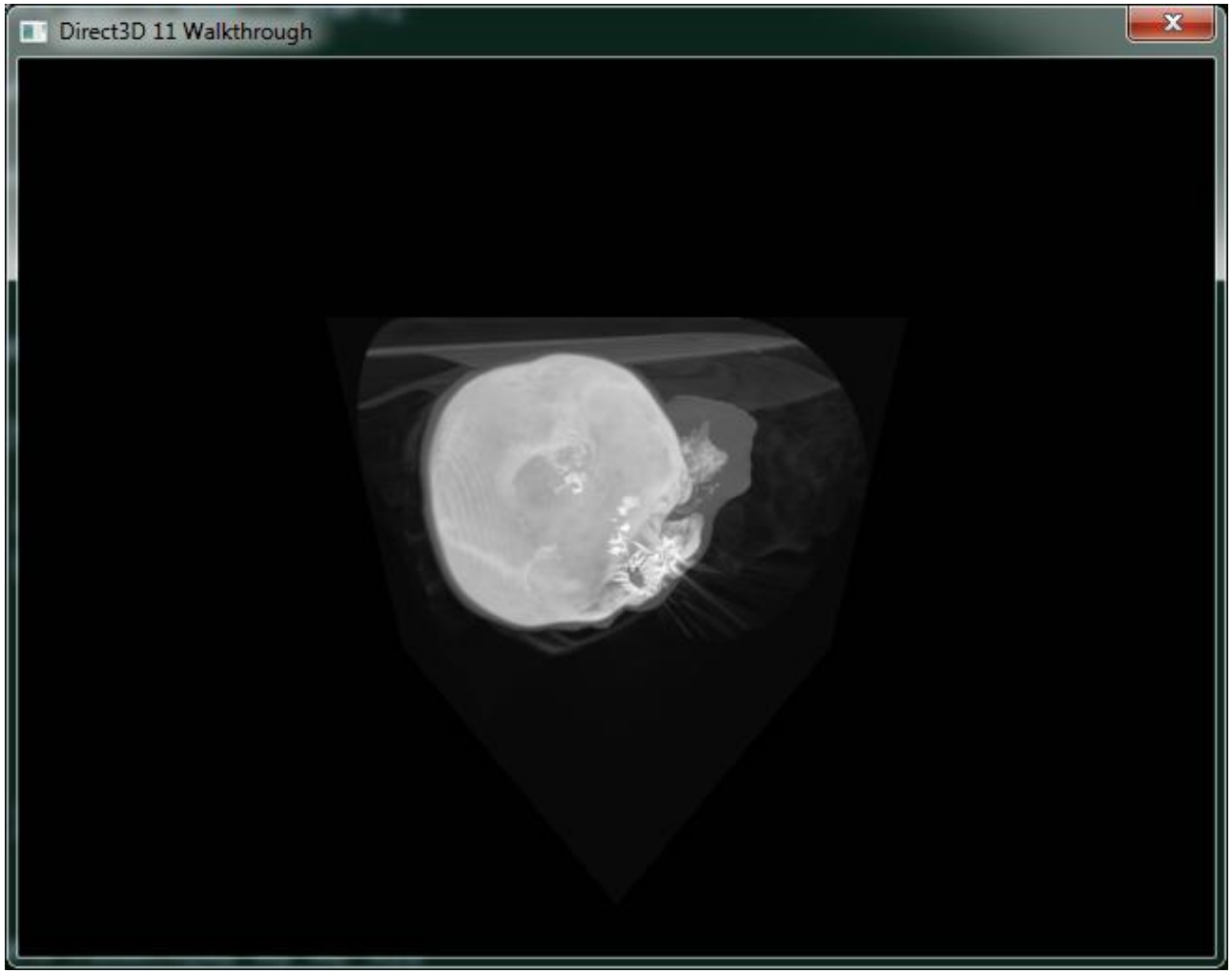
Transfer Function

Practice

We Will...

- Start from the ray-casting MIP code
- Create a transfer function texture
- On-the-fly gradient calculation
- Implement direct volume ray-casting
- Early ray termination

MIP Ray Casting



Transfer Function Definition

- We use 1D function
  - (intensity) → (color, opacity)
- Can be defined as 1D texture
  - Width = domain(intensity) resolution
  - RGBA channel
    - RGB for emission term
    - A for absorption term
- 16bit/voxel volume data
  - Width = 64k → too big!
  - 1D 64k → 2D (1k x 64)

Transfer Function Definition

- TF is stored in a binary file
  - TF.bin
  - 64k x float4 raw file

```
// Create the TF texture
float *tf_data = new float[65536 * 4];

ifs.open(L"TF.bin", std::ios_base::binary);
ifs.read(reinterpret_cast<char*>(tf_data), 65536 * 4 * sizeof(float));
ifs.close();

initialData.pSysMem = tf_data;
initialData.SysMemPitch = 1024 * 4 * sizeof(float);
initialData.SysMemSlicePitch = 0;
```

Transfer Function Definition

- TF texture and SRV creation

```
ID3D11Texture2D*            g_pTFTexture = NULL;
ID3D11ShaderResourceView*   g_pTFSRV = NULL;
```

```
// Create the TF texture
fdesc.Width = 1024; fdesc.Height = 64;
fdesc.BindFlags = D3D11_BIND_SHADER_RESOURCE;

hr = g_pd3dDevice->CreateTexture2D(&fdesc, &initialData, &g_pTFTexture);
   if( FAILED( hr ) )
      return hr;

hr = g_pd3dDevice->CreateShaderResourceView(g_pTFTexture, &ff_srv_desc, &g_pTFSRV );
                  if( FAILED( hr ) )
   return hr;
```

Transfer Function Definition

- Binding TF texture to the shader

```
Texture2D txTF : register ( t3 );
```

```
// Render
g_pImmediateContext->OMSetRenderTargets( 1, &g_pRenderTargetView, NULL );
g_pImmediateContext->ClearRenderTargetView( g_pRenderTargetView, ClearColor );
g_pImmediateContext->PSSetShaderResources( 2, 1, &g_pBackFaceSRV );
g_pImmediateContext->PSSetShaderResources( 3, 1, &g_pTFSRV );
g_pImmediateContext->VSSetShader( g_pRayCastVertexShader, NULL, 0 );
g_pImmediateContext->PSSetShader( g_pRayCastMIPPixelShader, NULL, 0 );
g_pImmediateContext->RSSetState( g_pRasterizerStateNoCull );
g_pImmediateContext->DrawIndexed( 6, 36, 0 );
```

Gradient Calculation

- Central difference
- Texture.Sample method with **integer** offset

```
inline float3 GetGradient(float3 p)
{
	float3 Gradient;

	Gradient.x = txVolume.SampleLevel(samLinear, p, 0, int3( 1, 0, 0))
				- txVolume.SampleLevel(samLinear, p, 0, int3(-1, 0, 0));
	Gradient.y = txVolume.SampleLevel(samLinear, p, 0, int3( 0, 1, 0))
				- txVolume.SampleLevel(samLinear, p, 0, int3( 0,-1, 0));
	Gradient.z = txVolume.SampleLevel(samLinear, p, 0, int3( 0, 0, 1))
				- txVolume.SampleLevel(samLinear, p, 0, int3( 0, 0,-1));

	return normalize(Gradient);
}
```

Classification

- Sample from TF texture

```
float4 RayCastPS( VS_OUTPUT In ) : SV_Target
{
                        :
                        :
    dir = normalize(dir);
    float3 light = normalize(float3(-1, -1, 10));
    float4 output = 0;
    for(float step = 0.0f; step < len; step += stepSize) {
        if(step < len) {
            float3 samplePos = start + dir * step;
            int intensity = txVolume.SampleLevel(samLinear, samplePos, 0).r * 65535;
            float2 intensityPos;
            intensityPos.y = ((intensity / 1024) + 0.5f) / 64;
            intensityPos.x = ((intensity % 1024) + 0.5f) / 1024;

            float4      value = txTF.SampleLevel(samLinear, intensityPos, 0);
                        :
                        :
```

Shading

- Phong illumination model
- Eye vector = direction vector
- When alpha = 0, shading operations are meaningless

```
                    :
                    :
float4      value = txTF.SampleLevel(samLinear, intensityPos, 0);

if(value.a > 0)
{
      float3 gradient = GetGradient(samplePos);
      float diffuse = saturate(dot(light, gradient));
      float specular = saturate(pow(dot(2 * dot(light, gradient) * gradient - light, dir), 5));
                    :
                    :
```

Composition

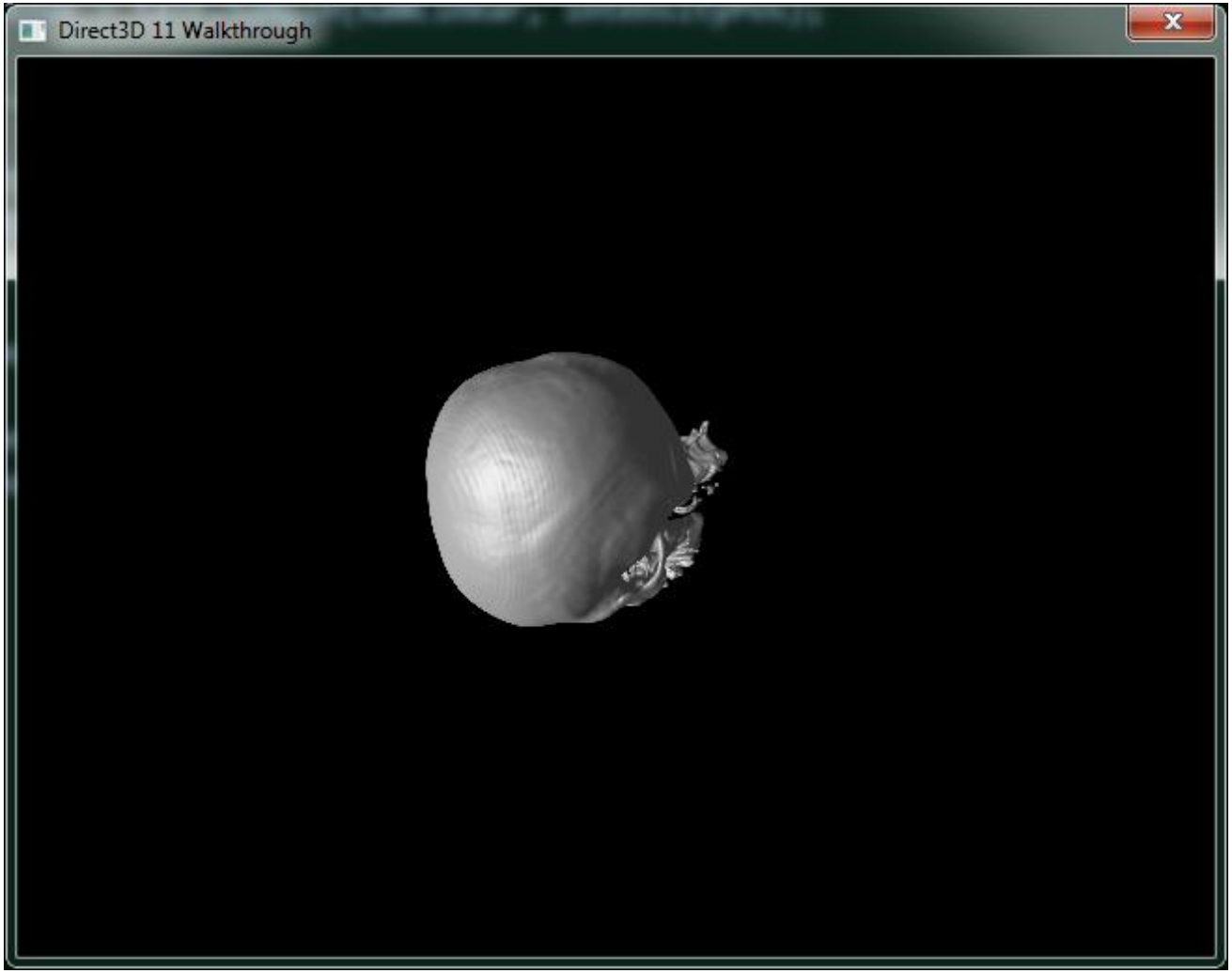
- Front-to-back composition

```
                    :
                    :
            float specular = saturate(pow(dot(2 * dot(light, gradient) * gradient - light, dir), 5));

            value.rgb *= value.a;
            value.rgb *= 0.2f + 0.5f * diffuse + 0.3f * specular;
            output.rgba += (1 - output.a) * value.rgba;
        }
    }

    return output;
}
```

Result



Early Ray Termination

- If alpha is saturated enough, further sampling / classification / composition process can be omitted

```
                    :
                    :
                float specular = saturate(pow(dot(2 * dot(light, gradient) * gradient - light, dir), 5));

                value.rgb *= value.a;
                value.rgb *= 0.2f + 0.5f * diffuse + 0.3f * specular;
                output.rgba += (1 - output.a) * value.rgba;
            }
            if(output.a > 0.98f)
                break;
        }

        return output;
}
```

Assignment #2