범위는
  - -1 <= x/w <= 1 , -1 <= y/w <= 1 , 0 <= z/w <= 1
- 직교 투영 행렬의 경우 w=1 이므로
  - -1 <= x <= 1 , -1 <= y <= 1 , 0 <= z <= 1

렌더링 하기 전에

- 3D텍스처 준비
- 3D텍스처의 width x depth x heigh에 대응하는 vertex 배열 준비
- XY, XZ, ZY평면에 렌더링 하기 위해 다음의 요소를 미리 준비
  - Viewport 3개
  - View x projection 행렬 3개



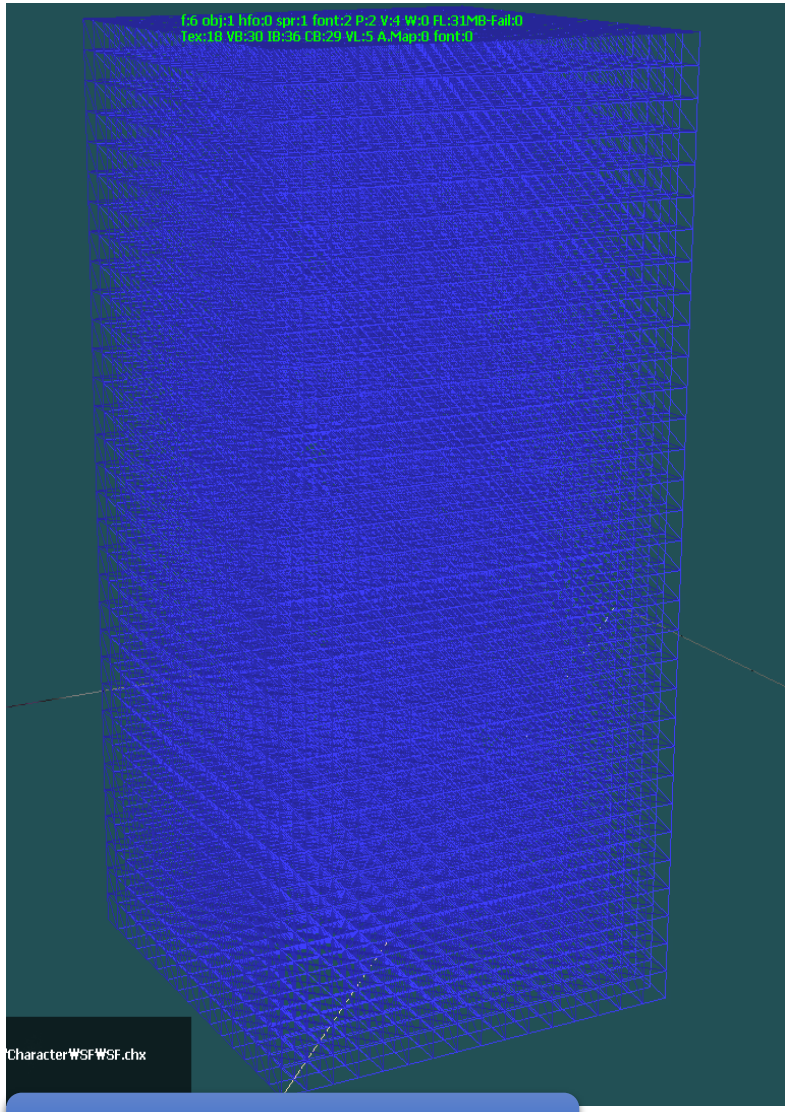
3D Texture(UAV)



Vertex Buffer

1st pass

- 3D Texture를 UAV로 전달
- 일반 3D모델과 똑같은 렌더링 과정을 거쳐 Vertex Shader호출
- Vertex Shader -> Geometry Shader
- Geometry Shader에서 3개의 행렬에 대해 변환후 각각의 SV_ViewportArrayIndex로 출력
- Pixel Shader에서 Render Target이 아닌 UAV로 전달받은 3D텍스처에 써넣기.
- 행렬 변환 후 화면 내의 점은 x= (-1 – 1), y = (-1 – 1), z = (0, 1)의 값을 가지게 된다.
- 이것으로 3D Texture의 texel위치를 구한다.

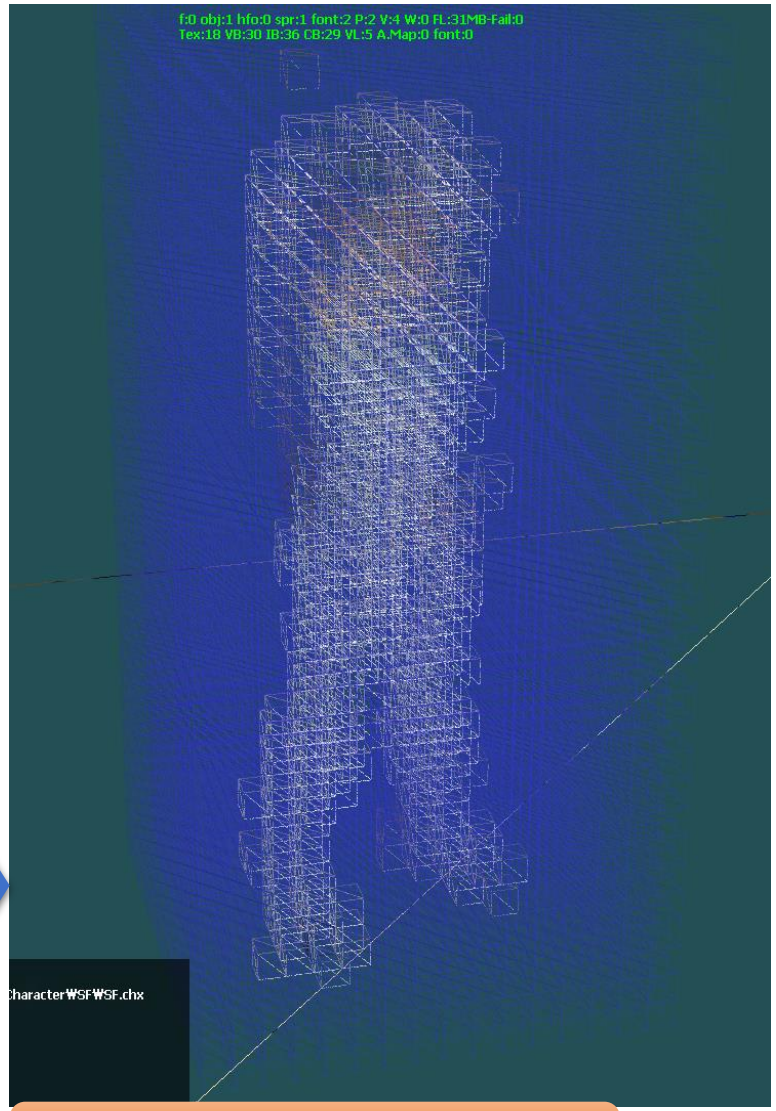


Input - 3D Texture(UAV)

Vertex Buffer Texture

Draw

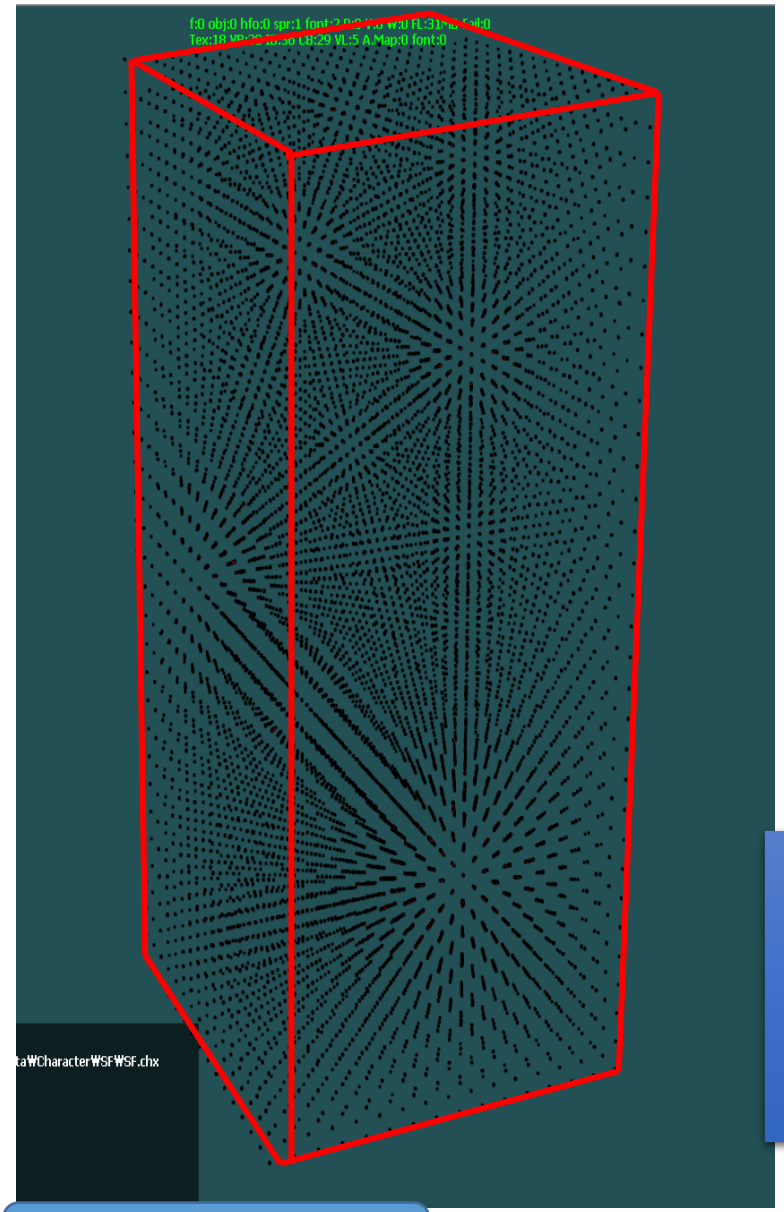
Vertex Shader
Geometry Shader
Pixel Shader



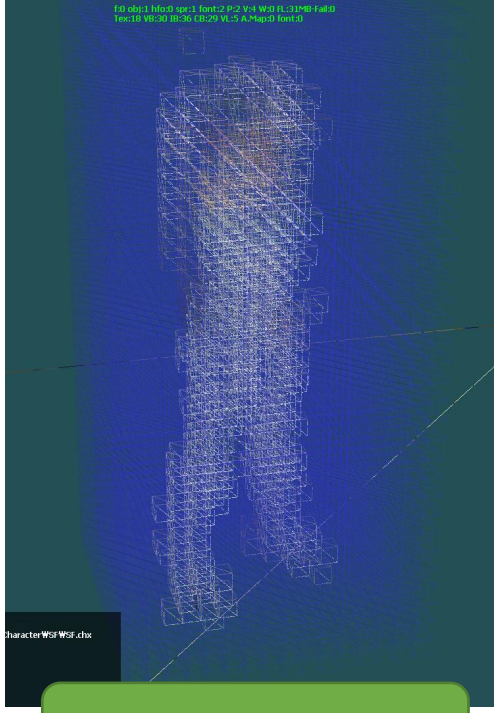
Output – 3D Texture

2nd pass

- 3D Texture의 width x depth x heigh에 대응하는 Vertex Buffer를 Point list로 렌더링(D3D11_PRIMITIVE_TOPOLOGY_POINTLIST)
- Vertex shader - 각 vertex에 대응하는 3D텍스처의 텍셀을 읽어서 컬러값으로 사용
- Geometry shader - 각 vertex를 6면체로 변환
- Pixel shader에서 최종 출력

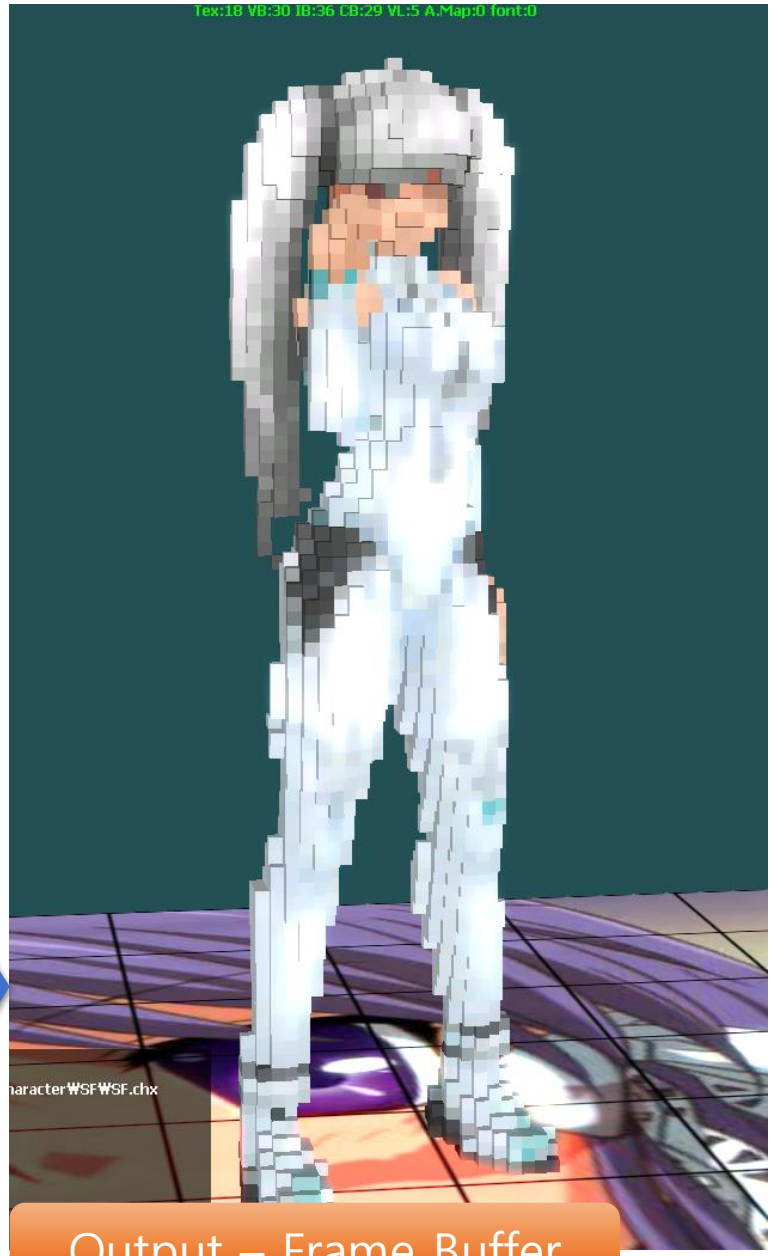


Vertex Buffer



3D Texture(SRV)

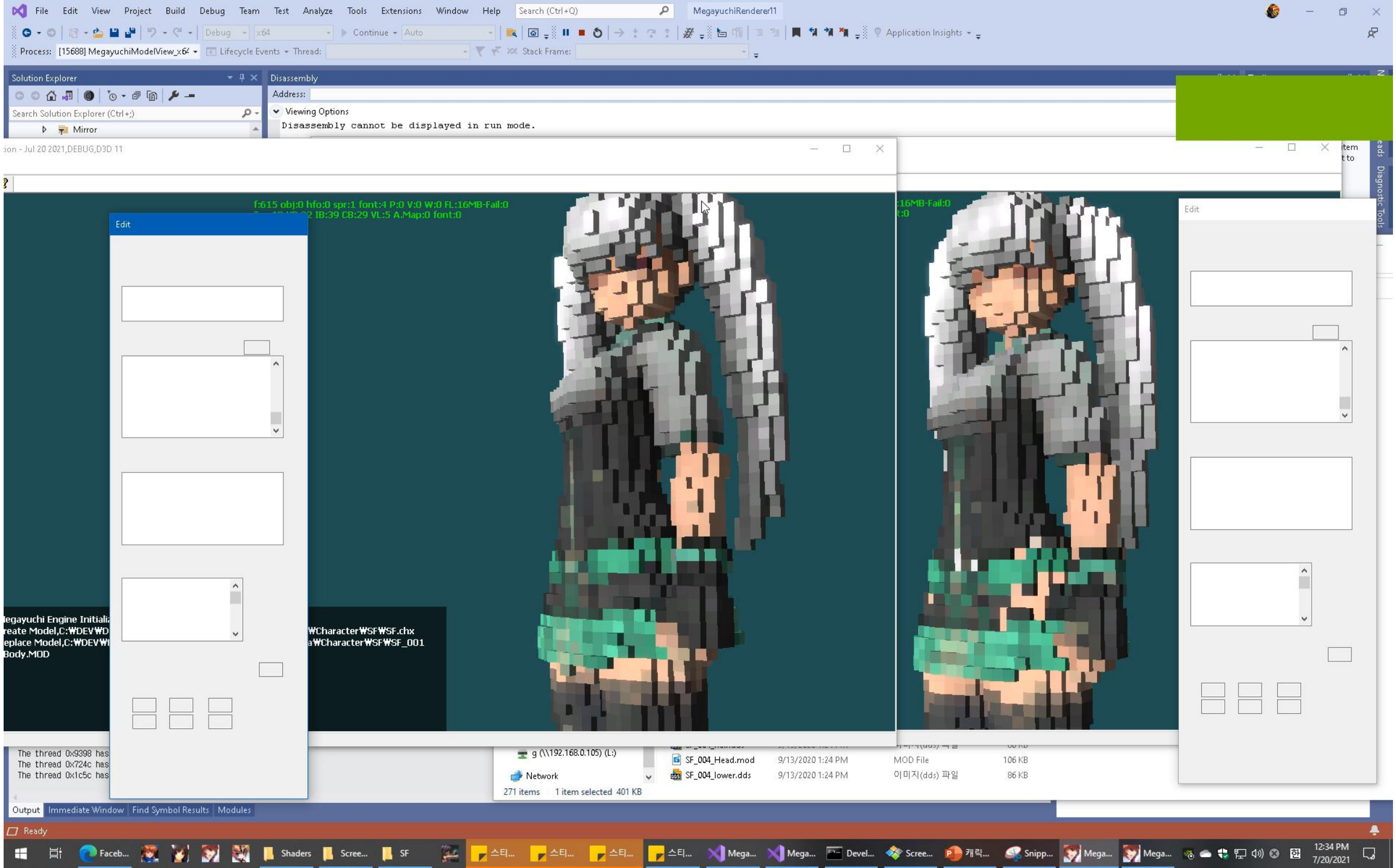
Vertex Shader
Geometry Shader
Pixel Shader



Output – Frame Buffer

깜빡임 문제

- 3D 텍스처에 써 넣을때 3개 면에 대해서 동시에 렌더링 하므로 같은 픽셀에 동시에 여러 개의 GPU스레드가 접근 가능.
- 매 프레임마다 스레드의 접근 순서가 바뀌므로 깜빡임 발생.
- Pixel Synchronization이 필요하다.


File Edit View Project Build Debug Team Test Analyze Tools Extensions Window Help
MegayuchiRenderer11
Solution Explorer
Disassembly
Viewing Options
Disassembly cannot be displayed in run mode.
Edit
f:615 obj:0 hfo:0 spr:1 font:4 P:0 V:0 W:0 FL:16MB-Fail:0
IB:39 CB:29 VL:5 A.Map:0 font:0
SF_004_Head.mod 9/13/2020 1:24 PM MOD File 106 KB
SF_004_lower.dds 9/13/2020 1:24 PM 이미지(dds) 파일 86 KB
271 items 1 item selected 401 KB
Output Immediate Window Find Symbol Results Modules
Ready

Pixel Synchronization

- HLSL의 Interlockedxxxx()를 사용할 수 있지만 텍스처는 RGBA 4 성분을 가지고 있으므로 CAS에서 비교기준이 모호함.
- 일관된 CAS 규칙을 유지하기 위해서 RGBA성분을 합쳐서 밝기로 만들고 A성분에 넣음.
- RGBA의 메모리 순서에서 A가 최상위 바이트이므로 InterlockedMax()함수로 일관되게 픽셀을 선택할 수 있다.

Pixel Synchronization

- 일관된 CAS 규칙을 유지하기 위해서 r,g,b성분을 합쳐서 밝기로 만들고 a성분에 넣음.
- RGBA의 메모리 순서에서 a가 최상위 바이트이므로 함수로 일관되게 픽셀을 선택할 수 있다. InterlockedMax()

```
    // 가장 밝은 픽셀을 선택하기 위해 최상위 8비트에 밝기값을 배치한다. 민감한 green을 그 다음에 배치한다.
    float bw = outColor.r * 0.3 + outColor.g * 0.59 + outColor.b * 0.11;
    uint a = (uint)(saturate(bw) * 255.0);
    uint r = (uint)(saturate(outColor.r) * 255.0);
    uint g = (uint)(saturate(outColor.g) * 255.0);
    uint b = (uint)(saturate(outColor.b) * 255.0);

    uint packedColor = (a << 24) | (g << 16) | (r << 8) | b;
    uint oldColor;
    InterlockedMax(BlockedBuffer[coord], packedColor, oldColor);
```

구현순서

1. XY, XZ, ZY 평면에 평행투영으로 그리기 테스트.
2. 버텍스 버퍼를 받아서 점 위치마다 6면체를 그리는 기능을 구현한다.
3. 3D Texture를 생성하고 구,박스등 공식을 이용해서 의의 값을 채워넣는다.
4. 2의 기능을 이용해서 3을 화면에 복셀로 렌더링 해본다.
5. 1,3을 이용해서 3D Texture에 3D모델을 그려 넣는다.

3D Texture 덩어리로 사용하기

- 캐릭터 한 마리마다 3D텍스처-Block buffer를 할당하지 말고 큰 덩어리의 3D텍스처를 할당.
- 덩어리 3D텍스처를 각 캐릭터의 크기별로 쪼개서 사용.
- OMSetRenderTarget, PSSetShaderResourceView, 비용을 줄일 수 있다.