МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

МЕТОДИЧНІ ВКАЗІВКИ

до виконання лабораторних робіт

«Комп'ютерна графіка. Частина 1»

з курсу «Комп'ютерна графіка»
для студентів спеціальностей
151 «Автоматизація та комп'ютерно-інтегровані технології»
172 «Телекомунікація та радіотехніка»

Рекомендовано
редакційно-видавничою
радою університету,
протокол №1 від 19.02.2020р

Харків НТУ «ХПІ» 2020

Методичні вказівки до виконання лабораторних робіт «Комп'ютерна графіка. Частина 1» з курсу «Комп'ютерна графіка» для студентів спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології», 172 «Телекомунікація та радіотехніка» денної та заочної форм навчання / уклад. А. О. Зуєв, О. М. Євсеєнко, В.А. Крилова. – Харків : НТУ «ХПІ». – 47 с.

Укладачі А. О. Зуєв
О. М. Євсеєнко
В.А. Крилова

Рецензент І.В. Григоренко

Кафедра інформаційно-вимірювальних технологій і систем

ЗМІСТ

ВСТУП

Подання даних на моніторі комп'ютера в графічному вигляді вперше було реалізовано в середині 50-х років для великих ЕОМ, які застосовувалися в наукових і військових дослідженнях. З тих пір графічний спосіб відображення даних став невід'ємною частиною комп'ютерних систем.

Комп'ютерна графіка – це розділ інформатики, який вивчає методи створення та обробки графічних зображень за допомогою обчислювальної техніки.

Комп'ютерна графіка використовується майже в усіх наукових та інженерних дисциплінах для збільшення наочності та поліпшення сприйняття інформації людиною. Без комп'ютерної графіки неможливо уявити собі не тільки комп'ютерний, але і звичайний, цілком матеріальний світ. Незважаючи на те, що комп'ютерна графіка є, по суті, інструментом, її структура і методи засновані на передових досягненнях фундаментальних і прикладних наук: математики, фізики, хімії, біології, статистики, програмування тощо. Це дійсно як для програмних, так і для апаратних засобів створення й обробки зображень на комп'ютері.

Тому комп'ютерна графіка як галузь інформатики активно розвивається і багато в чому виступає рушієм усієї комп'ютерної індустрії.

Найважливішою частиною комп'ютерної графіки є візуалізація – загальна назва прийомів подання інформації або явищ у вигляді, зручному для зорового спостереження та аналізу. Візуалізація даних застосовується в різних сферах людської діяльності: у медицині (комп'ютерна томографія), військовій справі (тренажерні комплекси), наукових дослідженнях (візуалізація будови речовини, векторних полів та інших даних), моделюванні тканин та одягу, дослідно-конструкторських розробках, індустрії розваг.

Методичні вказівки призначені для вивчення основ комп'ютерної графіки та її застосування у створенні систем візуалізації. Розглянуто питання кодування кольору, растеризації, подання та візуалізації тривимірних об'єктів, особливості програмно-апаратних систем тривимірної графіки.

Методичні вказівки містять також індивідуальні завдання для виконання лабораторних і практичних робіт, які допоможуть в ефективному освоєнні курсу комп'ютерної графіки.

ЛАБОРАТОРНА РОБОТА № 1
СТВОРЕННЯ НАЙПРОСТІШОГО ВІКОННОГО WIN32-ДОДАТКА

Мета роботи: Навчитися використовувати Win32 API для створення віконних додатків в операційній системі *Windows* в середовищі *Microsoft Visual*.

Необхідне обладнання та комплектуючі, ПЗ: Персональний комп'ютер з встановленою операційною системою (ОС) *Windows*, оболонка *Microsoft Visual* C++.

Загальні відомості

Для розробки додатків ОС *Windows* пропонує програмісту універсальний механізм доступу до функцій програмного інтерфейсу (*Application Programming Interface*). Цей механізм є прошарком між користувацькою програмою і власне операційною системою. З одного боку такий підхід забезпечує стандартизований доступ до функцій та інструментів операційної системи, а з іншого – забезпечується необхідний рівень безпеки операційної системи. Крім того, ОС Windows є багатозадачною операційною системою. Для взаємодії додатків реалізовано механізм на основі подій – повідомлень. З точки зору програми це повідомлення про те, що відбулася якась подія, яка може вимагати, а може і не вимагати виконання певних дій. Ця подія може бути наслідком дій користувача, наприклад, переміщення курсору або клацання кнопкою миші, зміни розмірів вікна або вибору пункту меню. Крім того, подія може генеруватися додатком, а також операційною системою. Повідомлення – це структура даних, яка містить наступні елементи:

- Дескриптор (описувач) вікна, якому адресоване повідомлення;
- Код (номер) повідомлення;
- Додаткову інформацію залежно від коду повідомлення.

З моменту старту програми операційна система створює в пам'яті глобальний об'єкт, названий системною чергою повідомлень. Усі повідомлення, що генеруються як апаратурою, так і додатками, розміщуються в цій черзі. ОС періодично опитує цю чергу і, якщо вона не порожня, посилає наступне повідомлення потрібному адресату, який визначається за допомогою дескриптора вікна.

Точкою входу в віконний Win32-додаток, створений на мові C++, є функція *WinMain()* (_tWinMain). Для роботи з Win32 API необхідно

підключити заголовковий файл *#include <windows.h>*.

Для обробки додатком повідомлень, що надходять до нього, використовується віконна процедура, яка зазвичай має заголовок зі стандартним синтаксисом:

LRESULT CALLBACK ім'я_функції(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)

У цьому визначенні *LRESULT* – тип значення, що повертається, *hWnd* – дескриптор вікна, якому адресоване повідомлення, *uMsg* – код повідомлення, *wParam* і *lParam* – параметри повідомлення. Ім'я функції може бути довільним, але для головного вікна програми зазвичай використовується ім'я *WndProc*.

Неодмінним компонентом усіх Windows-додатків є також цикл обробки повідомлень. Функція *WinMain* зазвичай містить виклики функцій для ініціалізації і створення вікон (*InitInstance, RegisterClass*), після чого починається цикл обробки повідомлень і надходить необхідний код для закриття додатків. Вилучає повідомлення з черги функція *PeekMessage*. Якщо чергове повідомлення має код WM_QUIT, то відбувається вихід з циклу, після чого додаток завершує свою роботу. Якщо чергове повідомлення не є повідомленням WM_QUIT, то воно передається функції *DispatchMessage*, яка повертає повідомлення назад в ОС Windows. Windows відправляє повідомлення для його обробки відповідній віконній процедурі – іншими словами, Windows викликає віконну процедуру. Після повернення з віконної процедури Windows передає управління оператору, який розташований після *DispatchMessage*, і робота циклу триває.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual* C++ *Express Edition*.
2. Створити новий (порожній) проект (див. додаток А).
3. У папці «*Source Files*» створити головний файл додатку *kg_lab.cpp*.
4. У папці «*Header Files*» створити заголовочні файли додатку *kg_lab.h*, *stdafx.h*.
5. У файлі *stdafx.h* підключити необхідні бібліотеки.
6. У головному файлі додатку створити необхідні функції, помістивши їх у простір імен *main* і додати їх опис до заголовкового файлу

kg_lab.h.

7. Створити в кінці файлу *kg_lab.cp*р функцію обробки повідомлень *WndProc,* яка обробляє повідомлення WM_DESTROY і WM_ACTIVATEAPP.

8. Налагодити і запустити програму. Після запуску програма повинна виводити порожнє вікно додатка з назвою "Комп'ютерна графіка. Лабораторна робота" і закриватися після натискання комбінації клавіш Alt + F4 або хрестика в системному меню.

9. Підготувати звіт про лабораторну роботу. Звіт повинен містити завдання і текст програми (файли *kg_lab.h* і *kg_lab.cpp*).

Вихідний код роботи

Файл stdafx.h

```
#pragma once

#ifndef WINVER
#define WINVER 0x0501
#endif
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0501
#endif
#ifndef _WIN32_WINDOWS
#define _WIN32_WINDOWS 0x0410
#endif
#ifndef _WIN32_IE
#define _WIN32_IE 0x0600
#endif
#define WIN32_LEAN_AND_MEAN

#include <windows.h>
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
#include <objbase.h>
```

Файл kg_lab.h

```
#pragma once
#pragma warning (disable 4100)
```

```
namespace main
{
extern HWND hWnd;
extern bool boActive;

void message_box_( const TCHAR* mes );

ATOM RegisterClass( HINSTANCE hInstance );
bool InitInstance( HINSTANCE, int );
LRESULT CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );

}; // namespace main
```

Файл kg_lab.cpp:

```
#include "stdafx.h"
#include "kg_lab.h"

namespace main
{

HINSTANCE hInst;
bool boActive = false;
HWND hWnd = 0;
const TCHAR* szClass = L"KG Lab Class";
const TCHAR* szTitle = L"Комп'ютерна графіка. Лабораторна робота.";

void message_box_( const TCHAR* mes )
{
::MessageBox( 0, mes, 0, MB_OK | MB_ICONERROR | MB_TASKMODAL );
}

ATOM RegisterClass( HINSTANCE hInstance )
{
WNDCLASSEX wcex;
wcex.cbSize = sizeof (WNDCLASSEX);

wcex.style          = CS_HREDRAW | CS_VREDRAW;
wcex.lpfnWndProc    = Main :: WndProc;
wcex.cbClsExtra     = 0;
wcex.cbWndExtra     = 0;
wcex.hInstance      = hInstance;
```

```
wcex.hIcon          = 0;
wcex.hCursor        = LoadCursor( 0, IDC_ARROW );
wcex.hbrBackground  = (HBRUSH) (COLOR_WINDOW + 1);
wcex.lpszMenuName = 0;
wcex.lpszClassName  = szClass;
wcex.hIconSm        = 0;
return RegisterClassEx( &wcex );
}

bool InitInstance ( HINSTANCE hInstance, int nCmdShow )
{
  hInst = hInstance;
  DWORD wid = GetSystemMetrics(SM_CXSCREEN);
  DWORD hei = GetSystemMetrics(SM_CYSCREEN);

 hWnd = CreateWindow( szClass, szTitle, WS_CAPTION | WS_VISIBLE |
        WS_SYSMENU | WS_POPUP, 0, 0, wid, hei, 0, 0, hInstance, 0 );
  if ( !hWnd )
  {
        message_box_( L"Неможливо створити вікно додатка!" );
        return false;
  } // if ( !hWnd )

  ShowWindow( hWnd, nCmdShow );
  UpdateWindow ( hWnd );
  return true;
}
LRESULT CALLBACK WndProc( HWND hWnd, UINT Message,
        WPARAM wParam, LPARAM lParam )
{
switch( Message )
{
case WM_ACTIVATEAPP:
        {
                boActive = wParam == 1;
                break;
        }
case WM_DESTROY:
        {
                PostQuitMessage(0);
                break;
        }
```

```
default:
	return DefWindowProc( hWnd, Message, wParam, lParam );
}
return 0;
}
}; // namespace main

int APIENTRY _tWinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
		LPTSTR lpCmdLine, int nCmdShow )
{
CoInitializeEx( 0, COINIT_MULTITHREADED );
main::RegisterClass( hInstance );
if( !main::InitInstance( hInstance, nCmdShow ) )
	return -1;

MSG msg;
do
{
	if( !PeekMessage( &msg, 0, 0, 0, PM_REMOVE ) )
	{
		continue;
	} // if( !PeekMessage( &msg, 0, 0, 0, PM_REMOVE ) )

	TranslateMessage( &msg );
	DispatchMessage( &msg );

}
while( WM_QUIT != msg.message );

return (int)msg.wParam;
}
```

Контрольні питання

1. Що таке Win32 API?
2. Призначення Win32 API.
3. На основі чого реалізований механізм взаємодії додатків?
4. Що таке повідомлення?
5. Коли виникає повідомлення?
6. Структура повідомлення.
7. Що називається системною чергою повідомлень?
8. Що є точкою входу в віконний Win32-додаток?
9. Призначення віконної процедури.
10. Що таке цикл обробки повідомлень?
11. Що являють собою функції *InitInstance*, *RegisterClass*?
12. Що станеться, якщо повідомлення має код WM_QUIT?
13. Що робить функція *DispatchMessage*?

ЛАБОРАТОРНА РОБОТА № 2
ПІДКЛЮЧЕННЯ БІБЛІОТЕКИ DIRECTX ДО WIN32-ДОДАТКА

Мета роботи: Ознайомитися з особливостями підключення і використання графічної бібліотеки Direct3D в додатках Win32

Необхідне обладнання та комплектуючі, ПЗ: встановлений пакет DirectX SDK, лабораторна робота № 1.

Загальні відомості

DirectX і його компонента Direct3D – це низькорівневий графічний API (програмний інтерфейс для додатків), який дозволяє відображати тривимірні сцени, використовуючи апаратні прискорювачі тривимірної графіки. Direct3D можна сприймати як посередника між додатком і графічним пристроєм (апаратурою тривимірної графіки).

Direct3D – це набір документованих інтерфейсів і функцій, які охоплюють повний набір функціональних можливостей з управління процесом візуалізації тривимірних сцен.

Для розробки додатків з використанням функцій Direct3D необхідно виконати підключення бібліотеки Direct3D. Бібліотека Direct3D основана на застосуванні т.зв. COM інтерфейсів. Модель компонентних об'єктів (Component Object Model, COM) – це технологія, яка дозволяє бібліотеці бути незалежною від мови програмування і сумісною з усіма попередніми версіями. Для отримання покажчика на COM-інтерфейс необхідно викликати спеціальну функцію або метод іншого COM-інтерфейсу (зазвичай назва такої функції починається зі слова *Create*). Крім того, завершивши роботу з COM-інтерфейсом, слід викликати його метод *Release*. COM-об'єкти самостійно здійснюють управління пам'яттю. У коді для позначення COM-інтерфейсів використовується велика літера I. Наприклад, COM-інтерфейс, який являє собою пристрій візуалізації (відеоадаптер), називається IDirect3DDevice9.

Для ініціалізації бібліотеки необхідно виконати наступні дії:

- Запросити інтерфейс IDirect3D9. Він застосовується для отримання інформації про встановлені в комп'ютері пристрої візуалізації і для створення інтерфейсу IDirect3DDevice9, який являє собою апаратний пристрій, що використовується для виведення тривимірної графіки.
- Ініціалізувати екземпляр структури

D3DPRESENT_PARAMETERS. Ця структура містить ряд параметрів, що дозволяють задати характеристики виводу для пристрою, який буде використовуватися для візуалізації.

- Створити об'єкт IDirect3DDevice9, грунтуючись на ініціалізованій структурі D3DPRESENT_PARAMETERS.

Додаток, який працює з графічною бібліотекою Direct3D, має виконати три основні етапи: ініціалізація, розрахунок і малювання сцени, звільнення графічних ресурсів перед завершенням додатку.

У додатку набір перелічених функцій доцільно реалізувати в окремих *.cpp* файлів.

Для компіляції додатку з використанням бібліотеки Direct3D необхідно виконати ряд налаштувань середовища розробки і самого проекту. Так, у середовищі розробки необхідно встановити шляхи до заголовкових файлів і бібліотек DirectX SDK (див. додаток А). У самому проекті необхідно вказати бібліотеку *d3d9.lib*. Функції для роботи з Direct3D необхідно помістити в простір імен directx.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual C++ Express Edition*.
2. Відкрити проект лабораторної роботи № 1.
3. У папці «*Source Files*» створити файл для роботи з бібліотекою Direct3D *directx.cpp* (див. додаток Б).
4. У середовищі розробки встановити шляхи до заголовкових файлів і бібліотек DirectX SDK.
5. Підключити бібліотеки "*d3d9.lib*" і "*d3dx9.lib*" в заголовковому файлі *kg_lab.h*.
6. Набрати текст програми у файлі *directx.cpp* і внести зміни в інші файли проекту.
7. Додати опис функцій у файлі *kg_lab.h*.
8. *Увага!* У функції *Paint*() (у файлі *directx.cpp*) у виклику функції *Clear* замість усіх знаків #### повинен бути заданий код кольору в 16-річній системі відповідно до варіанту
9. Налагодити і запустити програму. Програма повинна виводити порожнє вікно додатка, забарвлене в заданий відповідно до варіанту колір.
10. Підготувати звіт про лабораторну роботу. Звіт повинен містити

завдання, опис і призначення основних використаних функцій бібліотеки Direct3D і текст програми (*directx.cpp*).

Вихідний код роботи

Додати до файлу stdafx.h:

```
#include <memory.h>
#include <tchar.h>
#include <objbase.h>
#include <d3d9.h>
#include <d3dx9.h>
```

Додати до файлу kg_lab.h:

```
#pragma once
#pragma warning(disable:4100)
#pragma comment(lib, "d3d9.lib")
#pragma comment(lib, "d3dx9.lib")
namespace main
{
extern HWND hWnd;
extern bool boActive;
...
LRESULT CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );

}; // namespace main
namespace directx
{
void FillParams();
bool InitDX();
void CloseDX();
void FlipPages();
void Invalidate();
void Restore();
bool TestAvailDev();
void Paint();
void Pump();

typedef D3DXVECTOR3 s_vector3;
typedef D3DXVECTOR4 s_vector4;
typedef D3DXMATRIX s_matrix;
typedef WORD        t_index;
```

```
const float deg2rad = 3.14f / 180.f;
const s_vector3 vec0 = s_vector3( 0.f, 0.f, 0.f );
const s_vector3 vec1 = s_vector3( 1.f, 1.f, 1.f );

struct s_vertex
{
        s_vertex() {}
        s_vertex( const s_vector3& Pos_, const s_vector3& Norm_ = vec0 ):
        Pos(Pos_), Norm(Norm_) {}
        s_vector3       Pos;
        s_vector3       Norm;
};
extern D3DPRESENT_PARAMETERS PP;
}; // namespace directx
```

Додати до файлу kg_lab.cpp (у функцію InitInstance):

```
...
  if( !hWnd )
  {
        message_box_( L"Неможливо створити вікно додатка!" );
        return false;
  } // if( !hWnd )

  if( !directx::InitDX() )
  {
          PostMessage( hWnd, WM_CLOSE, 0, 0 );
          return false;
  } // if( !directx::InitDX() )
...
```

Додати до файлу kg_lab.cpp (у функцію WndProc):

```
LRESULT CALLBACK WndProc( HWND hWnd, UINT Message,
        WPARAM wParam, LPARAM lParam )
{
switch( Message )
{
case WM_ERASEBKGND: return 1;
case WM_ACTIVATEAPP:
        {
                boActive = wParam == 1;
```

```
                directx::TestAvailDev();
                break;
        }
case WM_PAINT:
        {
                PAINTSTRUCT ps;
                BeginPaint( hWnd, &ps );
                directx::Paint();
                EndPaint( hWnd, &ps );
                break;
        }
case WM_DESTROY:
        {
        ...
```

Додати до файлу kg_lab.cpp (у функцію_tWinMain):

```
MSG msg;
do
{
        if ( !PeekMessage( &msg, 0, 0, 0, PM_REMOVE ) )
        {
                directx::Pump();
                continue;
        } // if ( !PeekMessage( &msg, 0, 0, 0, PM_REMOVE ) )
        TranslateMessage( &msg );
        DispatchMessage( &msg );
}
while( WM_QUIT != msg.message );
directx::CloseDX();
        ...
```

До файлу directx.cpp:

```
#include "stdafx.h"
#include "kg_lab.h"
```

```
namespace directx
{
IDirect3D9*    pD3D = 0;
IDirect3DDevice9* pD3DDev = 0;
bool boInvalid = false;
D3DPRESENT_PARAMETERS PP = {0};

void FillParams()
{
RECT rc = {0};
GetClientRect( main::hWnd, &rc );

PP.BackBufferWidth = rc.right - rc.left;
PP.BackBufferHeight = rc.bottom - rc.top;
PP.BackBufferFormat = D3DFMT_X8R8G8B8;
PP.BackBufferCount = 0;
PP.MultiSampleType = D3DMULTISAMPLE_NONE;
PP.MultiSampleQuality = 0;
PP.SwapEffect = D3DSWAPEFFECT_DISCARD;
PP.hDeviceWindow = main::hWnd;
PP.Windowed = true;
PP.EnableAutoDepthStencil = true;
PP.AutoDepthStencilFormat = D3DFMT_D24S8;
PP.Flags = D3DPRESENTFLAG_DISCARD_DEPTHSTENCIL;
PP.FullScreen_RefreshRateInHz = 0;
PP.PresentationInterval = D3DPRESENT_INTERVAL_DEFAULT;
}
bool InitDX()
{
pD3D = Direct3DCreate9( D3D_SDK_VERSION );
if( !pD3D )
  return false;
FillParams();
pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL,
        main::hWnd, D3DCREATE_HARDWARE_VERTEXPROCESSING,
        &PP, &pD3DDev );
if( !pD3DDev )
{
        pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_REF,
                main::hWnd,   D3DCREATE_SOFTWARE_VERTEXPROCESSING,
                &PP, &pD3DDev );
        if( !pD3DDev )
```

```
		{
			main::message_box_( L"Неможливо ініціалізувати DirectX!" );
			return false;
		} // if( !pD3DDev )
} // if ( !pD3DDev )
return true;
}
void CloseDX()
{
if( pD3DDev )
{
		pD3DDev->Release();
		pD3DDev = 0;
} // if ( pD3DDev )
if( pD3D )
{
		pD3D->Release();
		pD3D = 0;
} // if( pD3D )
CoUninitialize();
}
void FlipPages()
{
		pD3DDev->Present( 0, 0, 0, 0 );
}
void Invalidate()
{
		if( !pD3DDev || boInvalid )
			return;
		pD3DDev->EvictManagedResources();
		boInvalid = true;
}
void Restore()
{
		if( !pD3DDev || !boInvalid )
			return;
		boInvalid = pD3DDev->Reset( &PP ) != S_OK;
}
bool TestAvailDev()
{
if( !pD3DDev )
		return false;
```

```
switch( pD3DDev->TestCooperativeLevel() )
{
case D3DERR_DEVICELOST: { Invalidate(); break; }
case D3DERR_DEVICENOTRESET: { Restore(); break; }
} // switch( pD3DDev->TestCooperativeLevel() )
return !boInvalid;
}
void Paint()
{
if( !pD3DDev )
        return;
if( !TestAvailDev() || pD3DDev->BeginScene() != D3D_OK )
        return;
pD3DDev->Clear( 0, 0, D3DCLEAR_STENCIL | D3DCLEAR_TARGET |
        D3DCLEAR_ZBUFFER, #####, 1.f, 0 );
pD3DDev->EndScene();
FlipPages();
}
void Pump()
{
if( !main::boActive )
        Sleep(20);
Paint();
}
}; // namespace directx
```

Увага! У функції *Paint*() у виклику функції Clear замість знаків #### повинен бути заданий код кольору в 16-річній системі (див. додаток В).

Контрольні питання

1. Що таке DirectX і Direct3D?
2. Що називається API?
3. Що являє собою Direct3D?
4. Як підключити бібліотеку Direct3D?
5. Що таке COM-інтерфейси?
6. Як отримати покажчик на COM-інтерфейс?
7. Призначення методу *Release*?
8. Призначення COM-інтерфейсу*IDirect3DDevice9.*
9. Етапи ініціалізації бібліотеки.
10. Які етапи має виконати додаток при використанні Direct3D?

ЛАБОРАТОРНА РОБОТА № 3
НАЛАШТУВАННЯ ПРОЦЕСУ РЕНДЕРИНГУ

Мета роботи: Ознайомитися з особливостями організації процесу рендерингу за допомогою графічної бібліотеки Direct3D.

Необхідне обладнання та комплектуючі, ПЗ: встановлений пакет DirectX SDK, лабораторна робота № 2.

Загальні відомості

Для управління процесом рендерингу в бібліотеці Direct3D використовуються ефекти, описані у файлі з розширенням *fx*. На етапі запуску програми файл з ефектом завантажується і компілюється. У процесі візуалізації кожного кадру вибирається техніка рендерингу для кожного об'єкта, встановлюються параметри рендерингу (матриці, параметри матеріалів і джерела світла) і проводиться розрахунок зображення. Перед закриттям програми ефект звільняється.

Ефект складається з 4-х частин:

1) Змінні, константи і структури, які використовуються в процесі візуалізації. Для визначення матриці використовується тип *float*4x4, для визначення вектора з 4-х елементів – *float*4, для визначення числа з плаваючою комою – *float*.

2) Вершинна мікропрограма (вершинний шейдер) – мікропрограма для графічного процесора, яка описує, яким чином здійснюється обробка геометрії моделей, що візуалізуються.

3) Фрагментна мікропрограма (піксельний шейдер) – описує, яким чином здійснюється обробка кожного пікселя, що візуалізується.

4) Опис техніки рендерингу – містить настройки конвеєра рендерингу Direct3D, а також вершинну і піксельну мікропрограму, які використовуються для візуалізації.

Техніка рендерингу (*technique*) wire задає параметри відтворення:

– тип виведення графіки: каркасна модель (*FillMode = wireframe*);

– режим відсікання невидимих граней: проти годинникової стрілки (*CullMode = ccw*);

– режим сортування фрагментів за глибиною (*ZEnable = true, ZFunc = LessEqual*);

– режим комбінування фрагментів;

– вершинну мікропрограму;

– фрагментну мікропрограму.

У проекті необхідно додати два файли: *render.cpp*, в якому буде міститися робота з ефектами, і *shader.fx*, в якому буде міститися текст мікропрограм і техніки рендерингу.

Функції для роботи з ефектом необхідно помістити в простір імен render.

Хід виконання

1. Запустити середовище розробки *Microsoft Visual* C++ *Express Edition*.
2. Відкрити проект лабораторної роботи № 2.
3. У папці «*Source Files*» створити файл для організації рендерингу *render.cpp*.
4. У папці «*Source Files*» створити файл опису технік рендерингу і мікропрограм *shader.fx*.
5. Набрати текст програми у файлі render.cpp і внести зміни до інших файлів проекту згідно з вихідним кодом роботи.
6. Набрати текст ефекту у файлі *shader.fx*.
7. Додати опис функцій до файлу *kg_lab.h*.
8. Налагодити і запустити програму, перевірити у відладчику і записати порядок виконання функцій: *TestAvailDev*(), *LoadResources*(), *OnExit*(), *RenderScene*(), *FillParams*(), *InitDX*(), *CloseDX*(), *FlipPages*(), *Paint*(), *Pump*(), *RegisterClass*(), *InitInstance*().

Для цього встановити *на початку функцій* (на першому рядку функції) точки переривання (F9) і запустити додаток у режимі налагодження (F5), після зупинки на точці переривання необхідно записати назву функції у звіт і продовжити виконання програми (F5). Якщо якісь функції виконуються циклічно, необхідно записати послідовність 3 рази, а потім завершити програму.

9. Підготувати звіт про лабораторну роботу. Звіт повинен містити завдання, порядок виконання функцій з п.п.№8 і текст програми (*render.cpp, shader.fx*).

Вихідний код роботи
Додати до файлу kg_lab.h:

```
...
extern D3DPRESENT_PARAMETERS PP;

}; // namespace directx

namespace render
{
enum
{
        RM_WIRE,
        RM_MAX,
};

const int CurRenderMethod = RM_WIRE;
const DWORD BackColor = 0xff000000UL;

bool LoadResources( IDirect3DDevice9* Dev );
void RenderScene( IDirect3DDevice9* Dev, int TN );

void OnLostDevice( IDirect3DDevice9* Dev );
void OnRestoreDevice( IDirect3DDevice9* Dev );
void OnExit( IDirect3DDevice9* Dev );

extern D3DXHANDLE hMatWorld;
extern D3DXHANDLE hObjColor;

extern D3DXHANDLE hPower;
extern D3DXHANDLE hSpecLevel;

extern D3DXHANDLE hLigColor;
extern D3DXHANDLE hAmbColor;
extern D3DXHANDLE hLigDir;
extern D3DXHANDLE hLigPos;

extern D3DXHANDLE hMatViewProj;
extern D3DXHANDLE hMatIView;
}; // namespace render
```

Додати до файлу shader.fx:

```
//параметри камери
float4x4 m_view_proj;
float4x4 m_iview;
//джерело світла
float4 v_lig_dir;
float4 v_lig_pos;
float4 v_lig_color, v_amb_color;
//параметри об'єкта
float4x4 m_world;
float4 v_obj_color;
float f_power, f_spec_level;

technique wire
{
   pass P0
   {
         FillMode          = Wireframe;
         FogEnable         = false;
         AlphaBlendEnable  = false;
         AlphaTestEnable   = false;
         ZEnable           = true;
         ZFunc             = LessEqual;
         CullMode          = CCW;
  }
}
```

Додати до файлу render.cpp:

```
#include "stdafx.h"
#include "kg_lab.h"

namespace render
{
IDirect3DVertexDeclaration9* pDecl = 0;

LPD3DXEFFECT pEffect = 0;
D3DXHANDLE hTechnique[RM_MAX] = {0};
D3DXHANDLE hMatViewProj = 0;
D3DXHANDLE hMatIView = 0;
D3DXHANDLE hMatWorld = 0;
D3DXHANDLE hObjColor = 0;
```

```
D3DXHANDLE hPower = 0;
D3DXHANDLE hSpecLevel = 0;
D3DXHANDLE hLigColor = 0;
D3DXHANDLE hAmbColor = 0;
D3DXHANDLE hLigDir = 0;
D3DXHANDLE hLigPos = 0;

void OnLostDevice( IDirect3DDevice9* Dev )
{
	if( pEffect )
		pEffect->OnLostDevice();
}

void OnRestoreDevice( IDirect3DDevice9* Dev )
{
	if( pEffect )
		pEffect->OnResetDevice();
}

void OnExit( IDirect3DDevice9* Dev )
{
if( pEffect )
{
	pEffect->Release();
	pEffect = 0;
} // if( pEffect )

if( pDecl )
{
	pDecl->Release();
	pDecl = 0;
} // if (pDecl)
}

bool LoadResources( IDirect3DDevice9* Dev )
{
LPD3DXBUFFER errors = 0;
D3DXCreateEffectFromFile( Dev, L"shader.fx", 0, 0, 0, 0, &pEffect, &errors );

if( errors )
{
	const char* e = (const char*) errors->GetBufferPointer();
```

```
        TCHAR buf [512];
        ::MultiByteToWideChar( CP_ACP, 0, e, -1, buf, sizeof(buf) / sizeof(buf [0]) );
        main::message_box_( buf );
        errors->Release();
        return false;
} // if( errors )
if( !pEffect )
{
        main::message_box_( L"Шейдер не найден!" );
        return false;
} // if( !pEffect )
static const char* tnames[RM_MAX] = { "wire" };
for( int i = 0; i < RM_MAX; ++i )
{
        hTechnique[i] = pEffect->GetTechniqueByName( tnames[i] );
        if( !hTechnique[i] )
        {
                main::message_box_( L"Відповідний метод рендеру не найден!" );
                return false;
        } // if( !hTechnique[i] )
} // for( int i = 0; i < RM_MAX; ++i )

hMatViewProj = pEffect->GetParameterByName( 0, "m_view_proj" );
hMatIView = pEffect->GetParameterByName( 0, "m_iview" );
hMatWorld = pEffect->GetParameterByName( 0, "m_world" );
hObjColor = pEffect->GetParameterByName( 0, "v_obj_color" );
hPower = pEffect->GetParameterByName( 0, "f_power" );
hSpecLevel = pEffect->GetParameterByName( 0, "f_spec_level" );

hLigColor = pEffect->GetParameterByName( 0, "v_lig_color" );
hAmbColor = pEffect->GetParameterByName( 0, "v_amb_color" );
hLigDir = pEffect->GetParameterByName( 0, "v_lig_dir" );
hLigPos = pEffect->GetParameterByName ( 0, "v_lig_pos" );

static const D3DVERTEXELEMENT9 dcl[] =
{
        {0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT,
                D3DDECLUSAGE_POSITION, 0},
        {0, 12, D3DDECLTYPE_FLOAT3,  D3DDECLMETHOD_DEFAULT,
                D3DDECLUSAGE_NORMAL, 0},
        D3DDECL_END ()
};
```

```
Dev->CreateVertexDeclaration( dcl, &pDecl );
if( !pDecl )
	return false;
return true;
}

void RenderScene( IDirect3DDevice9* Dev, int TN )
{
if ( !pEffect || !hTechnique || !pDecl )
	return;
pEffect->SetTechnique( hTechnique[TN] );
Dev->SetVertexDeclaration( pDecl );
UINT passes = 0;
if( pEffect->Begin( &passes, 0 ) != D3D_OK )
	return;
for( UINT i = 0; i < passes; ++i )
{
	if( pEffect->BeginPass( i ) != D3D_OK )
		break;
	pEffect->EndPass();
} // for( UINT i = 0; i < passes; ++i )
pEffect->End();
}
}; // namespace render
```

Додати до файлу directx.cpp:

```
...
bool InitDX()
{
...
} // if( !PD3DDev )

if( !render::LoadResources( pD3DDev ) )
	return false;
return true;
}

void CloseDX()
{
render::OnExit( pD3DDev );
if( pD3DDev )
{
```

```
        pD3DDev->Release();
...
void Invalidate()
{
if( !pD3DDev || boInvalid )
        return;
render::OnLostDevice( pD3DDev );
pD3DDev->EvictManagedResources();
boInvalid = true;
}

void Restore()
{
if( !pD3DDev || !boInvalid )
        return;
boInvalid = pD3DDev->Reset( &PP ) != S_OK;
if( !boInvalid )
        render::OnRestoreDevice( pD3DDev );
}
...
void Paint()
{
if( !pD3DDev )
        return;
if( !TestAvailDev() || pD3DDev->BeginScene() != D3D_OK )
        return;

pD3DDev->Clear( 0, 0, D3DCLEAR_STENCIL | D3DCLEAR_TARGET |
        D3DCLEAR_ZBUFFER, render::BackColor, 1.f, 0 );
render::RenderScene( pD3DDev, render::CurRenderMethod );
pD3DDev->EndScene();
FlipPages();
}
...
```

Зверніть увагу, що функція Paint() вже є в коді лабораторної роботи, і треба її замінити новою.

Контрольні питання

1. Яке розширення мають файли для управління процесом рендерингу?
2. З яких частин складається ефект?
3. Що таке вершинний шейдер?
4. Що таке піксельний шейдер?
5. Які параметри відтворення задає техніка рендерингу?
6. Що означає параметр відтворення: *FillMode* = *Wireframe*?
7. Що означає параметр відтворення: *CullMode* = *CCW*?
8. Що означає параметр відтворення: *ZEnable* = *true*, *ZFunc* = *LessEqual*?

ЛАБОРАТОРНА РОБОТА № 4
ВІЗУАЛІЗАЦІЯ КАРКАСНОЇ МОДЕЛІ КУБА ЗА ДОПОМОГОЮ МІКРОПРОГРАМИ

Мета роботи: Отримати зображення каркасної моделі куба за допомогою найпростішої мікропрограми.

Необхідне обладнання та комплектуючі, ПЗ: лабораторна робота № 3, мікропрограма для відеоадаптера з лабораторної роботи № 3.

Загальні відомості

Усі мікропрограми записуються мовою HLSL, подібною синтаксисом з мовою C / C++. Для задання матриці використовується тип *float*4x4, для задання вектора з 4-х елементів – *float*4, для задання числа з плаваючою комою – *float*. Для задання константи перед типом ставиться ключове слово *const*.

Для кожної мікропрограми необхідно задати вхідні та вихідні дані у вигляді структури. Для вершинної мікропрограми *vs_simple* вхідними даними буде опис вершини: позиція і нормаль у локальній системі координат (задаються в структурі *s_vertex*2), а вихідними – позиція в проекційній системі координат, нормаль, позиція спостерігача і позиція у світовій системі координат (задаються в структурі *s_output*2). Ця ж структура буде слугувати вхідними даними для фрагментної мікропрограми *ps_wire*. На виході фрагментної мікропрограми формується колір пікселя і його прозорість (вектор з 4-х елементів).

У проекті необхідно додати два файли: *scene.cpp* і *figures.cpp*, в яких буде міститися опис інтерфейсу геометричного об'єкта і опис куба у вигляді списку трійок індексів вершин, які задають трикутники, і координат вершин.

Для опису геометричної фігури використовується загальний інтерфейс *s_figure*, який дозволяє проводити конструювання фігури, зміну її параметрів і отримувати опис геометрії й топології фігури. Опис фігур поміщається в простір імен *figures*. Нова фігура створюється шляхом успадкування інтерфейсу *s_figure*. Наприклад, куб створюється наступним чином: *struct s_cube: public scene*::*s_figure*{}.

Для кожної фігури необхідно визначити дві функції: *verts*() – повертає список вершин фігури, та *inds*() – повертає топологію фігури

(трійки індексів описують трикутники). Усередині кожної з функцій відповідно міститься масив вершин у локальній системі координат і масив індексів.

При запуску програми у функції *InitScene()* створюється модель куба з заданими параметрами. Під час формування зображення у функції *UpdateSceneMatrices()* розраховуються і встановлюються матриці, які описують систему координат спостерігача: у прикладі камера розташована на початку координат світової системи і напрямок спостереження збігається з напрямком осі *Z*.

У функції *RenderFigure()* проводиться розрахунок матриці об'єкта, установлення її в мікропрограму і виведення геометрії (вершин і індексів) фігури на екран.

У функції *s_figure::matrix()* проводиться розрахунок матриці об'єкта, яка перетворює його вершини з локальної системи координат у світову. Для формування матриці масштабування використовується функція *D3DXMatrixScaling()*, для створення матриці обертання – *D3DXMatrixRotationYawPitchRoll()*, а для створення матриці зсуву – *D3DXMatrixTranslation()*. Отримані матриці послідовно перемножуються за допомогою функції *D3DXMatrixMultiply()*, у результаті чого виходить загальна матриця об'єкта, яка зберігається в полі структури *s_figure - world*.

У вершинній мікропрограмі vs_simple для перетворення матрицею об'єкта *m_world* його позиції *i.pos* в локальній системі координат використовується функція mul у такий спосіб: *float*4 *P=mul*(*i.pos, m_world*).

Для перетворення нормалі (повороту) використовується підматриця 3x3 матриці об'єкта *m_world* *o.nor = mul*(*i.nor,* (*float*3x3)*m_world*).

Для передачі значень матриць і векторів з основної програми в мікропрограму (ефект) використовуються функції *SetMatrix()* і *SetVector()*. Для відправки геометрії і топології об'єкта на візуалізацію використовується функція *DrawIndexedPrimitiveUP()*.

У фрагментній мікропрограмі *ps_wire* колір фрагмента формується із заданого кольору об'єкта *v_obj_color* з рівнем прозорості 1 (повністю непрозорий фрагмент). Колір у мікропрограмі задається у вигляді вектора з 4-х елементів: RGBA. Діапазон зміни кожного елемента – від 0 до 1, при цьому 1 відповідає значенню 255 (0xFF), яке використовується при заданні кольору в основній програмі. Для приведення кольору до діапазону від 0 до 1 використовується функція *color*2*vec*().

Хід виконання

1. Запустити середовище розробки *Microsoft Visual* C++ *Express Edition*.
2. Відкрити проект лабораторної роботи № 3.
3. У папці «*Source Files*» створити файли *scene.cpp* і *figures.cpp*.
4. Додати мікропрограми до файлу *shader.fx*.
5. Набрати текст програми у файлі *scene.cpp* і *figures.cpp* і внести зміни в інші файли проекту згідно вихідним кодом роботи.
6. Додати опис функцій і структур до файлу *kg_lab.h*.
7. У функції *InitScene*() у виклику конструктора *figures::s_cube()* замість знаків **####, TTT, RRR, SSS** повинні бути задані код кольору, зміщення, обертання і масштаб фігури відповідно до варіанту (див. додаток Г).
8. Налагодити і запустити програму. На екрані повинна з'явитися каркасна модель куба з заданими параметрами на чорному фоні.
9. Установити точку переривання у функції *s_figure::matrix(),* на рядку *changes = false*; (F9). Запустити програму в режимі налагодження (F5), після зупинки на точці переривання записати елементи матриці *world,* для чого натиснути правою кнопкою миші на слові world у тексті програми (*D3DXMatrixMultiply*(&*world*, &*matSR*, &*matT*);) і вибрати пункт меню *Add Watch*. У вікні перегляду значень змінних біля слова world натиснути кнопку '+' і записати числа, що з'явилися. Аналогічно записати матриці *matS, matR, matT, matSR*.
10. Підготувати звіт про лабораторну роботу, який повинен містити:
   - завдання;
   - схематичний малюнок моделі куба і проекції піраміди видимості у вигляді: зверху (проти осі *Y*), праворуч (проти осі *X*), спереду (уздовж осі *Z*);
   – матриці перетворень: *matS, matR, matT, matSR, world* (5 штук);
   – текст програм і мікропрограм (*scene.cpp, figures.cpp, kg_lab.h, shader.fx*).

Вихідний код роботи

Додати до файлу kg_lab.h:

```
...
extern D3DXHANDLE hMatIView;
}; // namespace render

namespace scene
{
using namespace directx;
using namespace render;
void RenderFigures( IDirect3DDevice9* Dev, LPD3DXEFFECT pEffect );
void UpdateSceneMatrices( LPD3DXEFFECT pEffect );
void color2vec( DWORD color_, s_vector4& obj_color );
struct s_figure
{
	s_vector3	obj_scale, obj_rot, obj_pos;
	s_vector4	obj_color;
	float		level, spower;
	mutable		s_matrix	world;
	mutable		bool		changes;
	bool		anim;

	virtual const s_vertex* verts( DWORD& cnt ) const = 0;
	virtual const t_index* inds( DWORD& cnt ) const = 0;

	const s_matrix& matrix() const;
	const s_vector4& color() const { return obj_color; }
	float spec_level() const { return level; }
	float power() const { return spower; }

	void init();
	void scale( const s_vector3& scale_ ) { obj_scale = scale_; changes = true; }
	void rotate( const s_vector3& rot_ ) { obj_rot = rot_; changes = true; }
	void translate( const s_vector3& pos_ ) { obj_pos = pos_; changes = true; }
	void set_color( DWORD color_ ) { color2vec( color_, obj_color ); }
	s_figure( const s_vector3& pos_, const s_vector3& rot_ = vec0,
		const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
		float level_ = 0.5f, float spower_ = 16.f, bool anim_ = true );
};

void ReleaseScene();
void InitScene();
}; // namespace scene

namespace figures
{
using namespace directx;
struct s_cube: public scene::s_figure
{
```

```
        virtual const s_vertex* verts( DWORD& cnt ) const;
        virtual const t_index* inds( DWORD& cnt ) const;
        s_cube( const s_vector3& pos_, const s_vector3& rot_ = vec0,
                const s_vector3& scale_ = vec1, DWORD color_ = 0xff808080UL,
                float level_ = 0.5f, float spower_ = 16.f ):
        scene::s_figure( pos_, rot_, scale_, color_, level_, spower_ ) {}
};
}; // namespace figures
```

Додати до файлу figures.cpp:

```
#include "stdafx.h"
#include "kg_lab.h"
namespace figures
{
const s_vertex* s_cube::verts( DWORD& cnt ) const
{
static s_vertex verts[] =
{
        s_vertex( s_vector3( -1.f, -1.f, -1.f ) ),
        s_vertex( s_vector3(  1.f, -1.f, -1.f ) ),
        s_vertex( s_vector3( -1.f,  1.f, -1.f ) ),
        s_vertex( s_vector3(  1.f,  1.f, -1.f ) ),

        s_vertex( s_vector3( -1.f, -1.f,  1.f ) ),
        s_vertex( s_vector3(  1.f, -1.f,  1.f ) ),
        s_vertex( s_vector3( -1.f,  1.f,  1.f ) ),
        s_vertex( s_vector3(  1.f,  1.f,  1.f ) ),

        s_vertex( s_vector3( -1.f, -1.f, -1.f ) ),
        s_vertex( s_vector3(  1.f, -1.f, -1.f ) ),
        s_vertex( s_vector3( -1.f, -1.f,  1.f ) ),
        s_vertex( s_vector3(  1.f, -1.f,  1.f ) ),

        s_vertex( s_vector3( -1.f,  1.f, -1.f ) ),
        s_vertex( s_vector3(  1.f,  1.f, -1.f ) ),
        s_vertex( s_vector3( -1.f,  1.f,  1.f ) ),
        s_vertex( s_vector3(  1.f,  1.f,  1.f ) ),

        s_vertex( s_vector3( -1.f, -1.f, -1.f ) ),
        s_vertex( s_vector3( -1.f, -1.f,  1.f ) ),
        s_vertex( s_vector3( -1.f,  1.f, -1.f ) ),
        s_vertex( s_vector3( -1.f,  1.f,  1.f ) ),

        s_vertex( s_vector3(  1.f, -1.f, -1.f ) ),
        s_vertex( s_vector3(  1.f, -1.f,  1.f ) ),
        s_vertex( s_vector3(  1.f,  1.f, -1.f ) ),
        s_vertex( s_vector3(  1.f,  1.f,  1.f ) ),
};

cnt = sizeof( verts ) / sizeof( verts[0] );
```

```
return verts;
}

const t_index* s_cube::inds( DWORD& cnt ) const
{

static const t_index inds[] =
{
        1, 0, 2, 1, 2, 3,           //далека грань
        4, 5, 6, 5, 7, 6,           //ближня грань
        8, 9, 10, 9, 11, 10,        //верхня межа
        14, 13, 12, 14, 15, 13, //нижня межа

        16, 17, 19, 16, 19, 18, //ліва грань
        21, 20, 23, 20, 22, 23, //права грань
};
cnt = sizeof( inds ) / sizeof( inds[0] );
return inds;
}

}; // namespace figures
```

Додати до файлу scene.cpp:

```
#include "stdafx.h"
#include "kg_lab.h"
namespace scene
{
s_figure* pSceneRoot = 0;
s_vector3 vEyePos;
s_vector3 vAtPos;
float     fFovY;
void color2vec( DWORD color_, s_vector4& obj_color )
{
DWORD red = (color_ >> 16) & 0xff;
DWORD green = (color_ >> 8) & 0xff;
DWORD blue = color_ & 0xff;
DWORD alpha = (color_ >> 24) & 0xff;
obj_color = s_vector4( red / 255.f, green / 255.f, blue / 255.f, alpha / 255.f  );
}
void RenderFigures( IDirect3DDevice9* Dev, LPD3DXEFFECT pEffect )
{
const s_figure* f = pSceneRoot;
DWORD vcnt = 0, icnt = 0;
const s_vertex* v = f->verts( vcnt );
const t_index* i = f-> inds( icnt );
DWORD tri_cnt = icnt / 3;

const s_matrix& MatWorld = f->matrix();
const s_vector4& vObjColor = f->color();
```

```
pEffect->SetMatrix( hMatWorld, &MatWorld );
pEffect->SetVector( hObjColor, &vObjColor );

pEffect->CommitChanges();

Dev->DrawIndexedPrimitiveUP( D3DPT_TRIANGLELIST, 0, vcnt, tri_cnt,
                  i, D3DFMT_INDEX16,  v, sizeof(s_vertex) );
}
s_figure::s_figure( const s_vector3& pos_, const s_vector3& rot_,
const s_vector3& scale_,  DWORD color_, float level_, float spower_, bool anim_ ):
obj_scale(scale_), obj_rot(rot_), obj_pos(pos_), level(level_), spower(spower_),
changes(true)
{
D3DXMatrixIdentity( &world );
set_color( color_ );
}

const s_matrix& s_figure::matrix() const
{
if( changes )
{
        s_matrix matS, matR, matT, matSR;
        D3DXMatrixScaling( &matS, obj_scale.x, obj_scale.y, obj_scale.z );
        D3DXMatrixRotationYawPitchRoll( &matR, obj_rot.x * deg2rad,
                obj_rot.y * deg2rad, obj_rot.z * deg2rad );
        D3DXMatrixTranslation( &matT, obj_pos.x, obj_pos.y, obj_pos.z );
        D3DXMatrixMultiply( &matSR, &matS, &matR );
        D3DXMatrixMultiply( &world, &matSR, &matT );
        changes = false;
} // if( changes )

return world;
}

void ReleaseScene()
{
        delete pSceneRoot;
        pSceneRoot = 0;
}

void InitScene ()
{
vEyePos = vec0;
vAtPos = s_vector3( 0.f, 0.f, 1.f );
fFovY = 60.f;

pSceneRoot = new figures::s_cube(
        s_vector3( TTT ), s_vector3( RRR ), s_vector3( SSS ), #### );
}
```

Увага! У функції *InitScene*() у виклику конструктора *figures::s_cube* - замість знаків **####, TTT, RRR, SSS** повинні бути задані код кольору, зміщення, обертання і масштаб фігури (див. додаток Г).

```
void UpdateSceneMatrices( LPD3DXEFFECT pEffect )
{
const s_vector3 Up( 0.f, 1.f, 0.f );
s_matrix MatView, MatIView;
D3DXMatrixLookAtLH( &MatView, &vEyePos, &vAtPos, &Up );
float FOV = fFovY * deg2rad;
float Aspect = (float)PP.BackBufferWidth / PP.BackBufferHeight;
float ZNear = 1.f;
float ZFar = 100.f;

s_matrix MatProj;
D3DXMatrixPerspectiveFovLH( &MatProj, FOV, Aspect, ZNear, ZFar );
s_matrix MatViewProj;
D3DXMatrixMultiply( &MatViewProj, &MatView, &MatProj );

float det;
D3DXMatrixInverse( &MatIView, &det, &MatView );
pEffect-> SetMatrix( hMatIView, &MatIView );
pEffect-> SetMatrix( hMatViewProj, &MatViewProj );
}
}; // namespace scene
```

Додати виклики функцій *ReleaseScene*(), *InitScene*(), *UpdateSceneMatrices*(), *RenderFigures*() до файлу ***render.cpp***

```
void OnExit( IDirect3DDevice9* Dev )
{
scene::ReleaseScene();
...

bool LoadResources( IDirect3DDevice9* Dev )
{
...
if( !pDecl )
        return false;

scene::InitScene();
return true;
}


void RenderScene( IDirect3DDevice9* Dev, int TN )
{
...
pEffect->SetTechnique( hTechnique [TN] );
Dev->SetVertexDeclaration( pDecl );
```

```
scene::UpdateSceneMatrices( pEffect );
...
for( UINT i = 0; i < passes; ++i )
{
        if( pEffect->BeginPass(i) != D3D_OK )
                break;
        scene::RenderFigures( Dev, pEffect );
        pEffect->EndPass();
...
}
```

Додати у файл shader.fx:

```
...
// параметри об'єкта
float4x4 m_world;
float4 v_obj_color;
float f_power, f_spec_level;

struct s_vertex2
{
   float4       pos: POSITION;
   float3       nor: NORMAL;
};
struct s_output2
{
   float4       pos: POSITION;
   float3       nor: TEXCOORD0;
   float3       eye: TEXCOORD1;
   float3       wpos: TEXCOORD2;
};

s_output2 vs_simple(s_vertex2 i)
{
s_output2 o;
float4 P = mul(i.pos, m_world);

o.wpos = P.xyz;
o.pos = mul(P, m_view_proj);
o.nor = mul(i.nor, (float3x3)m_world);
o.eye = mul(float3( 0.f, 0.f, 1.f ), (float3x3)m_iview);
return o;
}

float4 ps_wire(s_output2 i): COLOR
{
        return float4(v_obj_color.rgb, 1.0);
}
technique wire
{
```

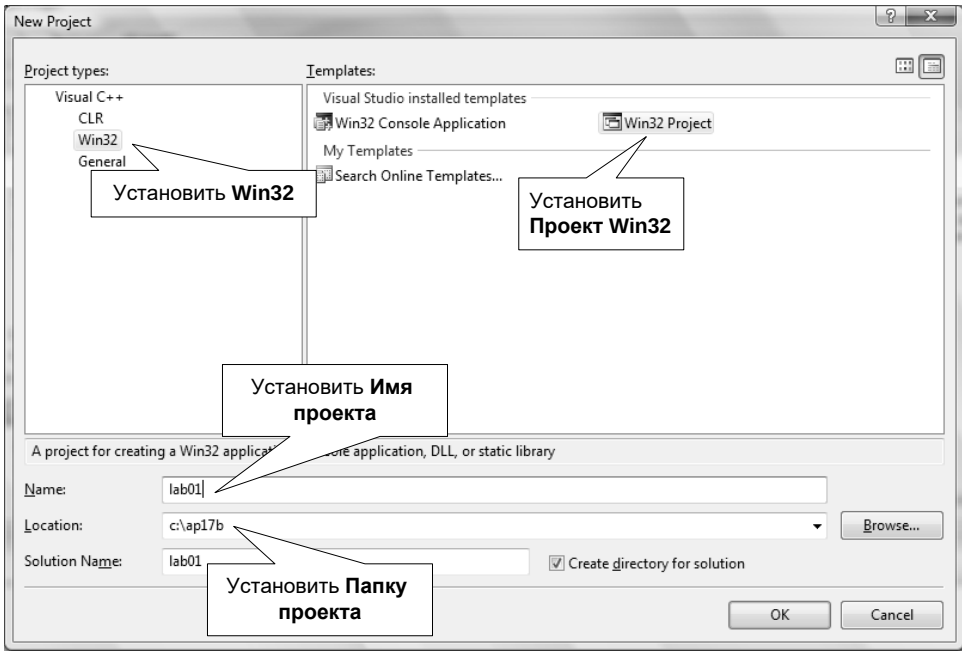
```
    pass P0
    {
        ...
        ZFunc           = LessEqual;
        CullMode        = CCW;
        VertexShader    = compile vs_2_0 vs_simple();
        PixelShader     = compile ps_2_0 ps_wire();
    }
}
```

Контрольні питання

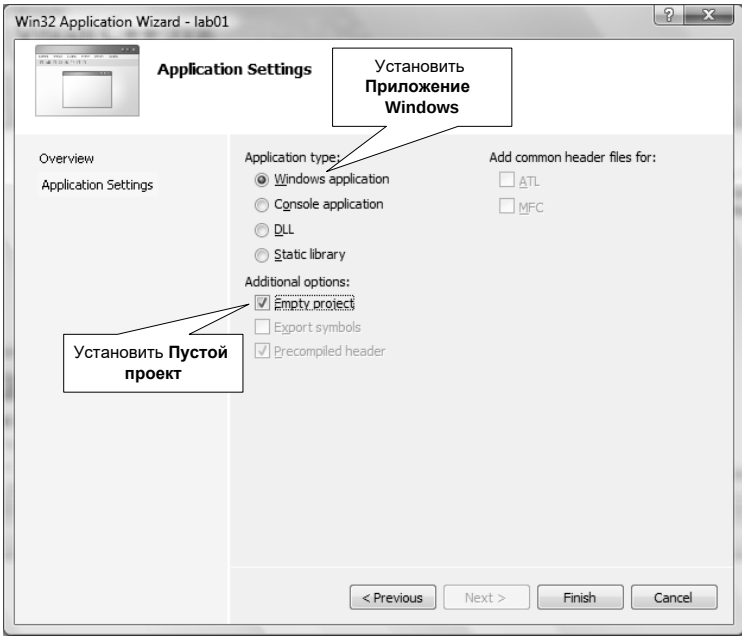
1. Який тип даних використовується для визначення матриці мовою HLSL?
2. Яке ключове слово використовується для визначення константи?
3. Які вхідні і вихідні дані необхідно задати для побудови моделі?
4. Який інтерфейс використовується для опису геометричної фігури?
5. Які функції необхідно визначити в програмі для створення фігури?
6. В якій функції розраховуються і встановлюються матриці, що описують систему координат спостерігача?
7. В якій функції проводиться розрахунок матриці об'єкта, її установка в мікропрограму і висновок геометрії (вершин і індексів) фігури на екран?

ДОДАТКИ
Додаток А
Створення проекту

1. Вибрати *File -> New -> Project*. У діалоговому вікні встановити:

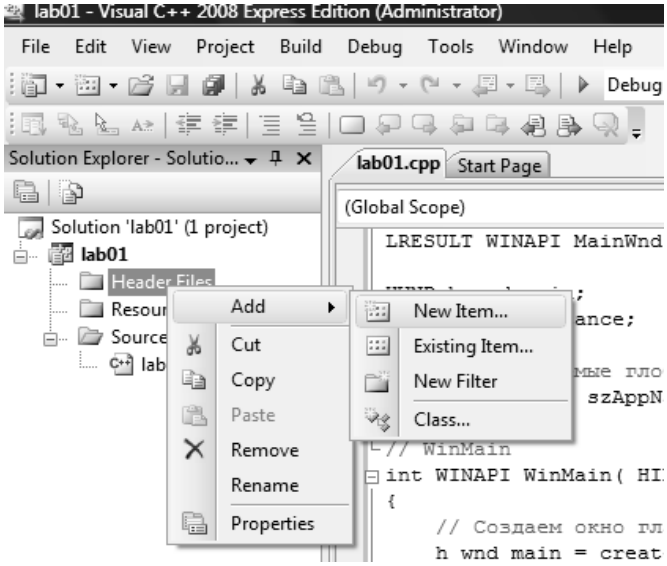


У новому вікні натиснути кнопку «*Next*» і в наступному вікні встановити і натиснути кнопку «*Finish*».

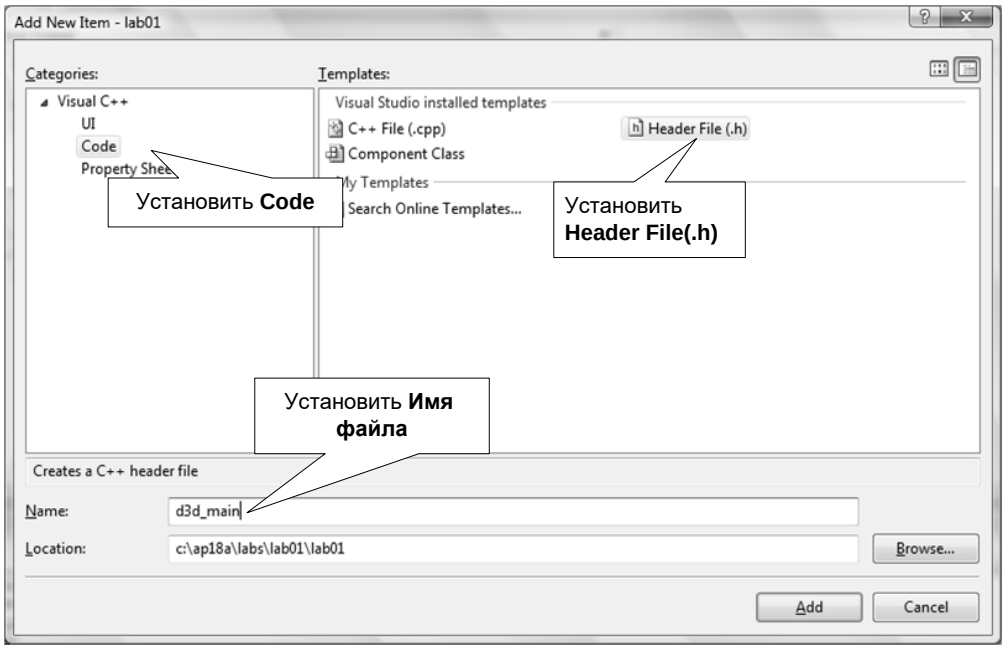


2. Додавання заголовних файлів до проекту.

Правою клавішею клікнути на папці «*Header files*» в «*Solution Explorer*». Далі вибрати «*Add*» -> «*New Item*»

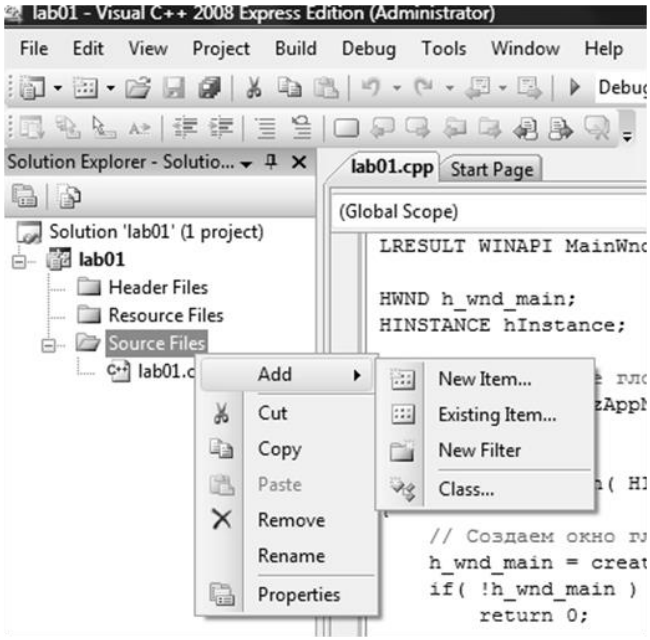


У діалоговому вікні вибрати наступне і натиснути кнопку «Add».

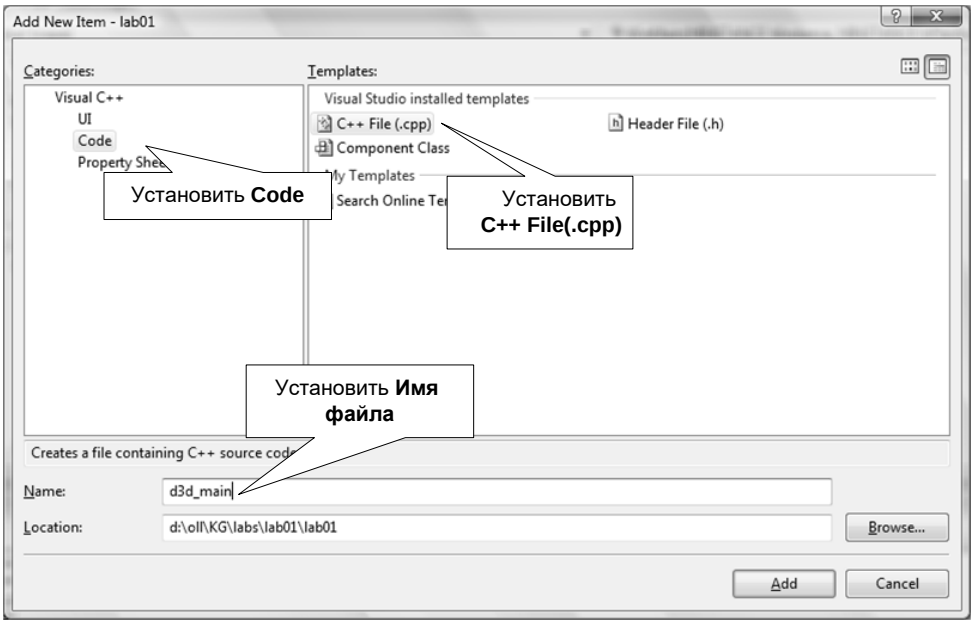


3. Додавання програмних файлів до проекту.

Правою клавішею клікнути на папці «*Header files*» в «*Solution Explorer*». Далі вибрати «*Add*» -> «*New Item*»

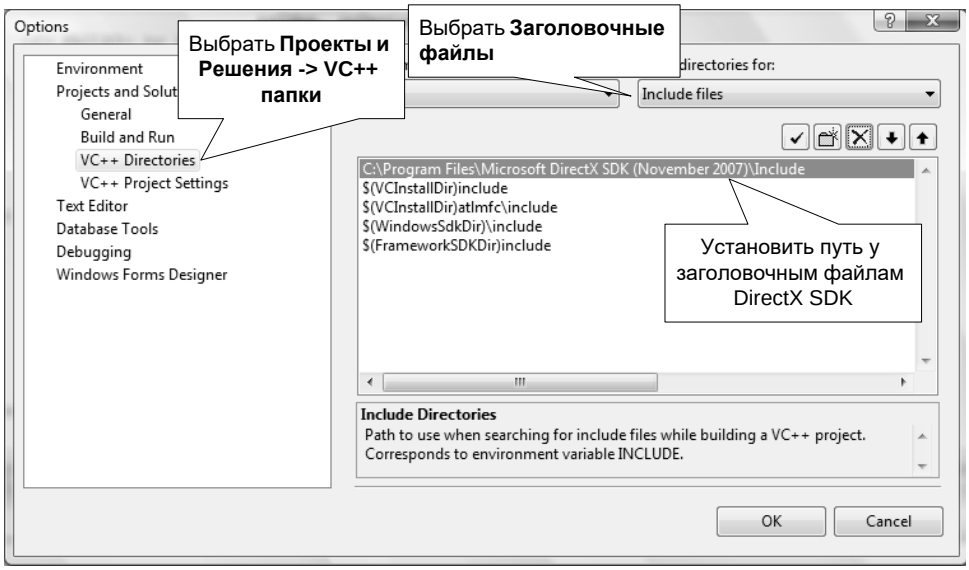


У діалоговому вікні вибрати наступне:

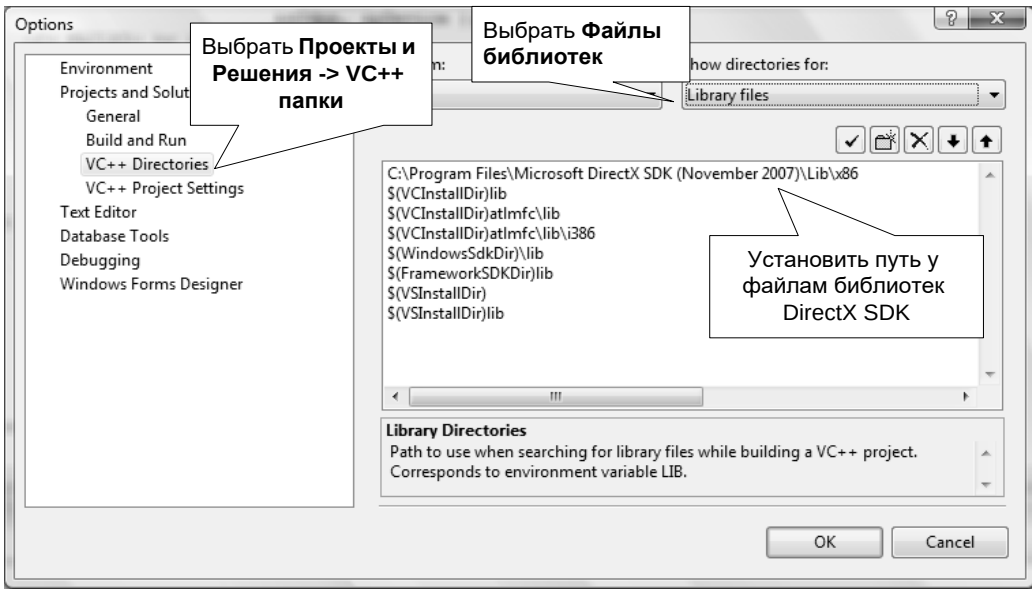


Додаток Б
Налаштування проекту для доступу до бібліотеки DirectX SDK

1. Підключення заголовкових файлів DirectX SDK. Вибрати *Tools -> Option*s. У діалоговому вікні встановити:



2. Підключення файлів бібліотек DirectX SDK. Вибрати *Tools -> Options*. У діалоговому вікні встановити:



Додаток В
Кольори для функції Clear

№ варіанту	Колір
1	темно-синій
2	червоний
3	світло-зелений
4	сірий
5	чорний
6	коричневий
7	жовтий
8	ціан
9	пурпурний
10	помаранчевий
11	фіолетовий
12	рожевий
13	темно-червоний
14	бурштиновий
15	бронзовий
16	лазурний
17	аквамариновий
18	кремовий
19	смарагдовий
20	золотий

Код кольору подається у вигляді 32-х розрядного числа в 16-річній системі числення. Кожні 8 біт числа (дві цифри) відповідають за свою компоненту кольору в системі RGB – починаючи зі старших до молодших бітів: A (додатковий канал), R,G,B.

Наприклад, для лимонного кольору код буде таким: 0xFFFDE910.

Канал	A	R	G	B
Біти	31-24	23-16	15-8	7-0
Значення	FF	FD	E9	10

Для задання кольору доцільно скористатися програмою Paint (натиснути клавіші Win+R, у рядку введення ввести pbrush і натиснути Enter). У діалозі вибору кольорів пересунути повзунки на заданий колір, скорегувати яскравість. Перевести 3 коди (Red, Green Blue) в 16-річну систему числення і записати у відповідних позиціях. Код у каналі A взяти рівним 255 (FF). Отримане значення записати у виклику функції Clear замість знаків ####.

Додаток Г
Колір і параметри каркасної моделі куба

№ варіанту	Колір (####)	Масштаб (**SSS**)	Обертання (**RRR**)	Зсув (**TTT**)
1	червоний	2.f, 2.f, 2.f	45.f, 0.f, 0.f	10.f, -4.f, 30.f
2	зелений	3.f, 3.f, 3.f	30.f, 30.f, 0.f	-4.f, -4.f, 25.f
3	білий	1.f, 1.f, 1.f	0.f, 0.f, 0.f	1.f, 0.f, 10.f
4	сірий	2.f, 3.f, 3.f	45.f, 0.f, 0.f	-5.f, 8.f, 25.f
5	жовтий	3.f, 3.f, 3.f	30.f, 30.f, 0.f	-4.f, -4.f, 25.f
6	ціан	2.f, 3.f, 3.f	0.f, 0.f, 0.f	1.f, 0.f, 25.f
7	пурпурний	2.f, 2.f, 2.f	0.f, 0.f, 60.f	-1.f, -0.5f, 8.f
8	синій	2.f, 3.f, 3.f	30.f, 30.f, 0.f	-8.f, 0.f, 35.f
9	фіолетовий	3.f, 1.f, 2.f	45.f, 0.f, 0.f	1.f, 0.f, 25.f
10	помаранчевий	5.f, 1.f, 1.f	0.f, 0.f, 60.f	-4.f, -4.f, 25.f
11	зелений	1.f, 1.f, 1.f	0.f, 0.f, 0.f	-5.f, 8.f, 25.f
12	червоний	3.f, 3.f, 3.f	0.f, 0.f, 60.f	-8.f, 0.f, 35.f
13	синій	2.f, 3.f, 3.f	30.f, 30.f, 0.f	10.f, -4.f, 30.f
14	ціан	3.f, 1.f, 2.f	0.f, 0.f, 60.f	-4.f, -4.f, 25.f
15	білий	2.f, 3.f, 3.f	0.f, 0.f, 0.f	-5.f, 8.f, 25.f
16	зелений	2.f, 2.f, 2.f	45.f, 0.f, 0.f	1.f, 0.f, 25.f
17	червоний	2.f, 3.f, 3.f	30.f, 30.f, 0.f	-4.f, -4.f, 25.f
18	синій	3.f, 1.f, 2.f	0.f, 0.f, 60.f	-5.f, 8.f, 25.f
19	жовтий	3.f, 3.f, 3.f	45.f, 0.f, 0.f	10.f, -4.f, 30.f
20	зелений	1.f, 1.f, 1.f	30.f, 30.f, 0.f	2.f, 0.f, 8.f

Список літератури

1. Гилой В. Интерактивная машинная графика : структуры данных, алгоритмы, языки / В. Гилой. — М. : Мир, 1981. — 380 с.
2. Дуда Р. Распознавание образов и анализ сцен : пер. с англ. / Р. Дуда, П. Харт. — М. : Мир, 1976. — Гл. 7, 9.
3. Ньюмен П. Основы интерактивной машинной графики / П. Ньюмен, Р. Спрулл ; пер. с англ. В. М. Грина, О. Н. Родинко. — М. : Мир, 1976. — 573 с.
4. Павлидис Т. Алгоритмы машинной графики и обработки изображений / Т. Павлидис. — М. : Радио и связь, 1986. — 198 с.
5. Порев В.Н. Компьютерная графика : учеб. пособие / Виктор Порев. — СПб. : BHV-Санкт-Петербург, 2002. — 428 с.
6. Прэтт У. Цифровая обработка изображений : в 2 кн. / У. Прэтт ; пер. с англ. под ред. Д. С. Лебедева. — М. : Мир, 1982. — Гл. 2, 3, 12, 13, 17, 18, 20.
7. Роджерс Д. Алгоритмические основы машинной графики : пер. с англ. / Д. Роджерс. — М. : Мир, 1989. — 503 с.
8. Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс ; пер. со 2-го англ. изд. П. А. Монахова [и др.]. — М. : Мир, 2001. — 604 с.
9. Тихомиров Ю. Программирование трехмерной графики : OpenGL, создание реалистических образов / Ю. Тихомиров. СПб. : BHV-Санкт-Петербург, 1998. — 245 с.
10. Фоли Дж. Основы интерактивной машинной графики : в 2 кн. / Дж. Фоли. — М. : Мир, 1987. — Кн. 1. — 367 с. ; Кн. 2. — 365 с.
11. Шикин Е. В. Компьютерная графика. Динамика, реалистические изображения / Е. В. Шикин, А. В. Боресков. — М. : ДИАЛОГ-МИФИ, 1995. — 286 с.
12. Шикин Е. В. Компьютерная графика. Полигональные модели / Е. В. Шикин, А. В. Боресков. — М. : Диалог-МИФИ, 2000. — 461 с.
13. Эйнджел Э. Интерактивная компьютерная графика : ввод. курс на базе OpenGL / Эдвард Эйнджел ; пер. с англ. и ред. В. Т. Тертышного. — 2-е изд. — М. : Вильямс, 2001. — 590 с.

Навчальне видання

ЗУЄВ Андрій Олександрович
ЄВСЕЄНКО Олег Миколайович
КРИЛОВА Вікторія Анатоліївна

КОМП'ЮТЕРНА ГРАФІКА.
ЧАСТИНА 1

Методичні вказівки

до виконання лабораторних робіт
для студентів спеціальностей
151 «Автоматизація та комп'ютерно-інтегровані технології»
172 «Телекомунікація та радіотехніка»

Відповідальний за випуск Качанов П.О.
Роботу до друку рекомендував О.В. Дудник

Редактор Самініна О.С.

План 2020 р., Поз. 58

Підписано до друку . Формат 60'84 1/16. Папір друк. № 2.
Друк - ризографія. Гарнітура Times New Roman. Умов. друк. арк. 3,2.
Обл.-вид. арк. 2,7. Наклад 100 прим. Зам. № . Ціна договірна.
