|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** \_***ИУ-КФ «Информатики и управления»*\_\_**\_\_\_\_\_\_\_\_\_\_\_\_

**КАФЕДРА** \_\_***ИУ5-КФ «Системы обработки информации»***\_\_\_\_\_\_\_\_\_\_

**ДОМАШНЕЕ ЗАДАНИЕ**

**«DirectX Math»**

**ДИСЦИПЛИНА: «Геометрическое моделирование»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. САПР.Б-61 | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_Комаров Д.С.\_\_\_\_\_)  (Подпись) (Ф.И.О.) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (\_\_Гуркина Е.Д.\_\_)  (Подпись) (Ф.И.О.) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга , 2019

**Цели:**

1. Ознакомиться с математической библиотекой DirectX directxmath;

2. Реализовать вершинные и пиксельные шейдеры и реализовать освещение

**Выполнение:**

1. Ознакомиться с теоретическим материалом;

2. Выполнить основные задания;

3. Предоставить отчет, по каждому заданию содержащий: формулировку

задания, исходный код программы, скриншоты работающей программы (один

или несколько, если необходимо);

4. Ответить на вопросы преподавателя.

**Выполнение лабораторной работы:**

Приложение реализовано на основе приложения из второй лабораторной работы:

**ЗАДАНИЕ 1**

Разработать приложение, выводящее на экран куб, вращающийся вокруг оси Y. При нажатии на клавишу «+» размер его увеличивается вдвое, при нажатии «-» уменьшается

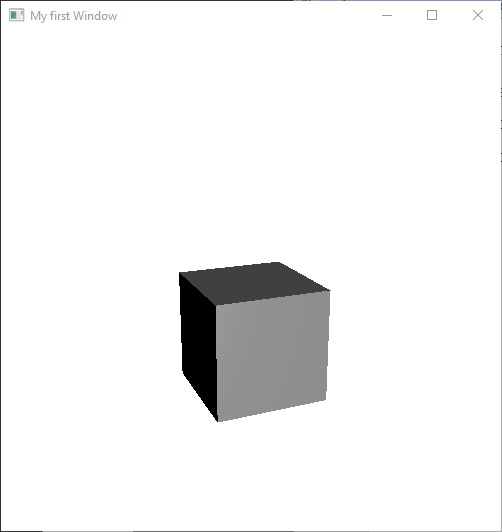


Рисунок 1 – Вращающийся куб

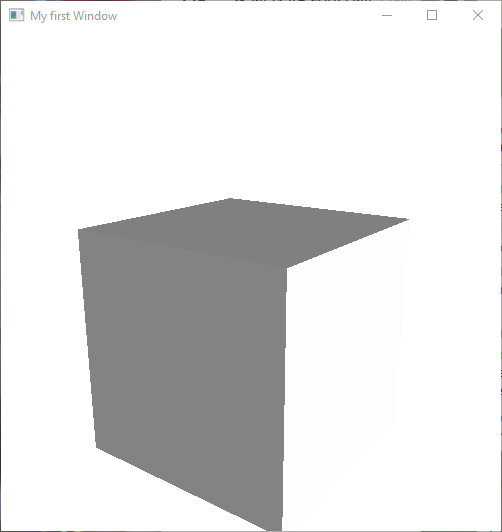


Рисунок 2 – Увеличенный посредством нажатия на «+» вращающийся куб

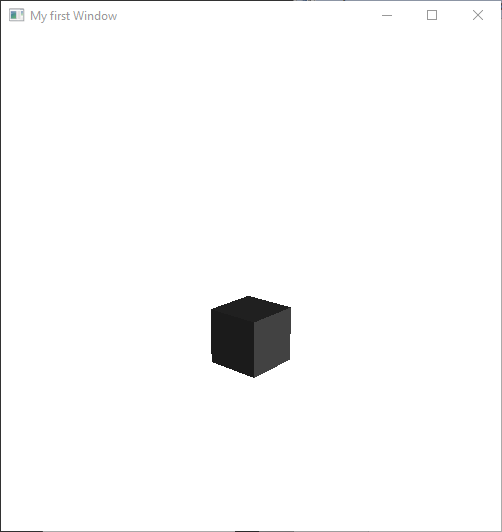


Рисунок 3 –Уменьшенный посредством нажатия на «-» вращающийся куб

**ЗАДАНИЕ 2**

Дополнить приложение из задания 1 реализацией освещения по модели Блинна-Фонга. Тип источника — точечный. Освещение производить на пиксельном шейдере.



Рисунок 4 – Квадрат с реализацией освещения Блинна-Фонга

#include <Windows.h>

#include <iostream>

#include <stdlib.h>

#include <string>

#include <tchar.h>

#include <d3d11.h>

#include <D3DCompiler.h>

#include <DirectXMath.h>

using namespace DirectX;

using namespace std;

float red = 1.0f;

float green = 1.0f;

float blue = 1.0f;

float alpha = 1.0f;

TCHAR szClassName[] = \_T("TestClass");

TCHAR szTitle[] = \_T("My first Window");

HWND hWnd;

ID3D11RenderTargetView\* renderTargetView = NULL;

D3D\_DRIVER\_TYPE driverType = D3D\_DRIVER\_TYPE\_NULL;

D3D\_FEATURE\_LEVEL featureLevel = D3D\_FEATURE\_LEVEL\_11\_0;

ID3D11Device\* device = NULL;

ID3D11DeviceContext\* immediateContext = NULL;

IDXGISwapChain\* swapChain = NULL;

HRESULT InitDevice();

ID3D11VertexShader\* vertexShader = NULL;

ID3D11PixelShader\* pixelShader = NULL;

ID3D11PixelShader\* pixelShaderSolid = NULL;

ID3D11InputLayout\* vertexLayout = NULL;

ID3D11Buffer\* vertexBuffer = NULL;

ID3D11Buffer\* constantBuffer = NULL;

ID3D11Buffer\* indexBuffer = NULL;

ID3D11Texture2D\* depthStencil = NULL;

ID3D11DepthStencilView\* depthStencilView = NULL;

XMMATRIX world;

XMMATRIX view;

XMMATRIX projection;

XMFLOAT4 lightDirs;

XMFLOAT4 lightColors;

FLOAT time = 0.0f;

FLOAT \_x = 1.0f;

FLOAT \_y = 1.0f;

FLOAT \_z = 1.0f;

typedef struct float4 {

float x;

float y;

float z;

float a;

} float4;

struct SimpleVertex

{

XMFLOAT3 Pos;

XMFLOAT3 Normal;// Каждая вершина содержит информацию о цвете

//XMFLOAT4 Color;

};

struct ConstantBuffer

{

XMMATRIX mWorld;// Матрица мира

XMMATRIX mView;// Матрица вида

XMMATRIX mProjection;// Матрица проекции

XMFLOAT4 vLightDir;//Направление света

XMFLOAT4 vLightColor;//Цвет источника

XMFLOAT4 vOutputColor;//Активный цвет

};

struct cst

{

float r; float g; float b; float a;

};

void SetMatrixes(float x, float y, float z)

{

// Обновление переменной-времени

static float time = 0.0f;

if (driverType == D3D\_DRIVER\_TYPE\_REFERENCE)

{

time += (float)XM\_PI \* 0.0125f;

}

else

{

static DWORD dwTimeStart = 0;

DWORD dwTimeCur = GetTickCount();

if (dwTimeStart == 0)

dwTimeStart = dwTimeCur;

time = (dwTimeCur - dwTimeStart) / 1000.0f;

}

XMMATRIX mSpin = XMMatrixRotationY(time \* 2);

XMMATRIX mScale = XMMatrixScaling(x, y, z);

// Результирующая матрица

world = mScale \* mSpin;

// Обновление константного буфера

ConstantBuffer cb;

cb.mWorld = XMMatrixTranspose(world);

cb.mView = XMMatrixTranspose(view);

cb.mProjection = XMMatrixTranspose(projection);

immediateContext->UpdateSubresource(constantBuffer, 0, NULL, &cb, 0, 0);

}

LRESULT CALLBACK WndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

{

PAINTSTRUCT ps;

HDC hdc;

switch (uMsg)

{

case WM\_CLOSE:

{

if (MessageBox(hWnd, \_T("Вы действительно хотите выйти?"), \_T("Выход"), MB\_ICONQUESTION | MB\_YESNO) == IDYES)

{

PostQuitMessage(0);

}

return 0;

}

case WM\_SIZE:

{

string sizestr = to\_string(LOWORD(lParam)) + "x" + to\_string(HIWORD(lParam));

std::wstring sizewstr = std::wstring(sizestr.begin(), sizestr.end());

LPCWSTR sizelpcwstr = sizewstr.c\_str();;

//SetWindowText(hWnd, sizelpcwstr);

return 0;

}

case WM\_DESTROY:

{

PostQuitMessage(0);

return 0;

}

case WM\_KEYDOWN:

{

if (wParam == 82)

{

red = 1.0f; green = 0.0f; blue = 0.0f;

}

if (wParam == 71)

{

red = 0.0f; green = 1.0f; blue = 0.0f;

}

if (wParam == 66)

{

red = 0.0f; green = 0.0f; blue = 1.0f;

}

/\*if (wParam == 107)

{

SetMatrixes(2.0f, 2.0f, 2.0f);

}

if (wParam == 108)

{

SetMatrixes(0.5f, 0.5f, 0.5f);

}\*/

if (wParam == 107)

{

\_x = \_x \* 2;

\_y = \_y \* 2;

\_z = \_z \* 2;

}

if (wParam == 109)

{

\_x = \_x / 2;

\_y = \_y / 2;

\_z = \_z / 2;

}

break;

}

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

EndPaint(hWnd, &ps);

break;

}

return DefWindowProc(hWnd, uMsg, wParam, lParam);

}

HRESULT InitDevice()

{/\*

HRESULT hr;

RECT rc;

GetClientRect(hWnd, &rc);

UINT width = rc.right - rc.left;

UINT height = rc.bottom - rc.top;

DXGI\_SWAP\_CHAIN\_DESC sd;

ZeroMemory(&sd, sizeof(sd));

sd.BufferCount = 1;

sd.BufferDesc.Width = width;

sd.BufferDesc.Height = height;

sd.BufferDesc.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;

sd.BufferDesc.RefreshRate.Numerator = 60;

sd.BufferDesc.RefreshRate.Denominator = 1;

sd.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

sd.OutputWindow = hWnd;

sd.SampleDesc.Count = 1;

sd.SampleDesc.Quality = 0;

sd.SwapEffect = DXGI\_SWAP\_EFFECT\_DISCARD;

sd.Windowed = TRUE;

hr = D3D11CreateDeviceAndSwapChain(

NULL,

D3D\_DRIVER\_TYPE\_HARDWARE,

NULL, 0, NULL, NULL,

D3D11\_SDK\_VERSION,

&sd,

&swapChain,

&device,

&featureLevel,

&immediateContext);

ID3D11Texture2D\* backBuffer;

swapChain->GetBuffer(0, \_\_uuidof(ID3D11Texture2D), (LPVOID\*)&backBuffer);

device->CreateRenderTargetView(backBuffer, NULL, &renderTargetView);

immediateContext->OMSetRenderTargets(1, &renderTargetView, NULL);

D3D11\_VIEWPORT vp;

vp.Width = (FLOAT)width;

vp.Height = (FLOAT)height;

vp.MinDepth = 0.0f;

vp.MaxDepth = 1.0f;

vp.TopLeftX = 0;

vp.TopLeftY = 0;

immediateContext->RSSetViewports(1, &vp);

return S\_OK;\*/

HRESULT hr;

RECT rc;

GetClientRect(hWnd, &rc);

UINT width = rc.right - rc.left;

UINT height = rc.bottom - rc.top;

//SwapChain

DXGI\_SWAP\_CHAIN\_DESC sd;

ZeroMemory(&sd, sizeof(sd));

sd.BufferCount = 1;

sd.BufferDesc.Width = width;

sd.BufferDesc.Height = height;

sd.BufferDesc.Format = DXGI\_FORMAT\_R8G8B8A8\_UNORM;

sd.BufferDesc.RefreshRate.Numerator = 60;

sd.BufferDesc.RefreshRate.Denominator = 1;

sd.BufferUsage = DXGI\_USAGE\_RENDER\_TARGET\_OUTPUT;

sd.OutputWindow = hWnd;

sd.SampleDesc.Count = 1;

sd.SampleDesc.Quality = 0;

sd.SwapEffect = DXGI\_SWAP\_EFFECT\_DISCARD;

sd.Windowed = TRUE;

//Создание основных сущеностей

hr = D3D11CreateDeviceAndSwapChain(

NULL,

D3D\_DRIVER\_TYPE\_HARDWARE,

NULL,

0,

NULL,

NULL,

D3D11\_SDK\_VERSION,

&sd,

&swapChain,

&device,

&featureLevel,

&immediateContext

);

ID3D11Texture2D\* pBackBuffer;

swapChain->GetBuffer(0, \_\_uuidof(ID3D11Texture2D), (LPVOID\*)&pBackBuffer);

device->CreateRenderTargetView(pBackBuffer, NULL, &renderTargetView);

pBackBuffer->Release();

// Создание текстуры-описание буфера глубин

D3D11\_TEXTURE2D\_DESC descDepth;

ZeroMemory(&descDepth, sizeof(descDepth));

descDepth.Width = width;

descDepth.Height = height;

descDepth.MipLevels = 1;

descDepth.ArraySize = 1;

descDepth.Format = DXGI\_FORMAT\_D24\_UNORM\_S8\_UINT;

descDepth.SampleDesc.Count = 1;

descDepth.SampleDesc.Quality = 0;

descDepth.Usage = D3D11\_USAGE\_DEFAULT;

descDepth.BindFlags = D3D11\_BIND\_DEPTH\_STENCIL;

descDepth.CPUAccessFlags = 0;

descDepth.MiscFlags = 0;

//Создание объекта текстуры

hr = device->CreateTexture2D(&descDepth, NULL, &depthStencil);

if (FAILED(hr)) return hr;

D3D11\_DEPTH\_STENCIL\_VIEW\_DESC descDSV;

ZeroMemory(&descDSV, sizeof(descDSV));

descDSV.Format = descDepth.Format;

descDSV.ViewDimension = D3D11\_DSV\_DIMENSION\_TEXTURE2D;

descDSV.Texture2D.MipSlice = 0;

hr = device->CreateDepthStencilView(depthStencil, &descDSV, &depthStencilView);

if (FAILED(hr)) return hr;

immediateContext->OMSetRenderTargets(1, &renderTargetView, depthStencilView);

D3D11\_VIEWPORT vp;

vp.Width = (FLOAT)width;

vp.Height = (FLOAT)height;

vp.MinDepth = 0.0f;

vp.MaxDepth = 1.0f;

vp.TopLeftX = 0;

vp.TopLeftY = 0;

immediateContext->RSSetViewports(1, &vp);

return S\_OK;

}

void Render()

{

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

immediateContext->ClearRenderTargetView(renderTargetView, ClearColor);

immediateContext->ClearDepthStencilView(depthStencilView, D3D11\_CLEAR\_DEPTH, 1.0f, 0);

lightDirs = XMFLOAT4(0.5f, 0.5f, -1.0f, 1.0f);

lightColors = XMFLOAT4(0.5f, 0.5f, 0.5f, 1.0f);

ConstantBuffer cb1;

cb1.mWorld = XMMatrixTranspose(world); //Загрузка матриц

cb1.mView = XMMatrixTranspose(view);

cb1.mProjection = XMMatrixTranspose(projection);

cb1.vLightDir = lightDirs;

cb1.vLightColor = lightColors;

cb1.vOutputColor = lightColors;

immediateContext->UpdateSubresource(constantBuffer, 0, NULL, &cb1, 0, 0);

// Рисовка куба

immediateContext->VSSetShader(vertexShader, NULL, 0);

immediateContext->VSSetConstantBuffers(0, 1, &constantBuffer);

immediateContext->PSSetShader(pixelShader, NULL, 0);

immediateContext->DrawIndexed(36, 0, 0);

immediateContext->PSSetConstantBuffers(0, 1, &constantBuffer);

immediateContext->PSSetShader(pixelShaderSolid, NULL, 0);

SetMatrixes(\_x, \_y, \_z);

swapChain->Present(0, 0);

}

HRESULT InitMatrixes()

{

RECT rc;

GetClientRect(hWnd, &rc);

UINT width = rc.right - rc.left;

UINT height = rc.bottom - rc.top;

// Инициализация матрицы мира

world = XMMatrixIdentity();

// Инициализация матрицы вида

XMVECTOR Eye = XMVectorSet(0.0f, 4.0f, -10.0f, 0.0f);

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

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

view = XMMatrixLookAtLH(Eye, At, Up);

// Инициализация матрицы проекции

projection = XMMatrixPerspectiveFovLH(XM\_PIDIV4, width / (FLOAT)height, 0.01f, 100.0f);

return S\_OK;

}

void CleanupDevice()

{

if (immediateContext) immediateContext->ClearState();

if (constantBuffer) constantBuffer->Release();

if (vertexBuffer) vertexBuffer->Release();

if (indexBuffer) indexBuffer->Release();

if (vertexLayout) vertexLayout->Release();

if (vertexShader) vertexShader->Release();

if (pixelShaderSolid) pixelShaderSolid->Release();

if (pixelShader) pixelShader->Release();

if (depthStencil) depthStencil->Release();

if (depthStencilView) depthStencilView->Release();

if (renderTargetView) renderTargetView->Release();

if (swapChain) swapChain->Release();

if (immediateContext) immediateContext->Release();

if (device) device->Release();

}

HRESULT InitGeometry()

{

HRESULT hr = S\_OK;

ID3DBlob\* pVSBlob = NULL;

hr = D3DCompileFromFile(L"F:\\графика\\L3\\shaders.fx", NULL, NULL, "VS", "vs\_5\_0",

D3DCOMPILE\_DEBUG, D3DCOMPILE\_ENABLE\_STRICTNESS, &pVSBlob, 0);

/\*////////////////////////////

if (FAILED(hr))

{

MessageBox(NULL, L"Невозможно скомпилировать файл FX. Пожалуйста, запустите данную программу из папки, содержащей файл FX.", L"Ошибка", MB\_OK);

return hr;

}\*/

hr = device->CreateVertexShader(pVSBlob->GetBufferPointer(), pVSBlob->GetBufferSize(), NULL, &vertexShader);

if (FAILED(hr))

{

pVSBlob->Release();

return hr;

}

D3D11\_INPUT\_ELEMENT\_DESC layout[] =

{

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

{ "NORMAL", 0, DXGI\_FORMAT\_R32G32B32\_FLOAT, 0, 12, D3D11\_INPUT\_PER\_VERTEX\_DATA, 0 },

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

};

UINT numElements = ARRAYSIZE(layout);

hr = device->CreateInputLayout(layout, numElements, pVSBlob->GetBufferPointer(),

pVSBlob->GetBufferSize(), &vertexLayout);

pVSBlob->Release();

if (FAILED(hr)) return hr;

immediateContext->IASetInputLayout(vertexLayout);

ID3DBlob\* pPSBlob = NULL;

hr = D3DCompileFromFile(L"F:\\графика\\L3\\shaders.fx", NULL, NULL, "PS", "ps\_5\_0",

D3DCOMPILE\_DEBUG, D3DCOMPILE\_ENABLE\_STRICTNESS, &pPSBlob, 0);

/\*//////////////////////

if (FAILED(hr))

{

MessageBox(NULL, L"Невозможно скомпилировать файл FX. Пожалуйста, запустите данную программу из папки, содержащей файл FX.", L"Ошибка", MB\_OK);

return hr;

}\*/

hr = device->CreatePixelShader(pPSBlob->GetBufferPointer(), pPSBlob->GetBufferSize(), NULL, &pixelShader);

pPSBlob->Release();

/////////if (FAILED(hr)) return hr;

pPSBlob = NULL;

hr = D3DCompileFromFile(L"F:\\графика\\L3\\shaders.fx", NULL, NULL, "PSSolid", "ps\_5\_0",

D3DCOMPILE\_DEBUG, D3DCOMPILE\_ENABLE\_STRICTNESS, &pPSBlob, 0);

/\*////////////////////

if (FAILED(hr))

{

MessageBox(NULL, L"Невозможно скомпилировать файл FX. Пожалуйста, запустите данную программу из папки, содержащей файл FX.", L"Ошибка", MB\_OK);

return hr;

}\*/

hr = device->CreatePixelShader(pPSBlob->GetBufferPointer(), pPSBlob->GetBufferSize(), NULL, &pixelShaderSolid);

pPSBlob->Release();

///////////////if (FAILED(hr)) return hr;

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) },

};

D3D11\_BUFFER\_DESC bd;

ZeroMemory(&bd, sizeof(bd));

bd.Usage = D3D11\_USAGE\_DEFAULT;

bd.ByteWidth = sizeof(SimpleVertex) \* 24;

bd.BindFlags = D3D11\_BIND\_VERTEX\_BUFFER;

bd.CPUAccessFlags = 0;

D3D11\_SUBRESOURCE\_DATA InitData;

ZeroMemory(&InitData, sizeof(InitData));

InitData.pSysMem = vertices;

hr = device->CreateBuffer(&bd, &InitData, &vertexBuffer);

if (FAILED(hr)) return hr;

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

};

bd.Usage = D3D11\_USAGE\_DEFAULT;

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

bd.BindFlags = D3D11\_BIND\_INDEX\_BUFFER;

bd.CPUAccessFlags = 0;

InitData.pSysMem = indices;

hr = device->CreateBuffer(&bd, &InitData, &indexBuffer);

if (FAILED(hr)) return hr;

UINT stride = sizeof(SimpleVertex);

UINT offset = 0;

immediateContext->IASetVertexBuffers(0, 1, &vertexBuffer, &stride, &offset);

immediateContext->IASetIndexBuffer(indexBuffer, DXGI\_FORMAT\_R16\_UINT, 0);

immediateContext->IASetPrimitiveTopology(D3D11\_PRIMITIVE\_TOPOLOGY\_TRIANGLELIST);

bd.Usage = D3D11\_USAGE\_DEFAULT;

bd.ByteWidth = sizeof(ConstantBuffer);

bd.BindFlags = D3D11\_BIND\_CONSTANT\_BUFFER;

bd.CPUAccessFlags = 0;

hr = device->CreateBuffer(&bd, NULL, &constantBuffer);

if (FAILED(hr)) return hr;

return S\_OK;

}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)

{

WNDCLASSEX win;

win.cbSize = sizeof(WNDCLASSEX);

win.style = CS\_HREDRAW | CS\_VREDRAW;

win.lpfnWndProc = WndProc;

win.cbClsExtra = 0;

win.cbWndExtra = 0;

win.hInstance = hInstance;

win.hIcon = LoadIcon(hInstance, (LPCTSTR)IDI\_WINLOGO);

win.hCursor = LoadCursor(NULL, IDC\_ARROW);

win.hbrBackground = (HBRUSH)(COLOR\_WINDOW + 1);//(HBRUSH)GetStockObject(WHITE\_BRUSH);

win.lpszMenuName = 0;

win.lpszClassName = szClassName;

win.hIconSm = LoadIcon(NULL, IDI\_APPLICATION);

if (!RegisterClassEx(&win))

{

MessageBox(NULL,

\_T("Cannot register class!"),

\_T("F"),

NULL);

return 1;

}

hWnd = CreateWindow(

szClassName,

szTitle,

WS\_OVERLAPPEDWINDOW,

128, 64, 516, 539, 0,

(HMENU)NULL,

hInstance,

NULL);

MSG msg;

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

InitDevice();

InitGeometry();

InitMatrixes();

while (GetMessage(&msg, NULL, 0, 0) > 0)

{

TranslateMessage(&msg);

DispatchMessage(&msg);

Render();

}

return 0;

CleanupDevice();

}

**Вывод:** в ходе выполнения лабораторной работы было произведено ознакомление с DirectX directxmath, а также было реализовано освещение по модели Блинна-Фонга на пиксельном шейдере.