# Работа 1 (Tutorial01)

# Описание

* язык программирования - C/C++
* графическая библиотека - DirectX11
* среда разработки - Visual Studio 17
* целевая ос - Windows7,8,10

**Замечание: если** при линковке debug configuration возникает

warning LNK4075: ignoring '/EDITANDCONTINUE' due to '/SAFESEH' specification

происходит это из-за конфликта настроек компиляции. Появился в VS2012 в проектах, которые апдейтились из более ранних студий.

*Лекарство*: для каждого проекта в дебажной конфигурации в настройках проекта нужно открыть вкладку Linker->Advanced и выключить параметр ImageHasSafeExceptionHandlers (ибо он все равно смысла не имеет для debug'a).

# Изображение выглядит как снимок экрана Автоматически созданное описание

# Изображение выглядит как снимок экрана, компьютер, ноутбук, монитор Автоматически созданное описание

# Проект -> Назначить автозагружаемым проетом

HRESULT InitWindow( HINSTANCE hInstance, int nCmdShow )

{

…………………………………………………………………………………………

RECT rc = { 0, 0, 800, 600 };

AdjustWindowRect( &rc, WS\_OVERLAPPEDWINDOW, FALSE );

g\_hWnd = CreateWindow( L"TutorialWindowClass", L"Direct3D 11 Tutorial 1: Direct3D 11

…………………………………………………………………………………………

}

HRESULT InitDevice()

{

…………………………………………………………………………………………

// Create a render target view

ID3D11Texture2D\* pBackBuffer = nullptr;

hr = g\_pSwapChain->GetBuffer( 0, \_\_uuidof( ID3D11Texture2D ), reinterpret\_cast<void\*\*>( &pBackBuffer ) );

hr = g\_pd3dDevice->CreateRenderTargetView( pBackBuffer, nullptr, &g\_pRenderTargetView );

…………………………………………………………………………………………

g\_pImmediateContext->OMSetRenderTargets( 1, &g\_pRenderTargetView, nullptr );

…………………………………………………………………………………………

# }

void Render()

{

// Just clear the backbuffer

float ClearColor[4] = { 1.0f, 0.0f, 0.0f, 1.0f }; // RGBA

g\_pImmediateContext->ClearRenderTargetView(g\_pRenderTargetView, ClearColor);

//g\_pImmediateContext->ClearRenderTargetView(g\_pRenderTargetView, //Colors::MidnightBlue);

g\_pSwapChain->Present( 0, 0 );

}

# <https://docs.microsoft.com/en-us/windows/desktop/direct3d11/atoc-dx-graphics-direct3d-11>

# Задание

# Внести следующие изменения:

# В заголовке окна – ФИО

# Изменить размер и цвет окна

# Работа 2 (Tutorial02)

# Описание

# 

HRESULT InitDevice()

{

…………………………………………………………………………………………

// Create the vertex shader

hr = CompileShaderFromFile( L"Tutorial02.fx", "VS", "vs\_4\_0", &pVSBlob );

hr = g\_pd3dDevice->CreateVertexShader( pVSBlob->GetBufferPointer(), pVSBlob- >GetBufferSize(), nullptr, &g\_pVertexShader );

// Set the input layout

D3D11\_INPUT\_ELEMENT\_DESC layout[] =

{

{ "POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

};

UINT numElements = ARRAYSIZE( layout );

hr = g\_pd3dDevice->CreateInputLayout( layout, numElements, pVSBlob->GetBufferPointer(),

pVSBlob->GetBufferSize(), &g\_pVertexLayout );

g\_pImmediateContext->IASetInputLayout( g\_pVertexLayout );

// Create the pixel shader

hr = CompileShaderFromFile( L"Tutorial02.fx", "PS", "ps\_4\_0", &pPSBlob );

hr = g\_pd3dDevice->CreatePixelShader( pPSBlob->GetBufferPointer(), pPSBlob->GetBufferSize(), nullptr, &g\_pPixelShader );

// Set vertex buffer

SimpleVertex vertices[] =

{

XMFLOAT3( 0.0f, 0.5f, 0.5f ),

XMFLOAT3( 0.5f, -0.5f, 0.5f ),

XMFLOAT3( -0.5f, -0.5f, 0.5f ),

};

D3D11\_BUFFER\_DESC bd;

bd.ByteWidth = sizeof( SimpleVertex ) \* 3;

D3D11\_SUBRESOURCE\_DATA InitData;

InitData.pSysMem = vertices;

hr = g\_pd3dDevice->CreateBuffer( &bd, &InitData, &g\_pVertexBuffer );

g\_pImmediateContext->IASetVertexBuffers(0, 1, &g\_pVertexBuffer, &stride, &offset);

// Set primitive topology

g\_pImmediateContext->IASetPrimitiveTopology(D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST );

}

//--------------------------------------------------------------------------------------

// Vertex Shader

//--------------------------------------------------------------------------------------

float4 VS( float4 Pos : POSITION ) : SV\_POSITION

{

return Pos;

// return Pos + float4(0.2f, 0.2f, 0.0f, 1.0f);

// float3 p = 1.5f \* Pos.xyz;

// return float4(p, 1);

}

//--------------------------------------------------------------------------------------

// Pixel Shader

//--------------------------------------------------------------------------------------

float4 PS( float4 Pos : SV\_POSITION ) : SV\_Target

{

return float4( 1.0f, 1.0f, 0.0f, 1.0f ); // Yellow, with Alpha = 1

}

void Render()

{

g\_pImmediateContext->ClearRenderTargetView(g\_pRenderTargetView, Colors::MidnightBlue );

// Render a triangle

g\_pImmediateContext->VSSetShader( g\_pVertexShader, nullptr, 0 );

g\_pImmediateContext->PSSetShader( g\_pPixelShader, nullptr, 0 );

g\_pImmediateContext->Draw( 3, 0 );

g\_pSwapChain->Present( 0, 0 );

}

# Задание

# Внести следующие изменения:

# Треугольник заменить на четырехугольник

# Сдвинуть его и изменить размер (в вершинном шейдере)

# Изменить его цвет

# Работа 3 (Tutorial04)

# Описание

# 

**HRESULT InitDevice():**

// Define the input layout

D3D11\_INPUT\_ELEMENT\_DESC layout[] =

{

{ "POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

{ "COLOR", 0, DXGI\_FORMAT\_R32G32B32A32\_FLOAT, 0, 12, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

};

hr = g\_pd3dDevice->CreateInputLayout( layout, numElements, pVSBlob->GetBufferPointer(),

pVSBlob->GetBufferSize(), &g\_pVertexLayout );

g\_pImmediateContext->IASetInputLayout( g\_pVertexLayout );

// Create vertex buffer

struct SimpleVertex

{

XMFLOAT3 Pos;

XMFLOAT4 Color;

};

SimpleVertex vertices[] =

{

{ XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT4( 0.0f, 0.0f, 1.0f, 1.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT4( 0.0f, 1.0f, 0.0f, 1.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, 1.0f ), XMFLOAT4( 0.0f, 1.0f, 1.0f, 1.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, 1.0f ), XMFLOAT4( 1.0f, 0.0f, 0.0f, 1.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, -1.0f ), XMFLOAT4( 1.0f, 0.0f, 1.0f, 1.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, -1.0f ), XMFLOAT4( 1.0f, 1.0f, 0.0f, 1.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, 1.0f ), XMFLOAT4( 1.0f, 1.0f, 1.0f, 1.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, 1.0f ), XMFLOAT4( 0.0f, 0.0f, 0.0f, 1.0f ) },

};

D3D11\_BUFFER\_DESC bd;

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

D3D11\_SUBRESOURCE\_DATA InitData;

InitData.pSysMem = vertices;

hr = g\_pd3dDevice->CreateBuffer( &bd, &InitData, &g\_pVertexBuffer );

g\_pImmediateContext->IASetVertexBuffers( 0, 1, &g\_pVertexBuffer, &stride, &offset );

// Create index buffer

WORD indices[] =

{

3,1,0,

2,1,3,

0,5,4,

1,5,0,

3,4,7,

0,4,3,

1,6,5,

2,6,1,

2,7,6,

3,7,2,

6,4,5,

7,4,6,

};

bd.ByteWidth = sizeof( WORD ) \* 36;

InitData.pSysMem = indices;

hr = g\_pd3dDevice->CreateBuffer( &bd, &InitData, &g\_pIndexBuffer );

g\_pImmediateContext->IASetIndexBuffer( g\_pIndexBuffer, DXGI\_FORMAT\_R16\_UINT, 0 );

// Set primitive topology

g\_pImmediateContext->IASetPrimitiveTopology( D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST );

// Create the constant buffer

struct ConstantBuffer

{

XMMATRIX mWorld;

XMMATRIX mView;

XMMATRIX mProjection;

};

bd.ByteWidth = sizeof(ConstantBuffer);

hr = g\_pd3dDevice->CreateBuffer( &bd, nullptr, &g\_pConstantBuffer );

// Initialize the world matrix

g\_World = XMMatrixIdentity();

// Initialize the view matrix

XMVECTOR Eye = XMVectorSet( 0.0f, 1.0f, -5.0f, 0.0f );

XMVECTOR At = XMVectorSet( 0.0f, 0.0f, 1.0f, 0.0f );

XMVECTOR Up = XMVectorSet( 0.0f, 1.0f, 0.0f, 0.0f );

//XMVECTOR Eye = XMVectorSet(0.0f, 0.0f, -5.0f, 0.0f);

//XMVECTOR At = XMVectorSet(0.0f, 0.0f, 1.0f, 0.0f);

//XMVECTOR Up = XMVectorSet(0.0f, 1.0f, 0.0f, 0.0f);

g\_View = XMMatrixLookAtLH( Eye, At, Up );

// Initialize the projection matrix

g\_Projection = XMMatrixPerspectiveFovLH(XM\_PIDIV2, width / (FLOAT)height, 0.01f, 100.0f);

void Render()

{

// Update our time

static float t = 0.0f;

static ULONGLONG timeStart = 0;

ULONGLONG timeCur = GetTickCount64();

if( timeStart == 0 )

timeStart = timeCur;

t = ( timeCur - timeStart ) / 1000.0f;

// Animate the cube

g\_World = XMMatrixRotationY( t );

ConstantBuffer cb;

cb.mWorld = XMMatrixTranspose( g\_World );

cb.mView = XMMatrixTranspose( g\_View );

cb.mProjection = XMMatrixTranspose( g\_Projection );

g\_pImmediateContext->UpdateSubresource( g\_pConstantBuffer, 0, nullptr, &cb, 0, 0 );

// Render

g\_pImmediateContext->VSSetShader( g\_pVertexShader, nullptr, 0 );

g\_pImmediateContext->VSSetConstantBuffers( 0, 1, &g\_pConstantBuffer );

g\_pImmediateContext->PSSetShader( g\_pPixelShader, nullptr, 0 );

g\_pImmediateContext->DrawIndexed( 36, 0, 0 ); // 36 vertices needed for 12 triangles in a triangle list

// Present our back buffer to our front buffer

g\_pSwapChain->Present( 0, 0 );

}

**Shaders:**

//--------------------------------------------------------------------------------------

// Constant Buffer Variables

//--------------------------------------------------------------------------------------

cbuffer ConstantBuffer : register( b0 )

{

matrix World;

matrix View;

matrix Projection;

}

//--------------------------------------------------------------------------------------

struct VS\_OUTPUT

{

float4 Pos : SV\_POSITION;

float4 Color : COLOR0;

};

//--------------------------------------------------------------------------------------

// Vertex Shader

//--------------------------------------------------------------------------------------

VS\_OUTPUT VS( float4 Pos : POSITION, float4 Color : COLOR )

{

VS\_OUTPUT output = (VS\_OUTPUT)0;

output.Pos = mul( Pos, World );

output.Pos = mul( output.Pos, View );

output.Pos = mul( output.Pos, Projection );

output.Color = Color;

return output;

}

//--------------------------------------------------------------------------------------

// Pixel Shader

//--------------------------------------------------------------------------------------

float4 PS( VS\_OUTPUT input ) : SV\_Target

{

return input.Color;

}

# Задание

# Внести следующие изменения:

1. Добавить перемещение объекта по оси Z, путем добавления в буфер констант матрицы Translation. Сделать это перемещение циклическим (sin).

# Работа 4 (Tutorial06)

# Описание

# Изображение выглядит как снимок экрана, монитор, электроника Автоматически созданное описание

**HRESULT InitDevice():**

// Create vertex buffer

SimpleVertex vertices[] =

{

{ XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, 1.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, 1.0f ), XMFLOAT3( 0.0f, 1.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, -1.0f ), XMFLOAT3( 0.0f, -1.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, -1.0f ), XMFLOAT3( 0.0f, -1.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, 1.0f ), XMFLOAT3( 0.0f, -1.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, 1.0f ), XMFLOAT3( 0.0f, -1.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, 1.0f ), XMFLOAT3( -1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, -1.0f ), XMFLOAT3( -1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT3( -1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, 1.0f ), XMFLOAT3( -1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, 1.0f ), XMFLOAT3( 1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, -1.0f ), XMFLOAT3( 1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT3( 1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, 1.0f ), XMFLOAT3( 1.0f, 0.0f, 0.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, -1.0f ), XMFLOAT3( 0.0f, 0.0f, -1.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, -1.0f ), XMFLOAT3( 0.0f, 0.0f, -1.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT3( 0.0f, 0.0f, -1.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT3( 0.0f, 0.0f, -1.0f ) },

{ XMFLOAT3( -1.0f, -1.0f, 1.0f ), XMFLOAT3( 0.0f, 0.0f, 1.0f ) },

{ XMFLOAT3( 1.0f, -1.0f, 1.0f ), XMFLOAT3( 0.0f, 0.0f, 1.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, 1.0f ), XMFLOAT3( 0.0f, 0.0f, 1.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, 1.0f ), XMFLOAT3( 0.0f, 0.0f, 1.0f ) },

};

WORD indices[] =

{

3,1,0,

2,1,3,

6,4,5,

7,4,6,

11,9,8,

10,9,11,

14,12,13,

15,12,14,

19,17,16,

18,17,19,

22,20,21,

23,20,22

};

**void Render()**

{

// Update our time

static float t = 0.0f;

static ULONGLONG timeStart = 0;

ULONGLONG timeCur = GetTickCount64();

if( timeStart == 0 )

timeStart = timeCur;

t = ( timeCur - timeStart ) / 1000.0f;

// Rotate cube around the origin

g\_World = XMMatrixRotationY( t );

// Setup our lighting parameters

XMFLOAT4 vLightDirs[2] =

{

XMFLOAT4( -0.577f, 0.577f, -0.577f, 1.0f ),

XMFLOAT4( 0.0f, 0.0f, -1.0f, 1.0f ),

};

XMFLOAT4 vLightColors[2] =

{

XMFLOAT4( 0.5f, 0.5f, 0.5f, 1.0f ),

XMFLOAT4( 0.5f, 0.0f, 0.0f, 1.0f )

};

// Rotate the second light around the origin

XMMATRIX mRotate = XMMatrixRotationY( -2.0f \* t );

XMVECTOR vLightDir = XMLoadFloat4( &vLightDirs[1] );

vLightDir = XMVector3Transform( vLightDir, mRotate );

XMStoreFloat4( &vLightDirs[1], vLightDir );

// Clear the back buffer

g\_pImmediateContext->ClearRenderTargetView(g\_pRenderTargetView,Colors::MidnightBlue);

// Clear the depth buffer to 1.0 (max depth)

g\_pImmediateContext->ClearDepthStencilView( …, 3D11\_CLEAR\_DEPTH, 1.0f, 0 );

//

// Update matrix variables and lighting variables

//

ConstantBuffer cb1;

cb1.mWorld = XMMatrixTranspose( g\_World );

cb1.mView = XMMatrixTranspose( g\_View );

cb1.mProjection = XMMatrixTranspose( g\_Projection );

cb1.vLightDir[0] = vLightDirs[0];

cb1.vLightDir[1] = vLightDirs[1];

cb1.vLightColor[0] = vLightColors[0];

cb1.vLightColor[1] = vLightColors[1];

cb1.vOutputColor = XMFLOAT4(0, 0, 0, 0);

// Render the cube

g\_pImmediateContext->VSSetShader( g\_pVertexShader, nullptr, 0 );

g\_pImmediateContext->VSSetConstantBuffers( 0, 1, &g\_pConstantBuffer );

g\_pImmediateContext->PSSetShader( g\_pPixelShader, nullptr, 0 );

g\_pImmediateContext->PSSetConstantBuffers( 0, 1, &g\_pConstantBuffer );

g\_pImmediateContext->DrawIndexed( 36, 0, 0 );

g\_pImmediateContext->UpdateSubresource(g\_pConstantBuffer, 0, nullptr, &cb1, 0, 0 );

// Render each light

for( int m = 0; m < 2; m++ )

{

XMMATRIX mLight = XMMatrixTranslationFromVector( 5.0f\*XMLoadFloat4( &vLightDirs[m]));

XMMATRIX mLightScale = XMMatrixScaling( 0.2f, 0.2f, 0.2f );

mLight = mLightScale \* mLight;

// Update the world variable to reflect the current light

cb1.mWorld = XMMatrixTranspose( mLight );

cb1.vOutputColor = vLightColors[m];

g\_pImmediateContext->UpdateSubresource(g\_pConstantBuffer,0,nullptr, &cb1, 0, 0 );

g\_pImmediateContext->PSSetShader( g\_pPixelShaderSolid, nullptr, 0 );

g\_pImmediateContext->DrawIndexed( 36, 0, 0 );

}

// Present our back buffer to our front buffer

g\_pSwapChain->Present( 0, 0 );

}

**Shaders:**

//--------------------------------------------------------------------------------------

// Constant Buffer Variables

//--------------------------------------------------------------------------------------

cbuffer ConstantBuffer : register( b0 )

{

matrix World;

matrix View;

matrix Projection;

float4 vLightDir[2];

float4 vLightColor[2];

float4 vOutputColor;

}

//--------------------------------------------------------------------------------------

struct VS\_INPUT

{

float4 Pos : POSITION;

float3 Norm : NORMAL;

};

struct PS\_INPUT

{

float4 Pos : SV\_POSITION;

float3 Norm : TEXCOORD0;

};

//--------------------------------------------------------------------------------------

// Vertex Shader

//--------------------------------------------------------------------------------------

PS\_INPUT VS( VS\_INPUT input )

{

PS\_INPUT output = (PS\_INPUT)0;

output.Pos = mul( input.Pos, World );

output.Pos = mul( output.Pos, View );

output.Pos = mul( output.Pos, Projection );

output.Norm = mul( float4( input.Norm, 1 ), World ).xyz;

return output;

}

//--------------------------------------------------------------------------------------

// Pixel Shader

//--------------------------------------------------------------------------------------

float4 PS( PS\_INPUT input) : SV\_Target

{

float4 finalColor = 0;

//do NdotL lighting for 2 lights

for(int i=0; i<2; i++)

{

finalColor += saturate( dot( (float3)vLightDir[i],input.Norm) \* vLightColor[i] );

}

finalColor.a = 1;

return finalColor;

}

//--------------------------------------------------------------------------------------

// PSSolid - render a solid color

//--------------------------------------------------------------------------------------

float4 PSSolid( PS\_INPUT input) : SV\_Target

{

return vOutputColor;

}

# Задание

**Внести следующие изменения:**

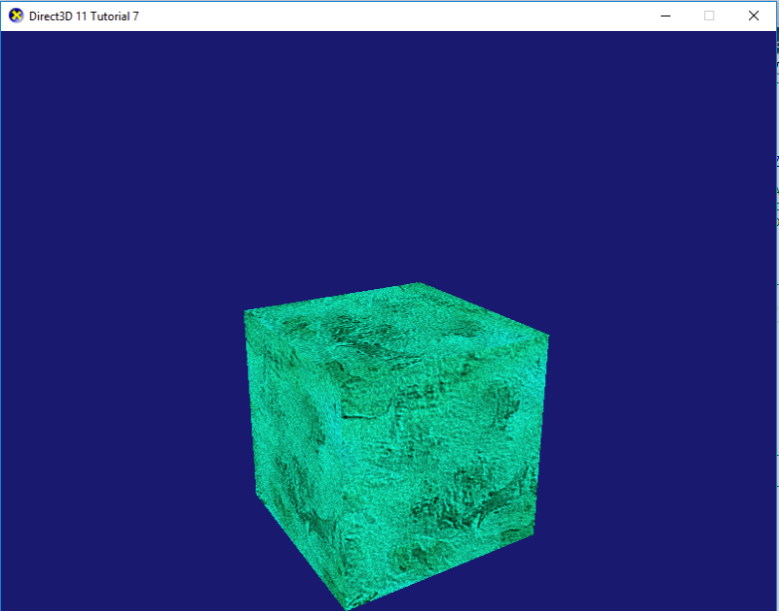
1. Убрать неподвижный объект (первый источник)
2. Центральный куб не вращать
3. Диффузное освещение от второго источника заменить на specular (для вычисления вектора отражения используйте функцию reflect. Возведение в степень – pow.)
4. Приблизить камеру и вращающийся источник к кубу
5. Камеру расположить на оси Z

**Изображение выглядит как снимок экрана

Автоматически созданное описание**

# Работа 5 (Tutorial07)

# Описание

****

**HRESULT InitDevice():**

struct SimpleVertex

{

XMFLOAT3 Pos;

XMFLOAT2 Tex;

};

D3D11\_INPUT\_ELEMENT\_DESC layout[] =

{

{ "POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

{ "TEXCOORD", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 12, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

};

SimpleVertex vertices[] =

{

{ XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT2( 1.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT2( 0.0f, 0.0f ) },

{ XMFLOAT3( 1.0f, 1.0f, 1.0f ), XMFLOAT2( 0.0f, 1.0f ) },

{ XMFLOAT3( -1.0f, 1.0f, 1.0f ), XMFLOAT2( 1.0f, 1.0f ) },

// Load the Texture

hr = CreateDDSTextureFromFile( g\_pd3dDevice, L"seafloor.dds", nullptr, &g\_pTextureRV );

// Create the sample state

D3D11\_SAMPLER\_DESC sampDesc;

ZeroMemory( &sampDesc, sizeof(sampDesc) );

sampDesc.Filter = D3D11\_FILTER\_MIN\_MAG\_MIP\_LINEAR;

sampDesc.AddressU = D3D11\_TEXTURE\_ADDRESS\_WRAP;

sampDesc.AddressV = D3D11\_TEXTURE\_ADDRESS\_WRAP;

sampDesc.AddressW = D3D11\_TEXTURE\_ADDRESS\_WRAP;

sampDesc.ComparisonFunc = D3D11\_COMPARISON\_NEVER;

sampDesc.MinLOD = 0;

sampDesc.MaxLOD = D3D11\_FLOAT32\_MAX;

hr = g\_pd3dDevice->CreateSamplerState( &sampDesc, &g\_pSamplerLinear );

**void Render()**

{

g\_pImmediateContext->PSSetShaderResources( 0, 1, &g\_pTextureRV );

g\_pImmediateContext->PSSetSamplers( 0, 1, &g\_pSamplerLinear );

**Shaders:**

Texture2D txDiffuse : register( t0 );

SamplerState samLinear : register( s0 );

cbuffer cbNeverChanges : register( b0 )

{

matrix View;

};

cbuffer cbChangeOnResize : register( b1 )

{

matrix Projection;

};

cbuffer cbChangesEveryFrame : register( b2 )

{

matrix World;

float4 vMeshColor;

};

//--------------------------------------------------------------------------------------

struct VS\_INPUT

{

float4 Pos : POSITION;

float2 Tex : TEXCOORD0;

};

struct PS\_INPUT

{

float4 Pos : SV\_POSITION;

float2 Tex : TEXCOORD0;

};

//--------------------------------------------------------------------------------------

// Vertex Shader

//--------------------------------------------------------------------------------------

PS\_INPUT VS( VS\_INPUT input )

{

PS\_INPUT output = (PS\_INPUT)0;

output.Pos = mul( input.Pos, World );

output.Pos = mul( output.Pos, View );

output.Pos = mul( output.Pos, Projection );

output.Tex = input.Tex;

return output;

}

//--------------------------------------------------------------------------------------

// Pixel Shader

//--------------------------------------------------------------------------------------

float4 PS( PS\_INPUT input) : SV\_Target

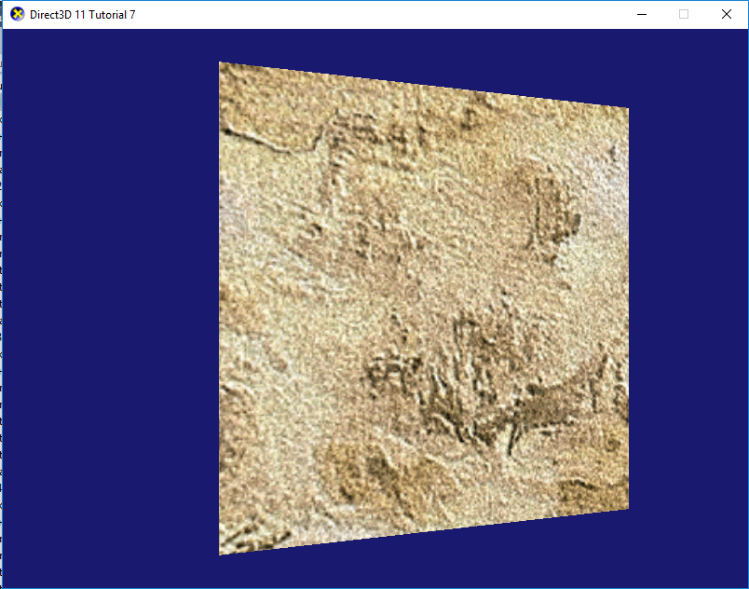
{

return txDiffuse.Sample( samLinear, input.Tex ) \* vMeshColor;

}

Задание

Для освещения использовать bump mapping

 Изображение выглядит как здание

Автоматически созданное описание

struct SimpleVertex

{

XMFLOAT3 Pos;

XMFLOAT2 Tex;

XMFLOAT3 Normal;

XMFLOAT3 Tangent;

};

D3D11\_INPUT\_ELEMENT\_DESC layout[] =

{

{ "POSITION", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 0, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

{ "TEXCOORD", 0, DXGI\_FORMAT\_R32G32\_FLOAT, 0, 12, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

///////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*new\*\*\*\*\*\*\*\*\*\*\*\*\*\*////////////////////

{ "NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 20, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0},

{ "TANGENT", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 32, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0} };

SimpleVertex vertices[] =

{

{ XMFLOAT3( -1.0f, -1.0f, -1.0f ), XMFLOAT2( 0.0f, 1.0f ), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3( 1.0f, -1.0f, -1.0f ), XMFLOAT2( 1.0f, 1.0f ), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3( 1.0f, 1.0f, -1.0f ), XMFLOAT2( 1.0f, 0.0f ), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

{ XMFLOAT3( -1.0f, 1.0f, -1.0f ), XMFLOAT2( 0.0f, 0.0f ), XMFLOAT3(0.0f, 0.0f, -1.0f), XMFLOAT3(1.0f, 0.0f, 0.0f) },

};

WORD indices[] =

{

3,1,0,

2,1,3,

};

struct VS\_INPUT

{

///////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*new\*\*\*\*\*\*\*\*\*\*\*\*\*\*////////////////////

float3 normal : NORMAL;

float3 tangent : TANGENT;

};

struct PS\_INPUT

{

///////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*new\*\*\*\*\*\*\*\*\*\*\*\*\*\*////////////////////

float3 normal : NORMAL;

float3 tangent : TANGENT;

};

//--------------------------------------------------------------------------------------

// Vertex Shader

//--------------------------------------------------------------------------------------

PS\_INPUT VS( VS\_INPUT input )

{

///////////////\*\*\*\*\*\*\*\*\*\*\*\*\*\*new\*\*\*\*\*\*\*\*\*\*\*\*\*\*////////////////////

output.normal = mul(input.normal, World);

output.tangent = mul(input.tangent, World);

}

float f(float4 col)

{

return (col.x + col.y + col.z) / 3.0;

}

//--------------------------------------------------------------------------------------

// Pixel Shader

//--------------------------------------------------------------------------------------

float4 PS( PS\_INPUT input) : SV\_Target

{

float2 t = input.Tex;

float2 dx = float2(1.0 / 256.0, 0.0);

float2 dy = float2(0.0, 1.0 / 256.0);

float3 N;

float scale = 12.0;

N.x=scale\*(f(txDiffuse.Sample(samLinear,t+dx))-f(txDiffuse.Sample(samLinear,t-dx)));

N.y=scale\*(f(txDiffuse.Sample(samLinear,t+dy))-f(txDiffuse.Sample(samLinear,t-dy)));

N.z = 1.0;

normalize(N);

//Create the biTangent

float3 biTangent = cross(input.normal, input.tangent);

//Create the "Texture Space"

float3x3 texSpace = float3x3(input.tangent, biTangent, input.normal);

//Convert normal from normal map to texture space and store in input.normal

float3 normal = normalize(mul(N, texSpace));

float4 finalColor = txDiffuse.Sample(samLinear, t);

finalColor = saturate(dot(float3(0.0, 0.0, -1.0), normal) \* finalColor);

return finalColor;

}