게임프로그래밍(N)

기말 최종 프로젝트

1871377 오병택

목차

1. 다양한 텍스쳐맵 활용

application.cpp에 ModelClass와 WindowModel 2개의 객체 생성

```
// 첫번째 모델, 텍스쳐, normal 설정
strcpy_s(modelFilename, "data/cube.txt");
strcpy_s(textureFilename1, "data/stone01.tga");
strcpy_s(textureFilename2, "data/normal03.tga");

// Create and initialize the cube model object.
m_Model = new ModelClass;

result = m_Model->Initialize(m_Direct3D->GetDevice(), m_Direct3D->GetDeviceContext(), modelFilename, textureFilename1, textureFilename2);
if(!result)
{
    MessageBox(hwnd, L"Could not initialize the cube model object.", L"Error", MB_OK);
    return false;
}

// 두번째 모델, 텍스쳐, normal 설정
strcpy_s(modelFilename, "data/square.txt");
strcpy_s(textureFilename1, "data/glass01.tga");
strcpy_s(textureFilename2, "data/normal03.tga");

// Create and initialize the window model object.
m_WindowModel  = new ModelClass;

result = m_WindowModel->Initialize(m_Direct3D->GetDevice(), m_Direct3D->GetDeviceContext(), modelFilename, textureFilename1, textureFilename2);
if(!result)
{
    MessageBox(hwnd, L"Could not initialize the window model object.", L"Error", MB_OK);
    return false;
}

// 세번째 모델, 텍스쳐, normal 설정
strcpy_s(textureFilename1, "data/ice01.tga");
strcpy_s(textureFilename2, "data/icebump01.tga");

m_WindowModel2 = new ModelClass;

result = m_WindowModel2->Initialize(m_Direct3D->GetDevice(), m_Direct3D->GetDeviceContext(), modelFilename, textureFilename1, textureFilename2);
if (!result)
{
    MessageBox(hwnd, L"Could not initialize the window model object.", L"Error", MB_OK);
    return false;
}
```

ModelClass의 TextureShader와 WindowModel의 GlassShader 생성

```
// TextureShader 생성
m_TextureShader  = new TextureShaderClass;

result = m_TextureShader->Initialize(m_Direct3D->GetDevice(), hwnd);
if(!result)
{
    MessageBox(hwnd, L"Could not initialize the texture shader object.", L"Error", MB_OK);
    return false;
}

// GlassShader 생성
m_GlassShader = new GlassShaderClass;

result = m_GlassShader->Initialize(m_Direct3D->GetDevice(), hwnd);
if(!result)
{
    MessageBox(hwnd, L"Could not initialize the glass shader object.", L"Error", MB_OK);
    return false;
}
```

1) 사각형 모델에 바위 Texture 적용

Layout 설정

```
polygonLayout[0].SemanticName = "POSITION";
polygonLayout[0].SemanticIndex = 0;
polygonLayout[0].Format = DXGI_FORMAT_R32G32B32_FLOAT;
polygonLayout[0].InputSlot = 0;
polygonLayout[0].AlignedByteOffset = 0;
polygonLayout[0].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
polygonLayout[0].InstanceDataStepRate = 0;

polygonLayout[1].SemanticName = "TEXCOORD";
polygonLayout[1].SemanticIndex = 0;
polygonLayout[1].Format = DXGI_FORMAT_R32G32_FLOAT;
polygonLayout[1].InputSlot = 0;
polygonLayout[1].AlignedByteOffset = D3D11_APPEND_ALIGNED_ELEMENT;
polygonLayout[1].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
polygonLayout[1].InstanceDataStepRate = 0;

polygonLayout[2].SemanticName = "NORMAL";
polygonLayout[2].SemanticIndex = 0;
polygonLayout[2].Format = DXGI_FORMAT_R32G32_FLOAT;
polygonLayout[2].InputSlot = 0;
polygonLayout[2].AlignedByteOffset = D3D11_APPEND_ALIGNED_ELEMENT;
polygonLayout[2].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
polygonLayout[2].InstanceDataStepRate = 0;
```

Description 설정 (Sampler, Matrix, Light)

```
samplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
samplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_WRAP;
samplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_WRAP;
samplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
samplerDesc.MipLODBias = 0.0f;
samplerDesc.MaxAnisotropy = 1;
samplerDesc.ComparisonFunc = D3D11_COMPARISON_ALWAYS;
samplerDesc.BorderColor[0] = 0;
samplerDesc.BorderColor[1] = 0;
samplerDesc.BorderColor[2] = 0;
samplerDesc.BorderColor[3] = 0;
samplerDesc.MinLOD = 0;
samplerDesc.MaxLOD = D3D11_FLOAT32_MAX;


result = device->CreateSamplerState(&samplerDesc, &m_sampleState);
if (FAILED(result))
{
    return false;
}

matrixBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
matrixBufferDesc.ByteWidth = sizeof(MatrixBufferType);
matrixBufferDesc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
matrixBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
matrixBufferDesc.MiscFlags = 0;
matrixBufferDesc.StructureByteStride = 0;

result = device->CreateBuffer(&matrixBufferDesc, NULL, &m_matrixBuffer);
if (FAILED(result))
{
    return false;
}

lightBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
lightBufferDesc.ByteWidth = sizeof(LightBufferType);
lightBufferDesc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
lightBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
lightBufferDesc.MiscFlags = 0;
lightBufferDesc.StructureByteStride = 0;

result = device->CreateBuffer(&lightBufferDesc, NULL, &m_lightBuffer);
if (FAILED(result))
{
    return false;
}

return true;
```

텍스처에 대한 Parameter 설정 (Map -> Unmap)

1. world, view, projection Matrix값 VS로 넘겨주기

```
bool TextureShaderClass::SetShaderParameters(ID3D11DeviceContext* deviceContext, XMMATRIX worldMatrix, XMMATRIX viewMatrix,
                                             XMMATRIX projectionMatrix, ID3D11ShaderResourceView* texture,XMFLOAT3 lightDirection, XMFLOAT4 diffuseColor, XMFLOAT3 lightDirection2, XMFLOAT4 diffuseColor2)
{
    HRESULT result;
    D3D11_MAPPED_SUBRESOURCE mappedResource;
    MatrixBufferType* dataPtr;
    LightBufferType* dataPtr2;
    unsigned int bufferNumber;


    // Transpose the matrices to prepare them for the shader.
    worldMatrix = XMMatrixTranspose(worldMatrix);
    viewMatrix = XMMatrixTranspose(viewMatrix);
    projectionMatrix = XMMatrixTranspose(projectionMatrix);

    // Lock the constant buffer so it can be written to.
    result = deviceContext->Map(m_matrixBuffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedResource);
    if(FAILED(result))
    {
        return false;
    }

    // Get a pointer to the data in the constant buffer.
    dataPtr = (MatrixBufferType*)mappedResource.pData;

    // Copy the matrices into the constant buffer.
    dataPtr->world = worldMatrix;
    dataPtr->view = viewMatrix;
    dataPtr->projection = projectionMatrix;

    deviceContext->Unmap(m_matrixBuffer, 0);

    bufferNumber = 0;

    deviceContext->VSSetConstantBuffers(bufferNumber, 1, &m_matrixBuffer);
    deviceContext->PSSetShaderResources(0, 1, &texture);
```

2. 텍스처에 대한 조명연산 ps로 넘겨주기

```
    dataPtr2 = (LightBufferType*)mappedResource.pData;
    dataPtr2->diffuseColor = diffuseColor;
    dataPtr2->lightDirection = lightDirection;
    dataPtr2->padding = 0.0f;

    dataPtr2->diffuseColor2 = diffuseColor2;
    dataPtr2->lightDirection2 = lightDirection2;
    dataPtr2->padding2 = 0.0f;

    deviceContext->Unmap(m_lightBuffer, 0);

    bufferNumber = 0;

    deviceContext->PSSetConstantBuffers(bufferNumber, 1, &m_lightBuffer);

    return true;
}


void TextureShaderClass::RenderShader(ID3D11DeviceContext* deviceContext, int indexCount)
{
    deviceContext->IASetInputLayout(m_layout);
    deviceContext->VSSetShader(m_vertexShader, NULL, 0);
    deviceContext->PSSetShader(m_pixelShader, NULL, 0);
    deviceContext->PSSetSamplers(0, 1, &m_sampleState);
    deviceContext->DrawIndexed(indexCount, 0, 0);

    return;
}
```

TextureClass의 Render()

```cpp
bool TextureShaderClass::Render(ID3D11DeviceContext* deviceContext, int indexCount, XMMATRIX worldMatrix, XMMATRIX viewMatrix,
                                XMMATRIX projectionMatrix, ID3D11ShaderResourceView* texture, XMFLOAT3 lightDirection, XMFLOAT4 diffuseColor, XMFLOAT3 lightDirection2, XMFLOAT4 diffuseColor2)
{
    bool result;


    // Set the shader parameters that it will use for rendering.
    result = SetShaderParameters(deviceContext, worldMatrix, viewMatrix, projectionMatrix, texture, lightDirection, diffuseColor, lightDirection2, diffuseColor2);
    if(!result)
    {
        return false;
    }

    // Now render the prepared buffers with the shader.
    RenderShader(deviceContext, indexCount);

    return true;
}
```

texture에 대한 정보 PixelShader로 넘겨주기

```cpp
////////////////////////////////////////////////////////////////////////////////
// Pixel Shader
////////////////////////////////////////////////////////////////////////////////
float4 TexturePixelShader(PixelInputType input) : SV_TARGET
{
    float4 textureColor;

    float4 color;

    float3 lightDir;
    float3 lightDir2;

    float lightIntensity;
    float lightIntensity2;


    textureColor = shaderTexture1.Sample(SampleType, input.tex);

    // Invert the light direction for calculations.
    lightDir = -lightDirection;
    lightIntensity = saturate(dot(input.normal, lightDir));

    lightDir2 = -lightDirection2;
    lightDir2 = normalize(lightDir2);


    float3 HalfV;
    float3 viewingV;
    viewingV = normalize(-input.normal.xyz);

    HalfV = normalize(lightDir2 + viewingV);
    float3 reflectV = reflect(lightDir2, input.normal);
    lightIntensity2 = pow(saturate(dot(viewingV, HalfV)), 16);

    color = saturate((diffuseColor * lightIntensity) + (diffuseColor2 * lightIntensity2));

    color = color * textureColor;

    return color;
}
```

2) RTT에 유리와 얼음 Texture 적용

GlassShaderClass의 Layout 설정

```
// glassShader
polygonLayout[0].SemanticName = "POSITION";
polygonLayout[0].SemanticIndex = 0;
polygonLayout[0].Format = DXGI_FORMAT_R32G32B32_FLOAT;
polygonLayout[0].InputSlot = 0;
polygonLayout[0].AlignedByteOffset = 0;
polygonLayout[0].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
polygonLayout[0].InstanceDataStepRate = 0;

polygonLayout[1].SemanticName = "TEXCOORD";
polygonLayout[1].SemanticIndex = 0;
polygonLayout[1].Format = DXGI_FORMAT_R32G32_FLOAT;
polygonLayout[1].InputSlot = 0;
polygonLayout[1].AlignedByteOffset = D3D11_APPEND_ALIGNED_ELEMENT;
polygonLayout[1].InputSlotClass = D3D11_INPUT_PER_VERTEX_DATA;
polygonLayout[1].InstanceDataStepRate = 0;
```

Description 설정 (Sampler, Glass)

```
matrixBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
matrixBufferDesc.ByteWidth = sizeof(MatrixBufferType);
matrixBufferDesc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
matrixBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
matrixBufferDesc.MiscFlags = 0;
matrixBufferDesc.StructureByteStride = 0;

result = device->CreateBuffer(&matrixBufferDesc, NULL, &m_matrixBuffer);
if(FAILED(result))
{
    return false;
}

samplerDesc.Filter = D3D11_FILTER_MIN_MAG_MIP_LINEAR;
samplerDesc.AddressU = D3D11_TEXTURE_ADDRESS_CLAMP;
samplerDesc.AddressV = D3D11_TEXTURE_ADDRESS_CLAMP;
samplerDesc.AddressW = D3D11_TEXTURE_ADDRESS_WRAP;
samplerDesc.MipLODBias = 0.0f;
samplerDesc.MaxAnisotropy = 1;
samplerDesc.ComparisonFunc = D3D11_COMPARISON_ALWAYS;
samplerDesc.BorderColor[0] = 0;
samplerDesc.BorderColor[1] = 0;
samplerDesc.BorderColor[2] = 0;
samplerDesc.BorderColor[3] = 0;
samplerDesc.MinLOD = 0;
samplerDesc.MaxLOD = D3D11_FLOAT32_MAX;

result = device->CreateSamplerState(&samplerDesc, &m_sampleState);
if(FAILED(result))
{
    return false;
}

glassBufferDesc.Usage = D3D11_USAGE_DYNAMIC;
glassBufferDesc.ByteWidth = sizeof(GlassBufferType);
glassBufferDesc.BindFlags = D3D11_BIND_CONSTANT_BUFFER;
glassBufferDesc.CPUAccessFlags = D3D11_CPU_ACCESS_WRITE;
glassBufferDesc.MiscFlags = 0;
glassBufferDesc.StructureByteStride = 0;

result = device->CreateBuffer(&glassBufferDesc, NULL, &m_glassBuffer);
if(FAILED(result))
{
    return false;
}

return true;
```

GlassShaderClass의 Render()

```
bool GlassShaderClass::Render(ID3D11DeviceContext* deviceContext, int indexCount, XMMATRIX worldMatrix, XMMATRIX viewMatrix, XMMATRIX projectionMatrix,
                              ID3D11ShaderResourceView* colorTexture, ID3D11ShaderResourceView* normalTexture, ID3D11ShaderResourceView* refractionTexture, float refractionScale)
{
    bool result;


    // Set the shader parameters that it will use for rendering.
    result = SetShaderParameters(deviceContext, worldMatrix, viewMatrix, projectionMatrix, colorTexture, normalTexture, refractionTexture, refractionScale);
    if(!result)
    {
        return false;
    }

    // Now render the prepared buffers with the shader.
    RenderShader(deviceContext, indexCount);

    return true;
}
```

Glass 텍스처에 대한 Parameter 설정 (Map -> Unmap)
(color, normal, refraction 값)

```
    // Transpose the matrices to prepare them for the shader.
    worldMatrix = XMMatrixTranspose(worldMatrix);
    viewMatrix = XMMatrixTranspose(viewMatrix);
    projectionMatrix = XMMatrixTranspose(projectionMatrix);

    result = deviceContext->Map(m_matrixBuffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedResource);
    if(FAILED(result))
    {
        return false;
    }

    dataPtr = (MatrixBufferType*)mappedResource.pData;

    dataPtr->world = worldMatrix;
    dataPtr->view = viewMatrix;
    dataPtr->projection = projectionMatrix;

    deviceContext->Unmap(m_matrixBuffer, 0);

    bufferNumber = 0;

    deviceContext->VSSetConstantBuffers(bufferNumber, 1, &m_matrixBuffer);

    deviceContext->PSSetShaderResources(0, 1, &colorTexture);
    deviceContext->PSSetShaderResources(1, 1, &normalTexture);
    deviceContext->PSSetShaderResources(2, 1, &refractionTexture);

    result = deviceContext->Map(m_glassBuffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedResource);
    if(FAILED(result))
    {
        return false;
    }

    dataPtr2 = (GlassBufferType*)mappedResource.pData;
    dataPtr2->refractionScale = refractionScale;
    dataPtr2->padding = XMFLOAT3(0.0f, 0.0f, 0.0f);

    deviceContext->Unmap(m_glassBuffer, 0);

    bufferNumber = 0;

    deviceContext->PSSetConstantBuffers(bufferNumber, 1, &m_glassBuffer);

    return true;
```

Glass texture에 대한 정보 PixelShader로 넘겨주기

```hlsl
////////////////////////////////////////////////////////////////////////////////
// Pixel Shader
////////////////////////////////////////////////////////////////////////////////
float4 GlassPixelShader(PixelInputType input) : SV_TARGET
{
    float2 refractTexCoord;
    float4 normalMap;
    float3 normal;
    float4 refractionColor;
    float4 textureColor;
    float4 color;


    // Calculate the projected refraction texture coordinates.
    refractTexCoord.x = input.refractionPosition.x / input.refractionPosition.w / 2.0f + 0.55f;
    refractTexCoord.y = -input.refractionPosition.y / input.refractionPosition.w / 2.0f + 0.5f;

    // Sample the normal from the normal map texture.
    normalMap = normalTexture.Sample(SampleType, input.tex);

    // Expand the range of the normal from (0,1) to (-1,+1).
    normal = (normalMap.xyz * 5.0f) - 1;

    // Re-position the texture coordinate sampling position by the normal map value to simulate light distortion through glass.
    refractTexCoord = refractTexCoord + (normal.xy * refractionScale);

    // Sample the texture pixel from the refraction texture using the perturbed texture coordinates.
    refractionColor = refractionTexture.Sample(SampleType, refractTexCoord);

    // Sample the texture pixel from the glass color texture.
    textureColor = colorTexture.Sample(SampleType, input.tex);

    // Evenly combine the glass color and refraction value for the final color.
    color = lerp(refractionColor, textureColor, 0.5f);
    return color;
}
```

2. Constant Buffer를 활용한 동적장면 생성

Map - UnMap을 통해 PixelShader로 전달

```
bool GlassShaderClass::SetShaderParameters(ID3D11DeviceContext* deviceContext, XMMATRIX worldMatrix, XMMATRIX viewMatrix, XMMATRIX projectionMatrix,
                                           ID3D11ShaderResourceView* colorTexture, ID3D11ShaderResourceView* normalTexture, ID3D11ShaderResourceView* refractionTexture,
                                           float refractionScale)
{
    HRESULT result;
    D3D11_MAPPED_SUBRESOURCE mappedResource;
    MatrixBufferType* dataPtr;
    unsigned int bufferNumber;
    GlassBufferType* dataPtr2;


    // Transpose the matrices to prepare them for the shader.
    worldMatrix = XMMatrixTranspose(worldMatrix);
    viewMatrix = XMMatrixTranspose(viewMatrix);
    projectionMatrix = XMMatrixTranspose(projectionMatrix);

    result = deviceContext->Map(m_matrixBuffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedResource);
    if(FAILED(result))
    {
        return false;
    }

    dataPtr = (MatrixBufferType*)mappedResource.pData;

    dataPtr->world = worldMatrix;
    dataPtr->view = viewMatrix;
    dataPtr->projection = projectionMatrix;

    deviceContext->Unmap(m_matrixBuffer, 0);

    bufferNumber = 0;

    deviceContext->VSSetConstantBuffers(bufferNumber, 1, &m_matrixBuffer);

    deviceContext->PSSetShaderResources(0, 1, &colorTexture);
    deviceContext->PSSetShaderResources(1, 1, &normalTexture);
    deviceContext->PSSetShaderResources(2, 1, &refractionTexture);

    result = deviceContext->Map(m_glassBuffer, 0, D3D11_MAP_WRITE_DISCARD, 0, &mappedResource);
    if(FAILED(result))
    {
        return false;
    }

    dataPtr2 = (GlassBufferType*)mappedResource.pData;
    dataPtr2->refractionScale = refractionScale;
    dataPtr2->padding = XMFLOAT3(0.0f, 0.0f, 0.0f);
```

```
cbuffer GlassBuffer
{
    float refractionScale;
    float3 padding;
};
```

cbuffer로 refractionScale을 받아놓음

```
////////////////////////////////////////////////////////////////////////////////
// Pixel Shader
////////////////////////////////////////////////////////////////////////////////
float4 GlassPixelShader(PixelInputType input) : SV_TARGET
{
    float2 refractTexCoord;
    float4 normalMap;
    float3 normal;
    float4 refractionColor;
    float4 textureColor;
    float4 color;


    // Calculate the projected refraction texture coordinates.
    refractTexCoord.x = input.refractionPosition.x / input.refractionPosition.w / 2.0f + 0.55f;
    refractTexCoord.y = -input.refractionPosition.y / input.refractionPosition.w / 2.0f + 0.5f;

    // Sample the normal from the normal map texture.
    normalMap = normalTexture.Sample(SampleType, input.tex);

    // Expand the range of the normal from (0,1) to (-1,+1).
    normal = (normalMap.xyz * 5.0f) - 1;

    // Re-position the texture coordinate sampling position by the normal map value to simulate light distortion through glass.
    refractTexCoord = refractTexCoord + (normal.xy * refractionScale);

    // Sample the texture pixel from the refraction texture using the perturbed texture coordinates.
    refractionColor = refractionTexture.Sample(SampleType, refractTexCoord);

    // Sample the texture pixel from the glass color texture.
    textureColor = colorTexture.Sample(SampleType, input.tex);

    // Evenly combine the glass color and refraction value for the final color.
    color = lerp(refractionColor, textureColor, 0.5f);
    return color;
}
```

PixelShader에서 refractionScale을 다룸

application에서 refractionScale 지정

```
bool ApplicationClass::Render(float rotation)
{
    XMMATRIX worldMatrix, viewMatrix, projectionMatrix;
    static float refractionScale = 0.01f;
    bool result;


    // Set the refraction scale for the glass shader.
    //refractionScale
    refractionScale += 0.00001f;

    // Clear the buffers to begin the scene.
    m_Direct3D->BeginScene(0.0f, 0.0f, 0.0f, 1.0f);

    // Get the world, view, and projection matrices from the camera and d3d objects.
    m_Direct3D->GetWorldMatrix(worldMatrix);
    m_Camera->GetViewMatrix(viewMatrix);
    m_Direct3D->GetProjectionMatrix(projectionMatrix);

    // Rotate the world matrix by the rotation value so that the cube will spin.
    worldMatrix = XMMatrixRotationY(rotation);

    // Render the cube model using the texture shader.
    m_Model->Render(m_Direct3D->GetDeviceContext());

    result = m_TextureShader->Render(m_Direct3D->GetDeviceContext(), m_Model->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix, m_Model->GetTexture(0), m_Light->GetDirection(), m_Light->GetDiffuseColor(), m_Light2->GetDirection(), m_Light2->GetDiffuseColor());
    if(!result)
    {
        return false;
    }

    // Translate to back where the window model will be rendered.
    worldMatrix = XMMatrixTranslation(2.0f, 1.0f, -1.5f);
    worldMatrix *= XMMatrixScaling(0.5f, 0.5f, 1.f);


    // Render the window model using the glass shader.
    m_WindowModel->Render(m_Direct3D->GetDeviceContext());

    result = m_GlassShader->Render(m_Direct3D->GetDeviceContext(), m_WindowModel->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix, m_WindowModel->GetTexture(0),
                                   m_WindowModel->GetTexture(1), m_RenderTexture->GetShaderResourceView(), refractionScale);
    if(!result)
    {
        return false;
    }

    worldMatrix *= XMMatrixTranslation(-2, 0, 0.0f);
    // Render the window model using the glass shader.
    m_WindowModel2->Render(m_Direct3D->GetDeviceContext());

    result = m_GlassShader->Render(m_Direct3D->GetDeviceContext(), m_WindowModel2->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix, m_WindowModel2->GetTexture(0),
                                   m_WindowModel2->GetTexture(1), m_RenderTexture2->GetShaderResourceView(), refractionScale);
```

결과 - 굴절율이 서서히 증가

3. 키보드 입력에 따른 모델 회전

좌우 방향키를 이용하여 rotation 값 변경

```
bool ApplicationClass::Frame(InputClass* Input)
{
    static float rotation = 0.0f;
    bool result;


    // Check if the user pressed escape and wants to exit the application.
    if(Input->IsEscapePressed())
    {
        return false;
    }

    if (Input->IsLeftArrowPressed()) {
        rotation += 0.0174532925f * 0.6f;
    }

    if (Input->IsRightArrowPressed()) {
        rotation -= 0.0174532925f * 0.6f;
    }

     // Render the cube spinning scene to texture first.
    result = RenderSceneToTexture(rotation);
    if(!result)
    {
        return false;
    }

    return true;
}
```

Render() 시에 worldMatrix의 Y축을 변경하여 회전하게 변경함

```
bool ApplicationClass::RenderSceneToTexture(float rotation)
{
    XMMATRIX worldMatrix, viewMatrix, projectionMatrix;
    bool result;

    // Set the render target to be the render to texture and clear it.
    m_RenderTexture->SetRenderTarget(m_Direct3D->GetDeviceContext());
    m_RenderTexture->ClearRenderTarget(m_Direct3D->GetDeviceContext(), 0.0f, 0.0f, 0.0f, 1.0f);

    // Get the world, view, and projection matrices from the camera and d3d objects.
    m_Direct3D->GetWorldMatrix(worldMatrix);
    m_Camera->GetViewMatrix(viewMatrix);
    m_Direct3D->GetProjectionMatrix(projectionMatrix);

    // Rotate the world matrix by the rotation value so that the cube will spin.
    worldMatrix = XMMatrixRotationY(rotation);
    // worldMatrix *= XMMatrixTranslation(0, 0, 0);
    // Render the cube model using the texture shader.
    m_Model->Render(m_Direct3D->GetDeviceContext());

    result = m_TextureShader->Render(m_Direct3D->GetDeviceContext(), m_Model->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix,
                m_Model->GetTexture(0), m_Light->GetDirection(), m_Light->GetDiffuseColor(), m_Light2->GetDirection(), m_Light2->GetDiffuseColor());
    if(!result)
    {
        return false;
    }


    // Render the graphics scene.
    result = RenderSceneToTexture2(rotation);
    if (!result)
    {
        return false;
    }


    return true;
}
```

4. 두 개 이상의 RTT 활용

frame 에서 RendertoTexture 함수 호출

```
bool ApplicationClass::Frame(InputClass* Input)
{
    static float rotation = 0.0f;
    bool result;


    // Check if the user pressed escape and wants to exit the application.
    if(Input->IsEscapePressed())
    {
        return false;
    }

    if (Input->IsLeftArrowPressed()) {
        rotation += 0.0174532925f * 0.5f;
    }

    if (Input->IsRightArrowPressed()) {
        rotation -= 0.0174532925f * 0.5f;
    }

     // Render the cube spinning scene to texture first.
    result = RenderSceneToTexture(rotation);
    if(!result)
    {
        return false;
    }

    return true;
}
```

RendertoTexture 에서 RendertoTexture2 함수 호출

```
bool ApplicationClass::RenderSceneToTexture(float rotation)
{
    XMMATRIX worldMatrix, viewMatrix, projectionMatrix;
    bool result;

    // Set the render target to be the render to texture and clear it.
    m_RenderTexture->SetRenderTarget(m_Direct3D->GetDeviceContext());
    m_RenderTexture->ClearRenderTarget(m_Direct3D->GetDeviceContext(), 0.0f, 0.0f, 0.0f, 1.0f);

    // Get the world, view, and projection matrices from the camera and d3d objects.
    m_Direct3D->GetWorldMatrix(worldMatrix);
    m_Camera->GetViewMatrix(viewMatrix);
    m_Direct3D->GetProjectionMatrix(projectionMatrix);

    // Rotate the world matrix by the rotation value so that the cube will spin.
    worldMatrix = XMMatrixRotationY(rotation);
    worldMatrix += XMMatrixTranslation(0, 0, 0);
    // Render the cube model using the texture shader.
    m_Model->Render(m_Direct3D->GetDeviceContext());

    result = m_TextureShader->Render(m_Direct3D->GetDeviceContext(), m_Model->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix,
             m_Model->GetTexture(0), m_Light->GetDirection(), m_Light->GetDiffuseColor(), m_Light2->GetDirection(), m_Light2->GetDiffuseColor());
    if(!result)
    {
        return false;
    }


    // Render the graphics scene.
    result = RenderSceneToTexture2(rotation);
    if (!result)
    {
        return false;
    }


    return true;
}
```

RendertoTexture2 에서 Render 함수 호출

```
bool ApplicationClass::RenderSceneToTexture2(float rotation)
{
    XMMATRIX worldMatrix, viewMatrix, projectionMatrix;
    bool result;

    m_RenderTexture2->SetRenderTarget(m_Direct3D->GetDeviceContext());
    m_RenderTexture2->ClearRenderTarget(m_Direct3D->GetDeviceContext(), 0.0f, 0.0f, 0.0f, 1.0f);

    // Get the world, view, and projection matrices from the camera and d3d objects.
    m_Direct3D->GetWorldMatrix(worldMatrix);
    m_Camera->GetViewMatrix(viewMatrix);
    m_Direct3D->GetProjectionMatrix(projectionMatrix);

    // Render the window model using the glass shader.
    m_WindowModel->Render(m_Direct3D->GetDeviceContext());

    result = m_GlassShader->Render(m_Direct3D->GetDeviceContext(), m_WindowModel->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix,
                m_WindowModel->GetTexture(0), m_WindowModel->GetTexture(1), m_RenderTexture->GetShaderResourceView(), 0.01f);
    if (!result)
    {
        return false;
    }

    // Reset the render target back to the original back buffer and not the render to texture anymore.  And reset the viewport back to the original.

    m_Direct3D->SetBackBufferRenderTarget();
    m_Direct3D->ResetViewport();

    result = Render(rotation);
    if (!result)
    {
        return false;
    }
    return true;
}
```

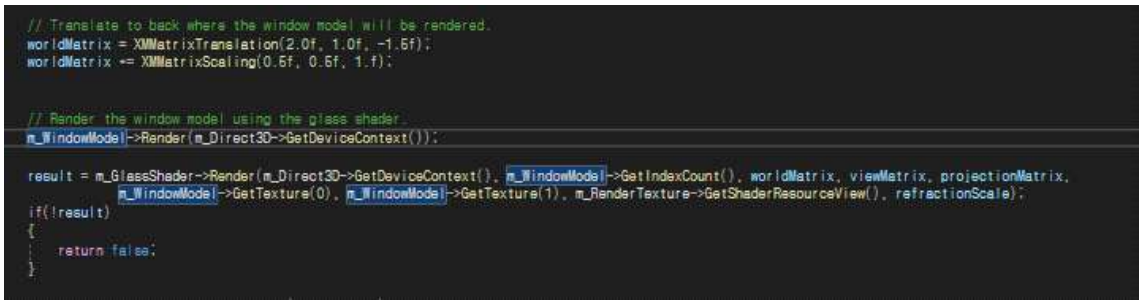
텍스쳐와 조명, 위치를 받아 model 그리기

```
bool ApplicationClass::Render(float rotation)
{
    XMMATRIX worldMatrix, viewMatrix, projectionMatrix;
    static float refractionScale = 0.01f;
    bool result;


    // Set the refraction scale for the glass shader.
    //refractionScale
    refractionScale += 0.00001f;

    // Clear the buffers to begin the scene.
    m_Direct3D->BeginScene(0.0f, 0.0f, 0.0f, 1.0f);

    // Get the world, view, and projection matrices from the camera and d3d objects.
    m_Direct3D->GetWorldMatrix(worldMatrix);
    m_Camera->GetViewMatrix(viewMatrix);
    m_Direct3D->GetProjectionMatrix(projectionMatrix);

    // Rotate the world matrix by the rotation value so that the cube will spin.
    worldMatrix = XMMatrixRotationY(rotation);

    // Render the cube model using the texture shader.
    m_Model->Render(m_Direct3D->GetDeviceContext());

    result = m_TextureShader->Render(m_Direct3D->GetDeviceContext(), m_Model->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix,
                m_Model->GetTexture(0), m_Light->GetDirection(), m_Light->GetDiffuseColor(), m_Light2->GetDirection(), m_Light2->GetDiffuseColor());
    if(!result)
    {
        return false;
    }
```

그리는 위치를 변경하여 Glass 텍스처의 windowModel 그리기

```
worldMatrix *= XMMatrixTranslation(-2, 0, 0.0f);
// Render the window model using the glass shader.
m_WindowModel2->Render(m_Direct3D->GetDeviceContext());

result = m_GlassShader->Render(m_Direct3D->GetDeviceContext(), m_WindowModel2->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix, m_WindowModel2->GetTexture(0),
                               m_WindowModel2->GetTexture(1), m_RenderTexture2->GetShaderResourceView(), refractionScale);



// Present the rendered scene to the screen.
m_Direct3D->EndScene();

return true;
```

그리는 위치를 변경하여 Ice 텍스처의 windowModel 그리기



```
// Translate to back where the window model will be rendered.
worldMatrix = XMMatrixTranslation(2.0f, 1.0f, -1.5f);
worldMatrix *= XMMatrixScaling(0.5f, 0.5f, 1.f);


// Render the window model using the glass shader.
m_WindowModel->Render(m_Direct3D->GetDeviceContext());

result = m_GlassShader->Render(m_Direct3D->GetDeviceContext(), m_WindowModel->GetIndexCount(), worldMatrix, viewMatrix, projectionMatrix,
          m_WindowModel->GetTexture(0), m_WindowModel->GetTexture(1), m_RenderTexture->GetShaderResourceView(), refractionScale);
if(!result)
{
    return false;
}
```

5. 조명 모델 구현

application.cpp에서 조명 객체 생성

```
    m_Light = new LightClass;
    m_Light->SetDiffuseColor(1.5f, 0.5f, 1.5f, 1.0f);
    m_Light->SetDirection(1.0f, 0.5f, 1.0f);


    m_Light2 = new LightClass;
    m_Light2->SetDiffuseColor(0.5f, 0.5f, 0.5f, 1.0f);
    m_Light2->SetDirection(0.0f, 0.0f, 1.0f);

    return true;
}
```

textureshaderclass.cpp 에서 텍스처 정보에 조명 정보 함께 넘겨주기

```
    dataPtr2 = (LightBufferType*)mappedResource.pData;
    dataPtr2->diffuseColor = diffuseColor;
    dataPtr2->lightDirection = lightDirection;
    dataPtr2->padding = 0.0f;

    dataPtr2->diffuseColor2 = diffuseColor2;
    dataPtr2->lightDirection2 = lightDirection2;
    dataPtr2->padding2 = 0.0f;

    deviceContext->Unmap(m_lightBuffer, 0);

    bufferNumber = 0;

    deviceContext->PSSetConstantBuffers(bufferNumber, 1, &m_lightBuffer);

    return true;
}
```

pixelshader에서 조명 정보 상수 버퍼로 받기

```
cbuffer LightBuffer
{
    float4 diffuseColor;
    float3 lightDirection;

    float4 diffuseColor2;
    float3 lightDirection2;
};
```

조명 하나는 일반 조명, 하나는 phong 조명을 연사하여 텍스쳐와 함께 반환

```
////////////////////////////////////////////////////////////////////////////////
// Pixel Shader
////////////////////////////////////////////////////////////////////////////////
float4 TexturePixelShader(PixelInputType input) : SV_TARGET
{
    float4 textureColor;

    float4 color;

    float3 lightDir;
    float3 lightDir2;

    float lightIntensity;
    float lightIntensity2;


    textureColor = shaderTexture1.Sample(SampleType, input.tex);

    lightDir = -lightDirection;
    lightIntensity = saturate(dot(input.normal, lightDir));

    lightDir2 = -lightDirection2;
    lightDir2 = normalize(lightDir2);


    float3 HalfV;
    float3 viewingV;
    viewingV = normalize(-input.normal.xyz);

    HalfV = normalize(lightDir2 + viewingV);
    float3 reflectV = reflect(lightDir2, input.normal);
    lightIntensity2 = pow(saturate(dot(viewingV, reflectV)), 16);

    color = saturate((diffuseColor * lightIntensity) + (diffuseColor2 * lightIntensity2));

    color = color * textureColor;

    return color;
}
```

결과: 조명이 오른쪽에 있고 phong조명은 가운데에 있어 색이 다르게 나옴