准备工作

新建项目,将DX教程代码中对应的项目9的代码的HSL文件夹,Texture文件夹,C++头文件和源文件复制到新项目中,并做好相应的属性配置

正方体六种不同纹理的实现和光源的切换

```
// 初始化魔方纹理
m_pcubes.resize(6);
for (int i = 1; i <= 6; ++i)
{
    wsprintf(strFile, L"Texture\\Resource\\cube%03d.dds", i);
    HR(CreateDDSTextureFromFile(m_pd3dDevice.Get(), strFile, nullptr, m_pcubes[static_cast<size_t>(i) - 1].GetAddressOf()));
}
```

```
else if(m_CurrMode == ShowMode::WoodCrate)
    for (int j = 0; j < 6; j++) {
    m_pd3dImmediateContext->PSSetShaderResources(0, 1, m_pcubes[j].GetAddressOf());
    m_pd3dImmediateContext->PSSetShaderResources(1, 1, m_pcubes[j].GetAddressOf());
    m_pd3dImmediateContext->DrawIndexed(j*6+6, j*6, 0);
}
```

```
// 键盘切换模式
if (m_KeyboardTracker.IsKeyPressed(Keyboard::U)) {
    m_PSConstantBuffer.dirLight[0].ambient = XMFLOAT4(0.3f, 0.3f, 0.3f, 1.0f);
    m_PSConstantBuffer.dirLight[0].diffuse = XMFLOAT4(0.7f, 0.7f, 0.7f, 1.0f);
    m_PSConstantBuffer.dirLight[0].specular = XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f);
    m_PSConstantBuffer.dirLight[0].direction = XMFLOAT3(-0.577f, -0.577f, 0.577f);

    m_PSConstantBuffer.numDirLight = 1;
    m_PSConstantBuffer.numPointLight = 0;
    m_PSConstantBuffer.numSpotLight = 0;
    D3D11_MAPPED_SUBRESOURCE mappedData;
    HR(m_pd3dImmediateContext->Map(m_pConstantBuffers[1].Get(), 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedData));
    memcpy_s(mappedData.pData, sizeof(PSConstantBuffer), &m_PSConstantBuffer, sizeof(PSConstantBuffer));
    m_pd3dImmediateContext->Unmap(m_pConstantBuffers[1].Get(), 0);
}
```

纹理数组

在初始化函数中加入:

```
//初始化火焰纹理数组
std::vector<std::wstring> fileNames;
fileNames.resize(120);
for (int i = 1; i <= 120; ++i)
{
    if (i < 10) {
        fileNames[i - 1] = L"Texture\\FireAnim\\Fire00" + std::to_wstring(i) + L".bmp";
    }
    else if (i >= 10 && i < 100) {
        fileNames[i - 1] = L"Texture\\FireAnim\\Fire0" + std::to_wstring(i) + L".bmp";
    }
    else {
        fileNames[i - 1] = L"Texture\\FireAnim\\Fire" + std::to_wstring(i) + L".bmp";
    }
}
HR(CreateTexture2DArrayFromFile(m_pd3dDevice.Get(), m_pd3dImmediateContext.Get(), fileNames, nullptr, m_pFire.GetAddressOf()));
```

在HLSL文件夹的Basic头文件中加入:

```
Texture2DArray gTexArray : register(t2);
```

在2D像素着色器中修改:

```
// 像素着色器(2D)
float4 PS_2D(VertexPosHTex pIn) : SV_Target
{

    float4 texColor = gTexArray.Sample(g_SamLinear, float3(pIn.Tex, g_Pad1));
    return texColor;
}
```

随后在HLSL的PS常量缓冲区中以g_pad1作为参数(既C++中PS常量缓冲区的pad)来控制纹理数组的索引值,

```
if (totDeltaTime > 1.0f / 60)
{
    totDeltaTime -= 1.0f / 60;
    m_CurrFrame = (m_CurrFrame + 1) % 120;
    m_pd3dImmediateContext->PSSetShaderResources(2, 1, m_pFire.GetAddressOf());
    // 利用多余的变量pad进行传参
    m_PSConstantBuffer.pad = m_CurrFrame;
    D3D11_MAPPED_SUBRESOURCE mappedData;
    HR(m_pd3dImmediateContext->Map(m_pConstantBuffers[1].Get(), 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedData));
    memcpy_s(mappedData.pData, sizeof(m_PSConstantBuffer), &m_PSConstantBuffer, sizeof(m_PSConstantBuffer));
    m_pd3dImmediateContext->Unmap(m_pConstantBuffers[1].Get(), 0);
}
```

但不知道为何无效.2D动画一直卡在第一张纹理,因此本题被迫采用最初始的方法

纹理旋转与相乘

纹理旋转

在HLSL和C++的VS常量缓冲区中新增一个变量用于储存旋转矩阵

```
struct VSConstantBuffer
{
    DirectX::XMMATRIX world;
    DirectX::XMMATRIX view;
    DirectX::XMMATRIX proj;
    DirectX::XMMATRIX worldInvTranspose;
    //新增变量Tex
    DirectX::XMMATRIX Tex;
};
```

```
//纹理旋转
m_VSConstantBuffer.Tex = XMMatrixTranspose(XMMatrixRotationZ(2*z));
// 更新常量缓冲区，让纹理转起来
D3D11_MAPPED_SUBRESOURCE mappedData;
HR(m_pd3dImmediateContext->Map(m_pConstantBuffers[0].Get(), 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedData));
memcpy_s(mappedData.pData, sizeof(VSConstantBuffer), &m_VSConstantBuffer, sizeof(VSConstantBuffer));
m_pd3dImmediateContext->Unmap(m_pConstantBuffers[0].Get(), 0);
```

在3D顶点着色器中使纹理与旋转矩阵相乘(注意要先修改纹理的XY值使其绕中心旋转,然后再修改回来)

```
    vOut.Tex = mul(float4(vIn.Tex.x - 0.5f, vIn.Tex.y - 0.5f, 1.0f, 1.0f), Tex).xy;
    vOut.Tex += 0.5f;
```

纹理相乘

新增一个采样器,用两个采样器同时储存两个纹理

```
// 像素着色阶段设置好采样器
m_pd3dImmediateContext->PSSetSamplers(0, 1, m_pSamplerState.GetAddressOf());
m_pd3dImmediateContext->PSSetShaderResources(0, 1, m_pcubes[0].GetAddressOf());
m_pd3dImmediateContext->PSSetShader(m_pPixelShader3D.Get(), nullptr, 0);

m_pd3dImmediateContext->PSSetSamplers(1, 1, m_pSamplerState.GetAddressOf());
```

```
    m_pd3dImmediateContext->PSSetShaderResources(0, 1, m_pFlare.GetAddressOf());
    m_pd3dImmediateContext->PSSetShaderResources(1, 1, m_pFlarealpha.GetAddressOf());
```

随后在3D像素着色器中令两个纹理直接相乘

```
float4 texColor = g_Tex.Sample(g_SamLinear, pIn.Tex);
float4 texColoralpha = g_Tex1.Sample(g_SamLinear1, pIn.Tex);
texColor = texColor * texColoralpha;
```

已知问题

纹理数组的索引值不会随之变动