Volume Ray Casting

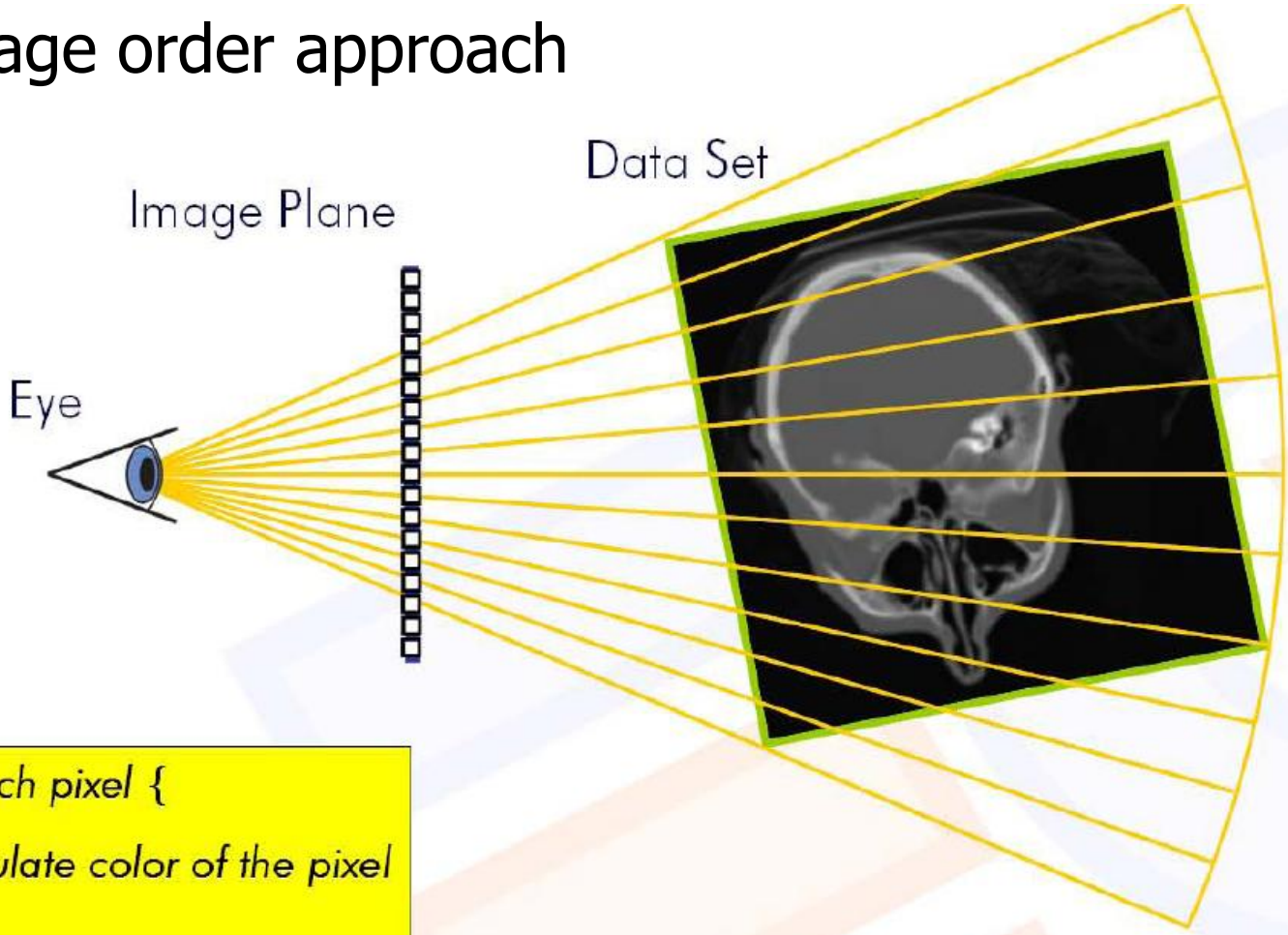
Practice

We Will…

- Start from the last code
- Implement volume ray-casting MIP

Volume Rendering

- Image order approach



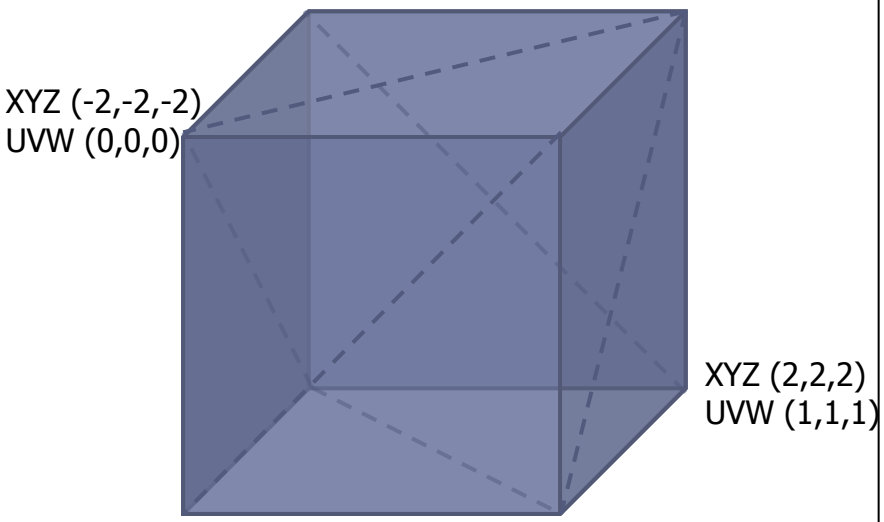

```
For each pixel {
    calculate color of the pixel
}
```

Volume Ray Casting

- Position setup for rays
  - Starting point
    - Rendering front faces of the bounding box
  - End point
    - Rendering back faces of the bounding box
  - Direction vector
    - End – Start (can be calculated on the fly)

Geometry Setup

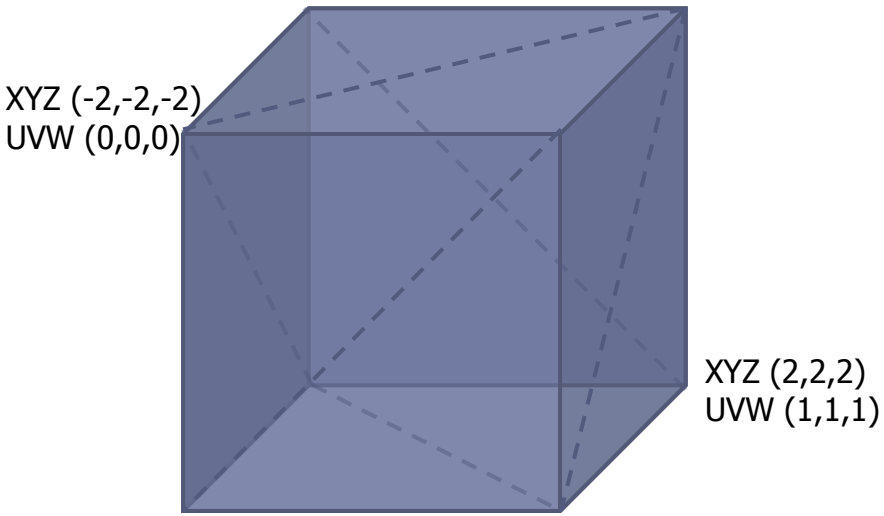
- Bounding Cube
  - Triangle List (check primitive type setup)
  - Beware the order of vertices (culling)
  - We use CCW as the front face




```
// Create vertex buffer
   SimpleVertex vertices[] =
   {
      XMFLOAT3(-2.0f,-2.0f,-2.0f ), XMFLOAT3( 0.0f, 0.0f, 0.0f ),
      XMFLOAT3(-2.0f,-2.0f, 2.0f ), XMFLOAT3( 0.0f, 0.0f, 1.0f ),
      XMFLOAT3(-2.0f, 2.0f,-2.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ),
      XMFLOAT3(-2.0f, 2.0f, 2.0f ), XMFLOAT3( 0.0f, 1.0f, 1.0f ),
      XMFLOAT3( 2.0f,-2.0f,-2.0f ), XMFLOAT3( 1.0f, 0.0f, 0.0f ),
      XMFLOAT3( 2.0f,-2.0f, 2.0f ), XMFLOAT3( 1.0f, 0.0f, 1.0f ),
      XMFLOAT3( 2.0f, 2.0f,-2.0f ), XMFLOAT3( 1.0f, 1.0f, 0.0f ),
      XMFLOAT3( 2.0f, 2.0f, 2.0f ), XMFLOAT3( 1.0f, 1.0f, 1.0f ),
   };

bd.ByteWidth = sizeof( SimpleVertex ) * 8;
```

Geometry Setup

- 12 triangles
  - 36 vertex definitions are needed on actual 8 vertices
- We use index buffer
  - 8 vertices + 36 indices




```
ID3D11Buffer*g_pIndexBuffer = NULL;
```

```
// Create index buffer
UINT indices[] =
{
                0, 1, 3,        0, 3, 2,        4, 5, 1,
                4, 1, 0,        5, 7, 3,        5, 3, 1,
                7, 6, 2,        7, 2, 3,        6, 4, 0,
                6, 0, 2,        6, 7, 5,        6, 5, 4,
};

bd.ByteWidth = sizeof( UINT ) * 36;
bd.BindFlags = D3D11_BIND_INDEX_BUFFER;
InitData.pSysMem = indices;
hr = g_pd3dDevice->CreateBuffer(&bd, &InitData, &g_pIndexBuffer );
   if( FAILED( hr ) )
      return hr;

 // Set index buffer
   g_pImmediateContext->IASetIndexBuffer
                ( g_pIndexBuffer, DXGI_FORMAT_R32_UINT, 0 );
```

Front Face Rendering

- Back face culling with CW option
  - Rename to g_pRasterizerStateFront
    - We will use other culling options later

```
// Create rasterizer state
D3D11_RASTERIZER_DESC rd;
ZeroMemory( &rd, sizeof(rd) );
rd.FillMode = D3D11_FILL_SOLID;
rd.FrontCounterClockwise = FALSE;
rd.CullMode = D3D11_CULL_BACK;
rd.DepthClipEnable = TRUE;
hr = g_pd3dDevice->CreateRasterizerState
                ( &rd, &g_pRasterizerStateFront );
if( FAILED( hr ) )
                return hr;

// Set rasterizer state
g_pImmediateContext->
                RSSetState( g_pRasterizerStateFront );
```

```
ID3D11RasterizerState*
    g_pRasterizerStateFront = NULL;
                                        :
                                        :


void CleanupDevice()
{
  if( g_pRasterizerStateFront )
                g_pRasterizerStateFront->Release();
}
```

Shader Code

- Walkthrough.fx
  - Rename VS to FaceVS, and PS to FacePS
    - We will use other VS/PS codes later

```
// Vertex Shader
VS_OUTPUT FaceVS( VS_INPUT In )
{
    VS_OUTPUT Out = (VS_OUTPUT)0;
    Out.Pos = float4( In.Pos.xyz, 1 );
    Out.Pos = mul( Out.Pos, World );
    Out.Pos = mul( Out.Pos, View );
    Out.Pos = mul( Out.Pos, Proj );
    Out.Tex = In.Tex;
    return Out;
}

// Pixel Shader
float4 FacePS( VS_OUTPUT In ) : SV_Target
{
   return float4(In.Tex, 1.0f);
}
```

```
ID3D11VertexShader*     g_pFaceVertexShader = NULL;
ID3D11PixelShader*      g_pFacePixelShader = NULL;
```

```
hr = CompileShaderFromFile( L"Walkthrough.fx", "FaceVS",
"vs_4_0", &pVSBlob );

hr = g_pd3dDevice->CreateVertexShader
  ( pVSBlob->GetBufferPointer(), pVSBlob->GetBufferSize(), NULL,
    &g_pFaceVertexShader );
                                        :
hr = CompileShaderFromFile( L"Walkthrough.fx", "FacePS",
"ps_4_0", &pPSBlob );

hr = g_pd3dDevice->CreatePixelShader
   ( pPSBlob->GetBufferPointer(), pPSBlob->GetBufferSize(), NULL,
    &g_pFacePixelShader );
```

Geometry Rendering

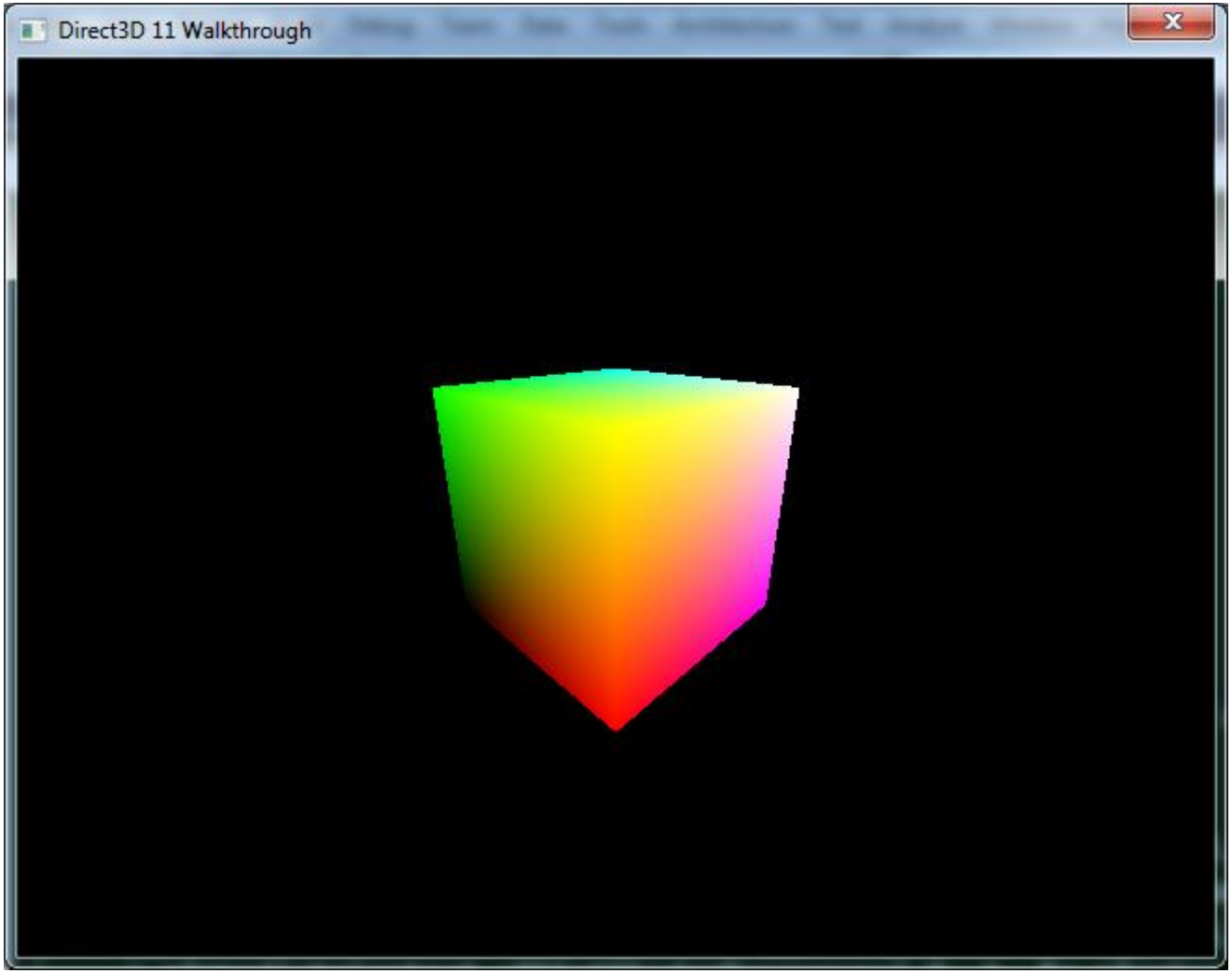
- Use DrawIndexed

```
// for(float z=0.0f; z<=1.0f; z+=0.01f)
// {
//  cb.texZ = z;
//  g_pImmediateContext->UpdateSubresource
//                          ( g_pConstantBuffer, 0, NULL, &cb, 0, 0 );
g_pImmediateContext->DrawIndexed( 36, 0, 0 );
//}
```

- Use other camera position to validate the result

```
// Initialize the view matrix
XMVECTOR Eye = XMVectorSet( 5.0f, 3.0f, -5.0f, 0.0f );
XMVECTOR At = XMVectorSet( 0.0f, 0.0f, 0.0f, 0.0f );
XMVECTOR Up = XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );
g_View = XMMatrixLookAtLH( Eye, At, Up );
```

Result



Render To Texture

- We use rendered front face image as an **input** of the next stage
  - Render front face to texture A, instead of the back buffer
  - Bind texture A as a shader resource of the next stage

Creating a RenderTarget Texture

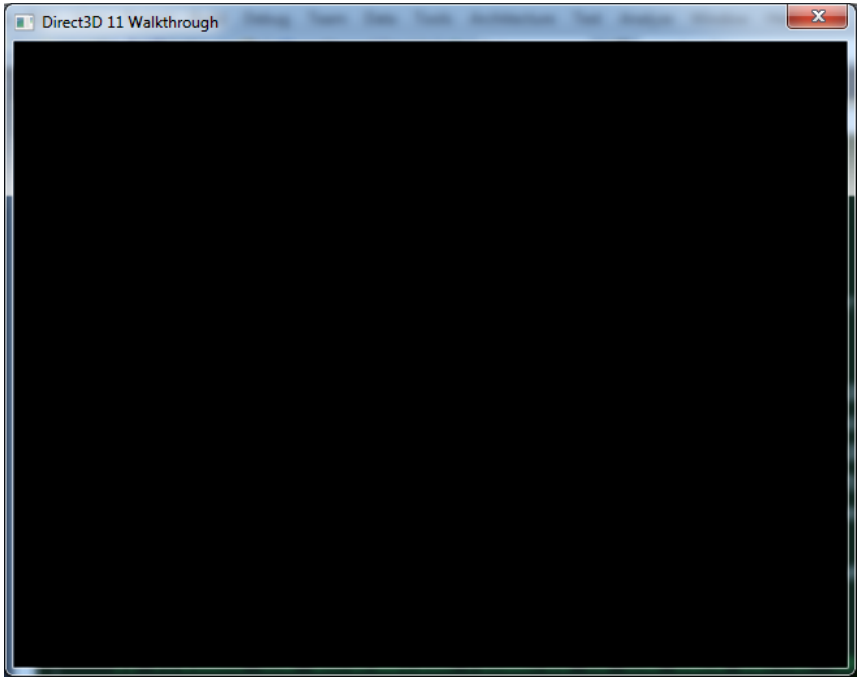
```
ID3D11Texture2D*                g_pFrontFaceTexture = NULL;
ID3D11RenderTargetView*         g_pFrontFaceRTView = NULL;
ID3D11ShaderResourceView*       g_pFrontFaceSRV = NULL;
```

```
// Create the front texture
D3D11_TEXTURE2D_DESC fdesc;
memset(&fdesc, 0, sizeof(fdesc));
fdesc.Width = 640; fdesc.Height = 480;          // size of the backbuffer
fdesc.Format = DXGI_FORMAT_R32G32B32A32_FLOAT;
fdesc.MipLevels = 1; fdesc.ArraySize = 1;
fdesc.SampleDesc.Count = 1; fdesc.SampleDesc.Quality = 0;
fdesc.Usage = D3D11_USAGE_DEFAULT;
fdesc.BindFlags = D3D11_BIND_RENDER_TARGET | D3D11_BIND_SHADER_RESOURCE;
fdesc.CPUAccessFlags = 0;
fdesc.MiscFlags = 0;

hr = g_pd3dDevice->CreateTexture2D(&fdesc, NULL, &g_pFrontFaceTexture);

hr = g_pd3dDevice->CreateRenderTargetView(g_pFrontFaceTexture , NULL, &g_pFrontFaceRTView );

D3D11_SHADER_RESOURCE_VIEW_DESC ff_srv_desc;
memset(&ff_srv_desc, 0, sizeof(ff_srv_desc));
ff_srv_desc.Format = fdesc.Format;
ff_srv_desc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
ff_srv_desc.Texture2D.MipLevels = fdesc.MipLevels;

hr = g_pd3dDevice->CreateShaderResourceView(g_pFrontFaceTexture, &ff_srv_desc, & g_pFrontFaceSRV );
```

Render to Target

- Set the render target as the created texture

```
g_pImmediateContext->OMSetRenderTargets( 1, &g_pFrontFaceRTView, NULL );
g_pImmediateContext->DrawIndexed( 36, 0, 0 );
```



- Empty result
  - We render the result to FrontFaceTexture
    - instead of the back buffer

Next Step: Backface Rendering

- End point of the sampling ray
- Identical to Front face rendering, except for culling option
  - We need the front-face culling rasterizer state

```
rd.DepthClipEnable = TRUE;
hr = g_pd3dDevice->CreateRasterizerState
                ( &rd, &g_pRasterizerStateFront );
if( FAILED( hr ) )
                return hr;

rd.CullMode = D3D11_CULL_FRONT;
hr = g_pd3dDevice->CreateRasterizerState
                ( &rd, &g_pRasterizerStateBack );
if( FAILED( hr ) )
                return hr;
```

```
ID3D11RasterizerState*
    g_pRasterizerStateBack = NULL;
                                        :
                                        :


void CleanupDevice()
{
  if( g_pRasterizerStateBack )
                g_pRasterizerStateBack->Release();
}
```

Back Face Texture

- Same as the front face texture..
  - Texture
  - Shader Resource View
  - Render Target View
- Do it yourself

Rendering Pass

```
float ClearColor[4] = { 0.0f, 0.0f, 0.0f, 1.0f }; // red,green,blue,alpha

g_pImmediateContext->OMSetRenderTargets( 1, &g_pFrontFaceRTView, NULL );
g_pImmediateContext->ClearRenderTargetView( g_pFrontFaceRTView, ClearColor );
g_pImmediateContext->PSSetShader( g_pFacePixelShader, NULL, 0 );
g_pImmediateContext->RSSetState(g_pRasterizerStateFront);
g_pImmediateContext->DrawIndexed( 36, 0, 0 );

g_pImmediateContext->OMSetRenderTargets( 1, &g_pBackFaceRTView, NULL );
g_pImmediateContext->ClearRenderTargetView(g_pBackFaceRTView, ClearColor );
g_pImmediateContext->PSSetShaderResources( 0, 1, &g_pBackFaceSRV );
g_pImmediateContext->RSSetState(g_pRasterizerStateBack);
g_pImmediateContext->DrawIndexed( 36, 0, 0 );
```

Let's MIP!

- What primitive do we render now?
  - Now we cast ray on the whole pixel of the image plane
- Proxy geometry
  - A quad which totally covers the image plane

```
// Create vertex buffer
    SimpleVertex vertices[] =
    {
        XMFLOAT3(-2.0f,-2.0f,-2.0f ), XMFLOAT3( 0.0f, 0.0f, 0.0f ),
        XMFLOAT3(-2.0f,-2.0f, 2.0f ), XMFLOAT3( 0.0f, 0.0f, 1.0f ),
        XMFLOAT3(-2.0f, 2.0f,-2.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ),
        XMFLOAT3(-2.0f, 2.0f, 2.0f ), XMFLOAT3( 0.0f, 1.0f, 1.0f ),
        XMFLOAT3( 2.0f,-2.0f,-2.0f ), XMFLOAT3( 1.0f, 0.0f, 0.0f ),
        XMFLOAT3( 2.0f,-2.0f, 2.0f ), XMFLOAT3( 1.0f, 0.0f, 1.0f ),
        XMFLOAT3( 2.0f, 2.0f,-2.0f ), XMFLOAT3( 1.0f, 1.0f, 0.0f ),
        XMFLOAT3( 2.0f, 2.0f, 2.0f ), XMFLOAT3( 1.0f, 1.0f, 1.0f ),

        XMFLOAT3(-1.0f, 1.0f, 0.5f), XMFLOAT3( 0.0f, 0.0f, 0.0f),
        XMFLOAT3( 1.0f, 1.0f, 0.5f), XMFLOAT3( 1.0f, 0.0f, 0.0f),
        XMFLOAT3(-1.0f,-1.0f, 0.5f), XMFLOAT3( 0.0f, 1.0f, 0.0f),
        XMFLOAT3( 1.0f,-1.0f, 0.5f), XMFLOAT3( 1.0f, 1.0f, 0.0f),
    };
    D3D11_BUFFER_DESC bd;
    ZeroMemory( &bd, sizeof(bd) );
    bd.Usage = D3D11_USAGE_DEFAULT;
    bd.ByteWidth = sizeof( SimpleVertex ) * 12;
```

Shader Code for Ray Casting

- Assign texture2D variables for front/back face SRV

```
Texture2D txFrontFace : register ( t1 );
Texture2D txBackFace : register ( t2 );
```

- Vertex Shader
  - Just bypass

```
VS_OUTPUT RayCastVS( VS_INPUT In )
{
        VS_OUTPUT Out = (VS_OUTPUT)0;
    Out.Pos = In.Pos;
        Out.Tex = In.Tex;
    return Out;
}
```

Shader Code for Ray Casting

- Pixel Shader

```
float4 RayCastPS( VS_OUTPUT In ) : SV_Target
{
        float stepSize = 0.005f;
        float2 screenPos;
        screenPos.x = In.Pos.x / 640;
        screenPos.y = In.Pos.y / 480;

        float3 start = txFrontFace.Sample(samLinear, screenPos);
        float3 end = txBackFace.Sample(samLinear, screenPos);

        float3 dir = end - start;
        float len = length(dir);

        clip(len);

        dir = normalize(dir);
                :
                :
```

Shader Code for Ray Casting

- Pixel Shader – loop
  - Use SampleLevel instead of Sample in the loop
    - Mip-map operations disable dynamic flow control!

```
                    :
        float m = 0.0f;

        for(float step = 0.0f; step < len; step += stepSize) {
                    float3 samplePos = start + dir * step;

                    float value = txVolume.SampleLevel(samLinear, samplePos, 0).r;

                    m = max(m, value);
        }

        return float4(m, m, m, 1);
}
```