Texture-Based Volume Rendering

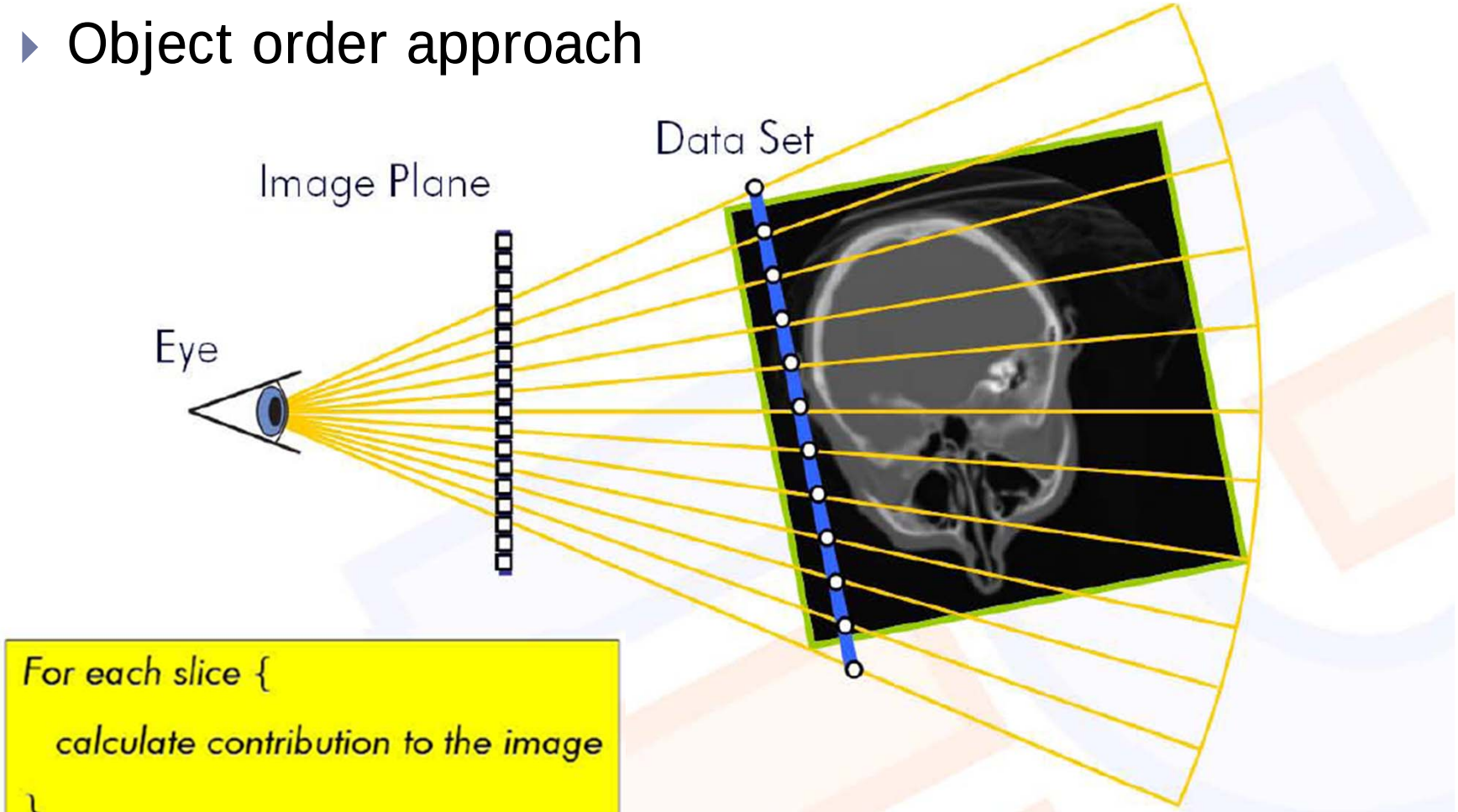
Practice

We Will...

- Start from previous walkthrough_3 source code
- Change 2D texture shader resource creation code
  - From simple “D3DX11CreateShaderResourceViewFromFile”
  - To manual texture creation and data filling
- Create a 3D texture and fill data
- Render a quad that maps the 3D texture
- Fold 3D texture mapped quads along the Z axis with MAX blending
  - i.e. Maximum Intensity Projection

Volume Rendering

- Object order approach




```
For each slice {
    calculate contribution to the image
}
```

Preparing a Texture : Previous Method

- D3DX11CreateShaderResourceViewFromFile
  - Automated utility function
    - Create a 2D texture with attribute(i.e. size of color depth) of the source file
    - Load the source image file and copy it to the texture
    - Create a shader resource view from the texture

Preparing a Texture : Manual Method

- We use RAW version of DDS texture for convenience
  - R8G8B8A8 with A=255

```
#include <fstream>
ID3D11ShaderResourceView*           g_pTextureSRV = NULL;
ID3D11Texture2D*                    g_pMyTexture = NULL;
                                    :
                                    :
// Load the Texture
unsigned char *image_data = new unsigned char[256*256*4];

std::ifstream ifs;
ifs.open(L"seafloor.raw", std::ios_base::binary);
ifs.read(reinterpret_cast<char*>(image_data), 256*256*4);
ifs.close();
```

Preparing a Texture : Manual Method

- Setting description structure for
  - Texture to create
  - Initial data buffer

```
D3D11_TEXTURE2D_DESC desc;
memset(&desc, 0, sizeof(D3D11_TEXTURE2D_DESC));
desc.Height = 256; desc.Width = 256;
desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
desc.ArraySize = 1; desc.MipLevels = 1;
desc.SampleDesc.Count = 1;
desc.Usage = D3D11_USAGE_IMMUTABLE;
desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;

D3D11_SUBRESOURCE_DATA initialData;
memset(&initialData, 0, sizeof(D3D11_SUBRESOURCE_DATA));
initialData.pSysMem = image_data;
initialData.SysMemPitch = 256 * 4;
```

Preparing a Texture : Manual Method

- Creating a texture object
  - CreateTexture2D method of D3D device object
  - Texture is filled with the buffer described in initlaiData

```
hr = g_pd3dDevice->CreateTexture2D(&desc, &initialData, &g_pMyTexture);

delete image_data;
```

Preparing a Texture : Manual Method

- Creating a shader resource view object
  - From the texture
  - With description

```
D3D11_SHADER_RESOURCE_VIEW_DESC srv_desc;
memset(&srv_desc, 0, sizeof(srv_desc));
srv_desc.Format = desc.Format;
srv_desc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D;
srv_desc.Texture2D.MipLevels = desc.MipLevels;

hr = g_pd3dDevice->CreateShaderResourceView(g_pMyTexture,
                                            &srv_desc, &g_pTextureSRV);
```

Extending to 3D texture

- We need...
  - 3D texture and SRV creation
  - 3D texture coordinate of primitives
- We will stop the annoying rotation, for convenience

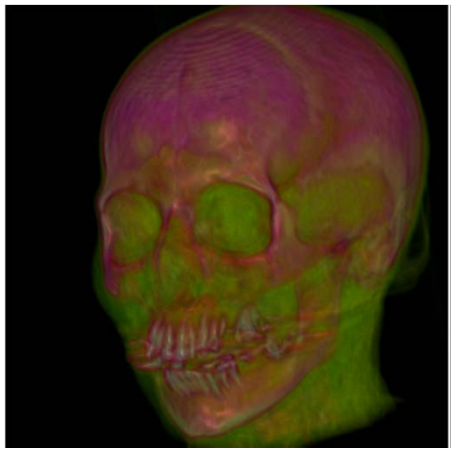
```
Void Render()
{
                    :
                    :
//
// Animate the cube
//
            g_World = XMMatrixIdentity();
//          g_World = XMMatrixRotationY( t );
```

Given Data

- Head CT data
- 256x256x225, 8bit per voxel (14,745,600 bytes)
- Raw data w/o header

```
ID3D11ShaderResourceView*           g_pTextureSRV = NULL;
ID3D11Texture3D*                    g_pMyTexture = NULL;
                                    :
                                    :
// Load the Texture
unsigned char *volume_data = new unsigned char[256*256*225];

std::ifstream ifs;
ifs.open(L“bighead.den", std::ios_base::binary);
ifs.read(reinterpret_cast<char*>(volume_data), 256*256*225);
ifs.close();
```



Preparing a 3D Texture

- Use just ...3D... structures instead of ...2D...

```
D3D11_TEXTURE3D_DESC desc;
memset(&desc, 0, sizeof(D3D11_TEXTURE3D_DESC));
desc.Height = 256; desc.Width = 256; desc.Depth = 225;
desc.Format = DXGI_FORMAT_R8_UNORM;
desc.MipLevels = 1;  // desc.ArraySize = 1;
// desc.SampleDesc.Count = 1;
desc.Usage = D3D11_USAGE_IMMUTABLE;
desc.BindFlags = D3D11_BIND_SHADER_RESOURCE;

D3D11_SUBRESOURCE_DATA initialData;
memset(&initialData, 0, sizeof(D3D11_SUBRESOURCE_DATA));
initialData.pSysMem = volume_data;
initialData.SysMemPitch = 256 * 1;
initialData.SysMemSlicePitch = 256 * 256 * 1;
```

Preparing a 3D Texture

- Creating a texture object
  - CreateTexture**3D** method of D3D device object
  - Texture is filled with the buffer described in initialData

```
hr = g_pd3dDevice->CreateTexture3D(&desc, &initialData, &g_pMyTexture);

delete volume_data;

D3D11_SHADER_RESOURCE_VIEW_DESC srv_desc;
memset(&srv_desc, 0, sizeof(srv_desc));
srv_desc.Format = desc.Format;
srv_desc.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE3D;
srv_desc.Texture3D.MipLevels = desc.MipLevels;

hr = g_pd3dDevice->CreateShaderResourceView(g_pMyTexture,
                                            &srv_desc, &g_pTextureSRV);
```

Using the 3D Texture

- Now TexCoord must be 3D coordinate

```
struct SimpleVertex
{
        XMFLOAT3 Pos;
        XMFLOAT3 Tex;
};
```

```
// Define the input layout
   D3D11_INPUT_ELEMENT_DESC layout[] =
   {
      { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0,
                                                    D3D11_INPUT_PER_VERTEX_DATA, 0 },
                     { "TEXCOORD", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12,
                                                    D3D11_INPUT_PER_VERTEX_DATA, 0 },
   };
```

Using the 3D Texture

- Primitive Definition
  - Let's draw a quad that maps a slice of "z=0.6f"

```
// Set primitive topology
   g_pImmediateContext->IASetPrimitiveTopology
                                        ( D3D11_PRIMITIVE_TOPOLOGY_TRIANGLESTRIP );
```

```
// Create vertex buffer
   SimpleVertex vertices[] =
   {
            XMFLOAT3( 2.0f, 2.0f, 0.5f ), XMFLOAT3( 1.0f, 0.0f, 0.6f ),
            XMFLOAT3( 2.0f,-2.0f, 0.5f ), XMFLOAT3( 1.0f, 1.0f, 0.6f ),
            XMFLOAT3(-2.0f, 2.0f, 0.5f ), XMFLOAT3( 0.0f, 0.0f, 0.6f ),
            XMFLOAT3(-2.0f,-2.0f, 0.5f ), XMFLOAT3( 0.0f, 1.0f, 0.6f ),
   };

   D3D11_BUFFER_DESC bd;
   ZeroMemory( &bd, sizeof(bd) );
   bd.Usage = D3D11_USAGE_DEFAULT;
   bd.ByteWidth = sizeof( SimpleVertex ) * 4;
```

Using the 3D Texture

- Change the shader code (.fx file) to use 3D texture

```
Texture3D txDiffuse : register( t0 );
```

```
struct VS_INPUT
{
    float4 Pos : POSITION;
    float3 Tex : TEXCOORD0;
};

struct VS_OUTPUT
{
    float4 Pos : SV_POSITION;
    float3 Tex : TEXCOORD0;
};
```

Using the 3D Texture

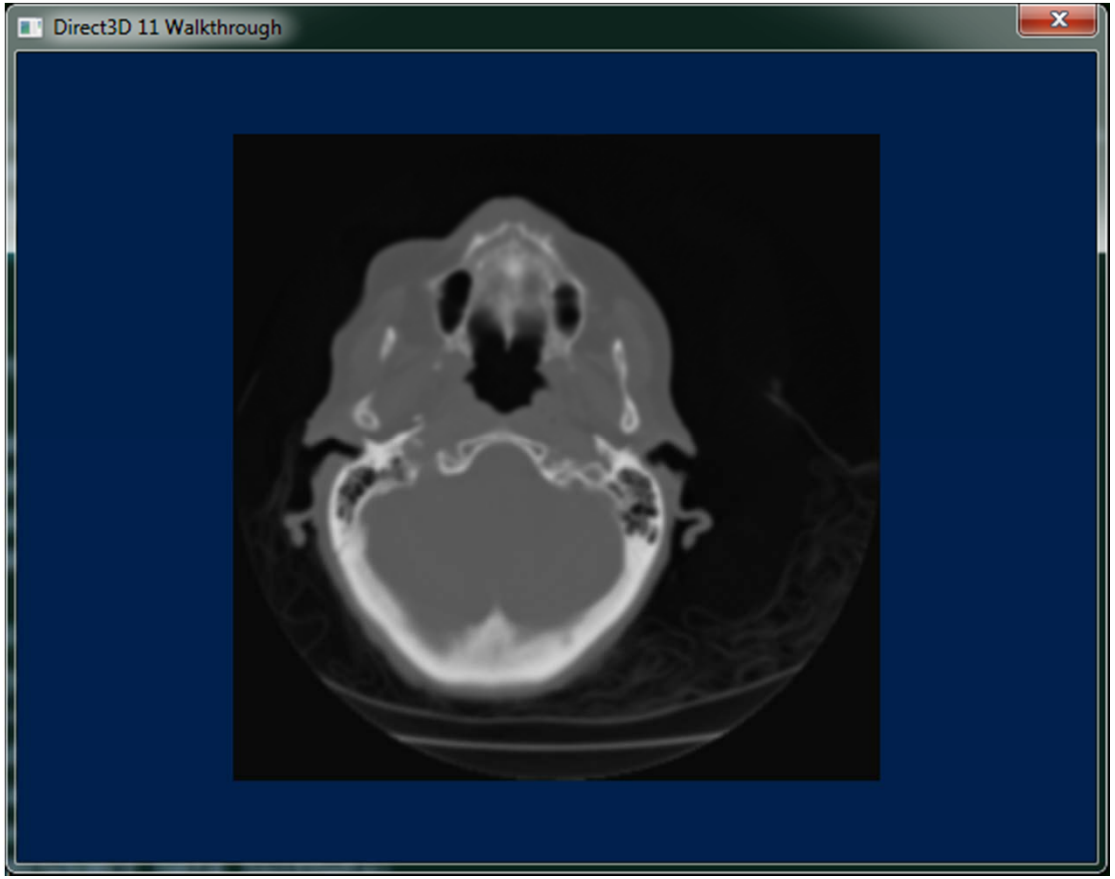
- The 3D texture we created has just 1 channel
  - Intensity value
  - For now, we just render the intensity value as gray level

```
float4 PS( VS_OUTPUT In ) : SV_Target
{
        float intensity = txDiffuse.Sample( samLinear, In.Tex );
        return float4(intensity, intensity, intensity, intensity);
}
```

```
Void Render()
{
                :
                :
        g_pImmediateContext->Draw( 4, 0 );
```

Result

- A slice quad is rendered



Let's Stack!

- We adopt a constant which stores the z-texcoord
  - In the constant buffer

```
struct ConstantBuffer
{
        XMMATRIX        mWorld;
        XMMATRIX        mView;
        XMMATRIX        mProj;
        float           texZ;
};
```

```
cbuffer ConstantBuffer : register( b0 )
{
        matrix World;
        matrix View;
        matrix Proj;
        float  texZ;
}
```

Let's Stack!

- We draw *series* of quads which varies only z-value of the texture coordinate!
  - Actual position of the quad in 3D world space has no meaning → ***proxy geometry***

```
void Render()
{
                                        :
                                        :
 for(float z=0.0f; z<=1.0f; z+=0.01f)       // 101 slices
 {
    cb.texZ = z;
    g_pImmediateContext->UpdateSubresource(g_pConstantBuffer, 0, NULL, &cb, 0, 0);
    g_pImmediateContext->Draw(4, 0);
 }
```

Let's Stack!

- Set texZ to the z-value of texture coordinate

```
//--------------------------------------------------------------------------------
// Vertex Shader
//--------------------------------------------------------------------------------
VS_OUTPUT VS( 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;
        Out.Tex.z = texZ;
    return Out;
}
```

Let's Stack!

- We implement Maximum Intensity Projection
  - Maximum value along the ray survived
- By setting blending option

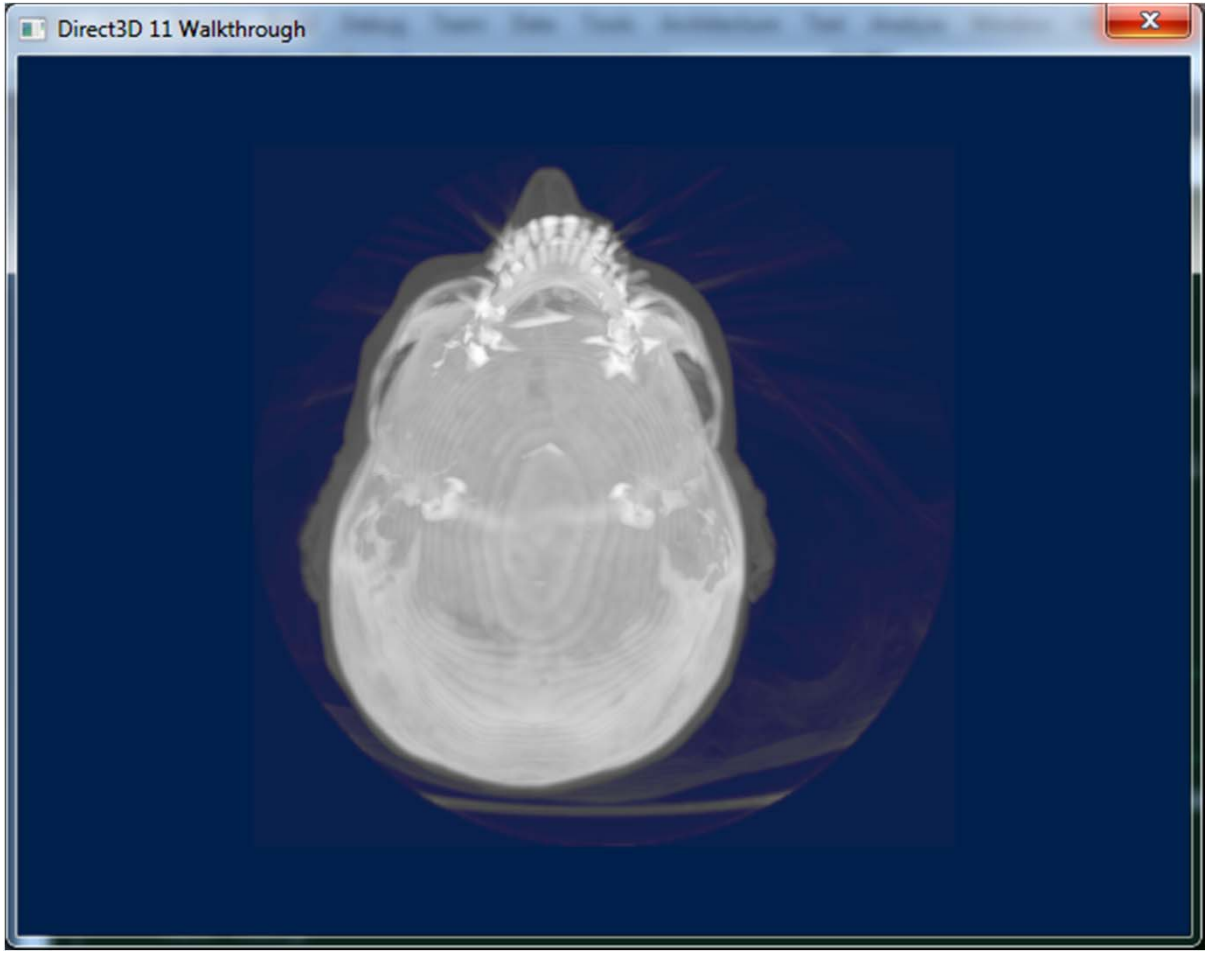
```
ID3D11BlendState*                                   g_pBlendState = NULL;
```

```
// Create blend state
D3D11_BLEND_DESC bld;
ZeroMemory( &bld, sizeof(bld) );
bld.RenderTarget[0].RenderTargetWriteMask = D3D11_COLOR_WRITE_ENABLE_ALL;
bld.RenderTarget[0].BlendEnable = TRUE;
bld.RenderTarget[0].SrcBlend = D3D11_BLEND_ONE;
bld.RenderTarget[0].DestBlend = D3D11_BLEND_ONE;
bld.RenderTarget[0].BlendOp = D3D11_BLEND_OP_MAX;
bld.RenderTarget[0].SrcBlendAlpha = D3D11_BLEND_ONE;
bld.RenderTarget[0].DestBlendAlpha = D3D11_BLEND_ONE;
bld.RenderTarget[0].BlendOpAlpha = D3D11_BLEND_OP_MAX;
hr = g_pd3dDevice->CreateBlendState( &bld, &g_pBlendState);

// Set blend state
g_pImmediateContext->OMSetBlendState( g_pBlendState, 0 ,0xFFFFFFFF);
```

Result

- Maximum Intensity Projection



Exercises / Discussions

- MinIP Implementation
  - Minimum Intensity Projection
  - With MIP code, you can simply test MinIP also!
  - You will get very weird result image for MinIP. Why?
- Rotating the volume
  - Is Rotating proxy geometry proper method for volume rotation?
  - How can we implement volume rotation?