МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ ВИЩИЙ НАВЧАЛЬНИЙ ЗАКЛАД
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
КАФЕДРА «ПРИКЛАДНА МАТЕМАТИКА ТА ІНФОРМАТИКА»

М Е Т О Д И Ч Н І В К А З І В К И
ДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ
ЧАСТИНА 1. РАСТРОВА ГРАФІКА

Дисципліна: Комп’ютерна графіка

Розробник:

Державний вищий навчальний заклад

«Донецький національний технічний університет»

факультет комп’ютерних наук і технологій

кафедра прикладної математики і інформатики

2018

УДК 004.032.6 : 004.92

Методичні вказівки до виконання лабораторних робіт за модулем «OpenGL» дисципліни «Комп’ютерна графіка» підготовки бакалаврів спеціальності 121 «Інженерія програмного забезпечення» / Укладачі: Є.О. Башков, П.О. Ануфрієв. – Покровськ: ДонНТУ, 2018. – 74 с.

Укладачі:
Є.О. Башков, професор кафедри ПМІ, д.т.н., професор;
П.О. Ануфрієв, асистент кафедри ПМІ.

ЗМІСТ

ЛАБОРАТОРНА РОБОТА №1. ВИВІД ОДИНОЧНОГО ПІКСЕЛЯ НА ЕКРАН

Мета лабораторної роботи: Ознайомитися з середовищем MS VisualStudio, мовою C++, особливостями створення вікна WinAPI, виводом одиночних пікселів на екрані. Опрацювати набуті знання щодо найменшого елементу на екрані; за допомогою середовища VisualStudio створити елементи у вікні.

1. Встановлення Visual Studio

Visual Studio — серія продуктів фірми Microsoft, які включають інтегроване середовище розробки програмного забезпечення та ряд інших інструментальних засобів. Ці продукти дозволяють розробляти як консольні програми, так і програми з *графічним* інтерфейсом, в тому числі з підтримкою технології Windows Forms [1].

Для виконання лабораторних робіт буде досить безкоштовної версії програми Visual Studio Community 2017 [2].

Для завантаження Visual Studio Installer перейдіть за посиланням https://www.visualstudio.com/ та під кнопкою «Download Visual Studio» обрати вкладку «Community 2017» (рисунок 1.1). Потім відкрити завантажений файл для встановлення програми. У встановленій програмі обрати Visual Studio Community (рисунок 1.2).

За допомогою Visual Studio Installer рекомендуємо обрати наступні робочі навантаження:

- Розробка класичних додатків на C++ (Desktop development with C++) (рисунок 1.3);
- Розробка ігор на мові C++ (Game development with C++) (рисунок 1.4).

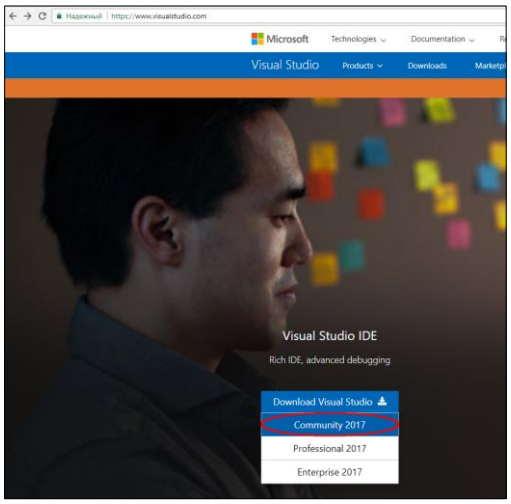

Рисунок 1.1 —Завантаження Visual Studio з офіційного сайту

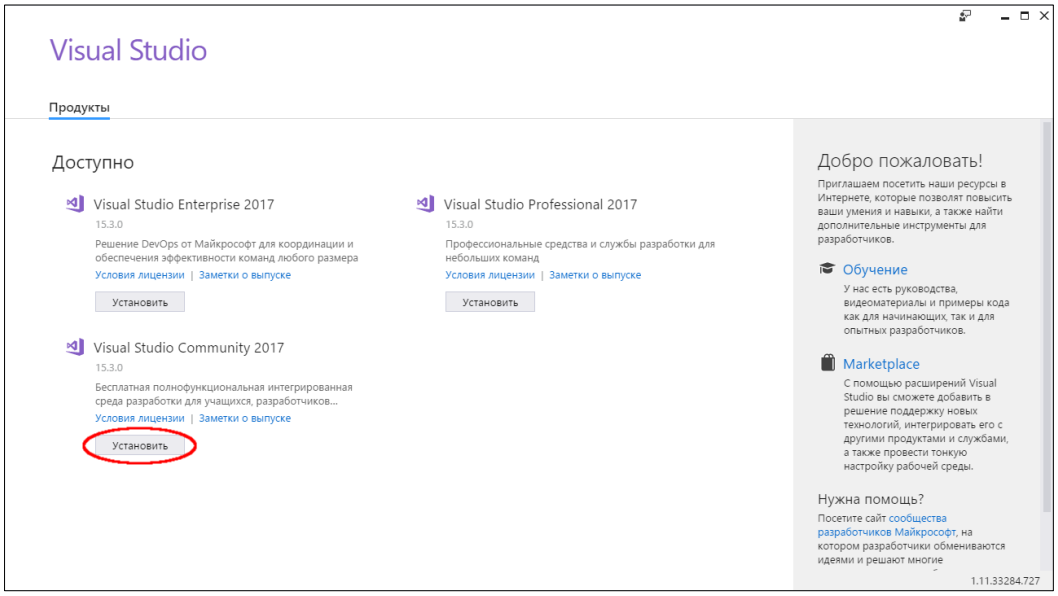

Рисунок 1.2 - Вибір програми Visual Studio Community

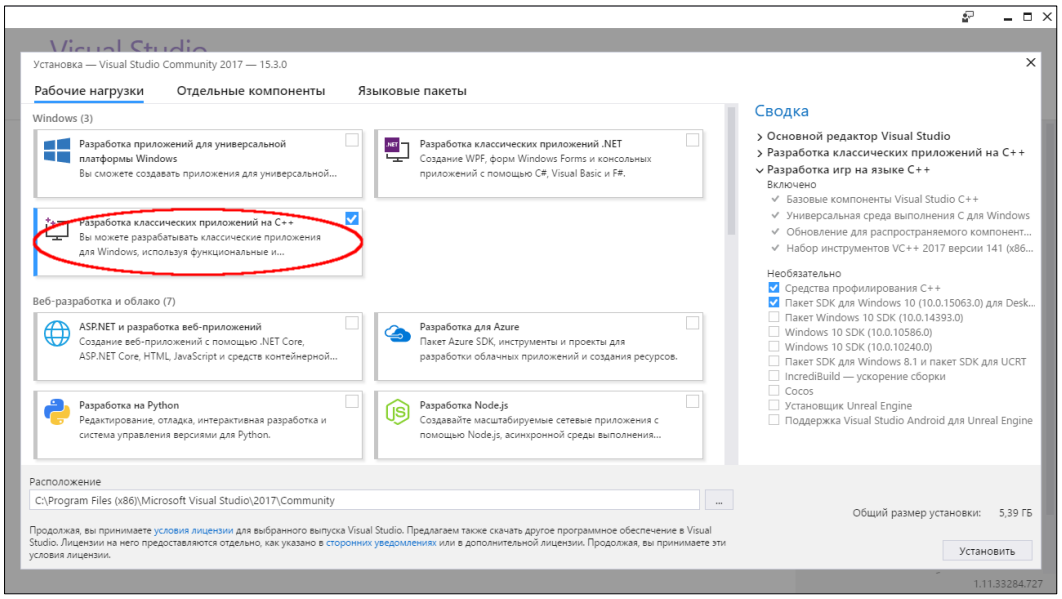

Рисунок 1.3 — Завантаження “Розробка класичних додатків на C++”

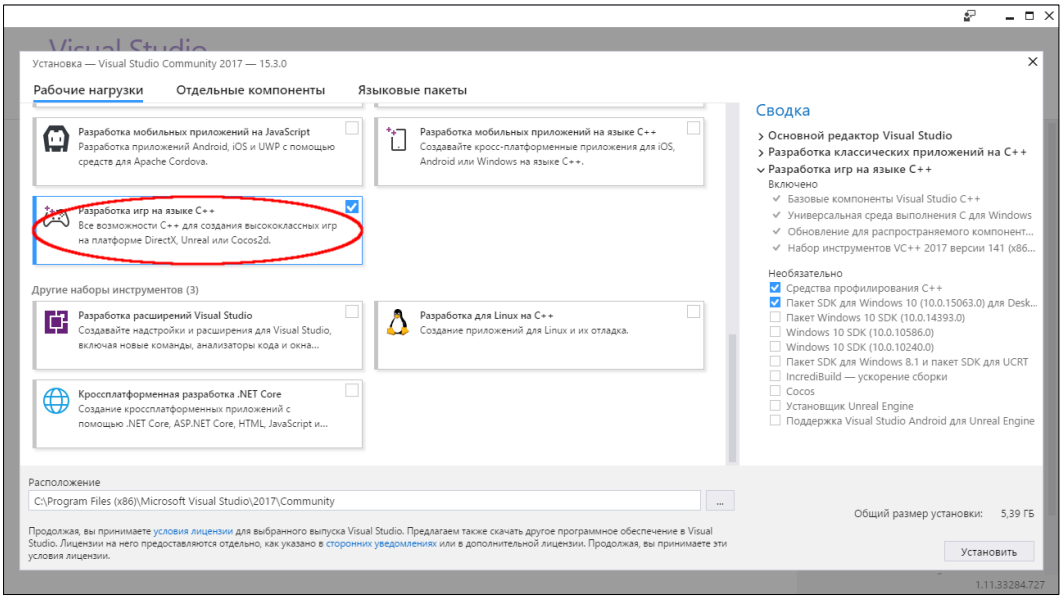

Рисунок 1.4 — Завантаження “Розробка ігор на мові C++“

Також рекомендуємо завантажити англомовний пакет. Практично вся література та інструкції для виконання робіт з графіки, робоча зона Visual Studio виконані на англійській мові. Це допоможе вам виконати все якомога простіше (рисунок 1.5).

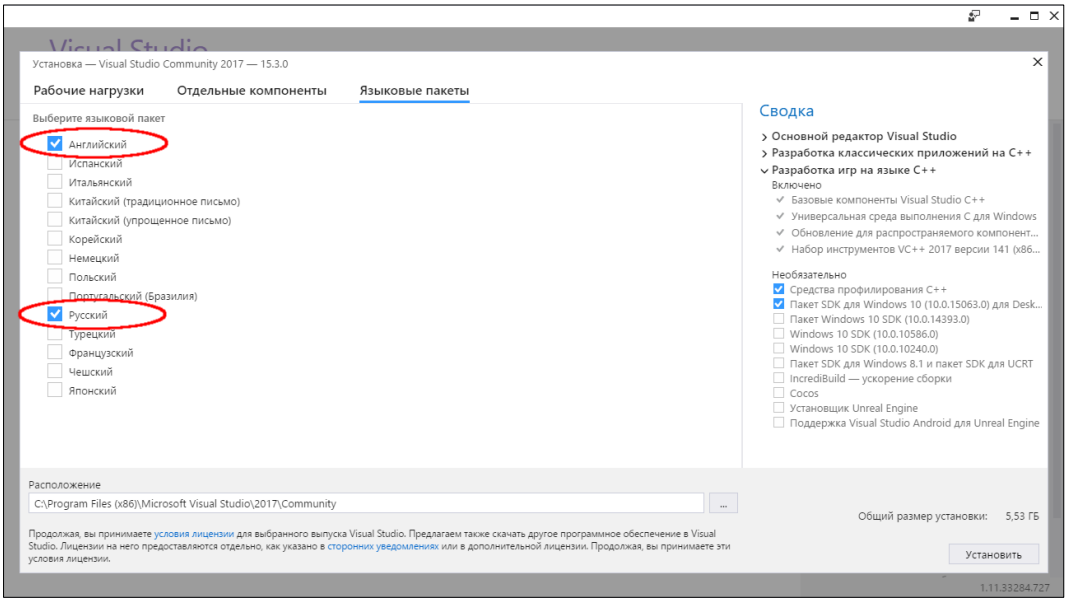

Рисунок 1.5 — Завантаження мовних пакетів

Після завантаження та першого запуску повинна бути створена наступна структура директорій:

- файли програми зберігаються в C:\Program Files (x86)\Microsoft Visual Studio\2017\Community (рисунок 1.6);
- файли проектів зберігаються в C:\Users\Student\Documents\Visual Studio 2017 (рисунок 1.6).

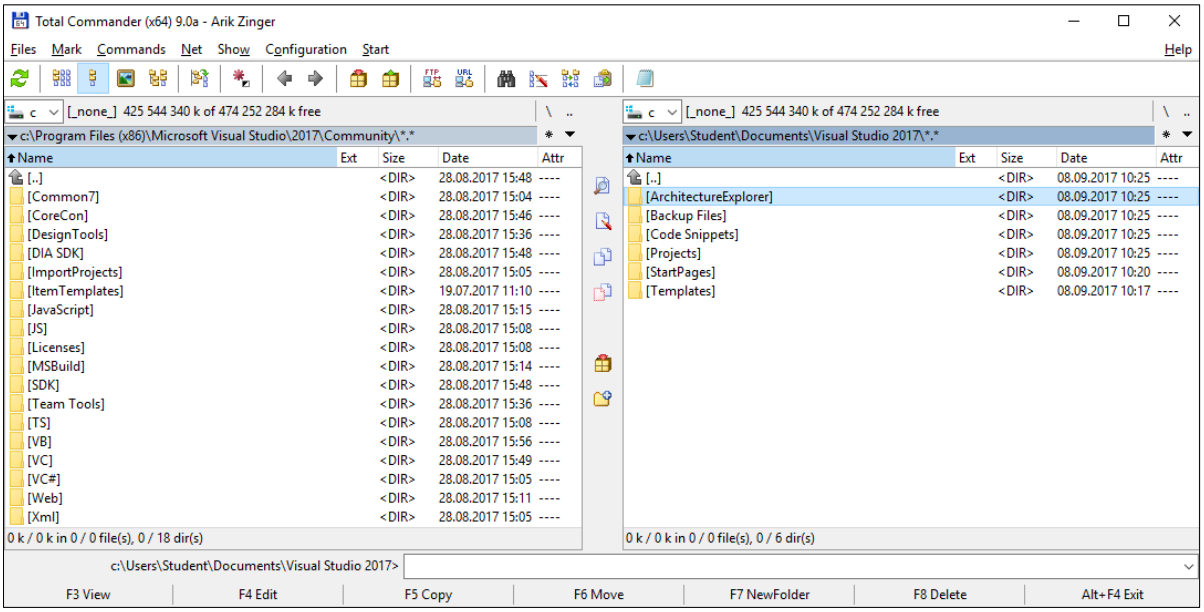

Рисунок 1.6 — Директорії, для розташування програмних та власних проектних файлів у Visual Studio

Примітка: згідно умов ліцензійної згоди на використання попередньої версії програмного забезпечення Microsoft Visual Studio Community 2017, пункту 3 «Програмне забезпечення, яке надається на певний час»:

“Це програмне забезпечення надається на певний термін. Це означає, що воно перестане працювати в дату, задану в програмному забезпеченні, і ваша ліцензія на використання ПЗ також закінчиться. Після припинення роботи програмного забезпечення ви можете втратити доступ до копій коду або іншими даними, які в ньому зберігалися.“ [3]. Для подальшого використання програми потрібно ввести власну адресу електронної пошти Microsoft та пароль в полі входу Visual Studio Community (рисунок 1.7).

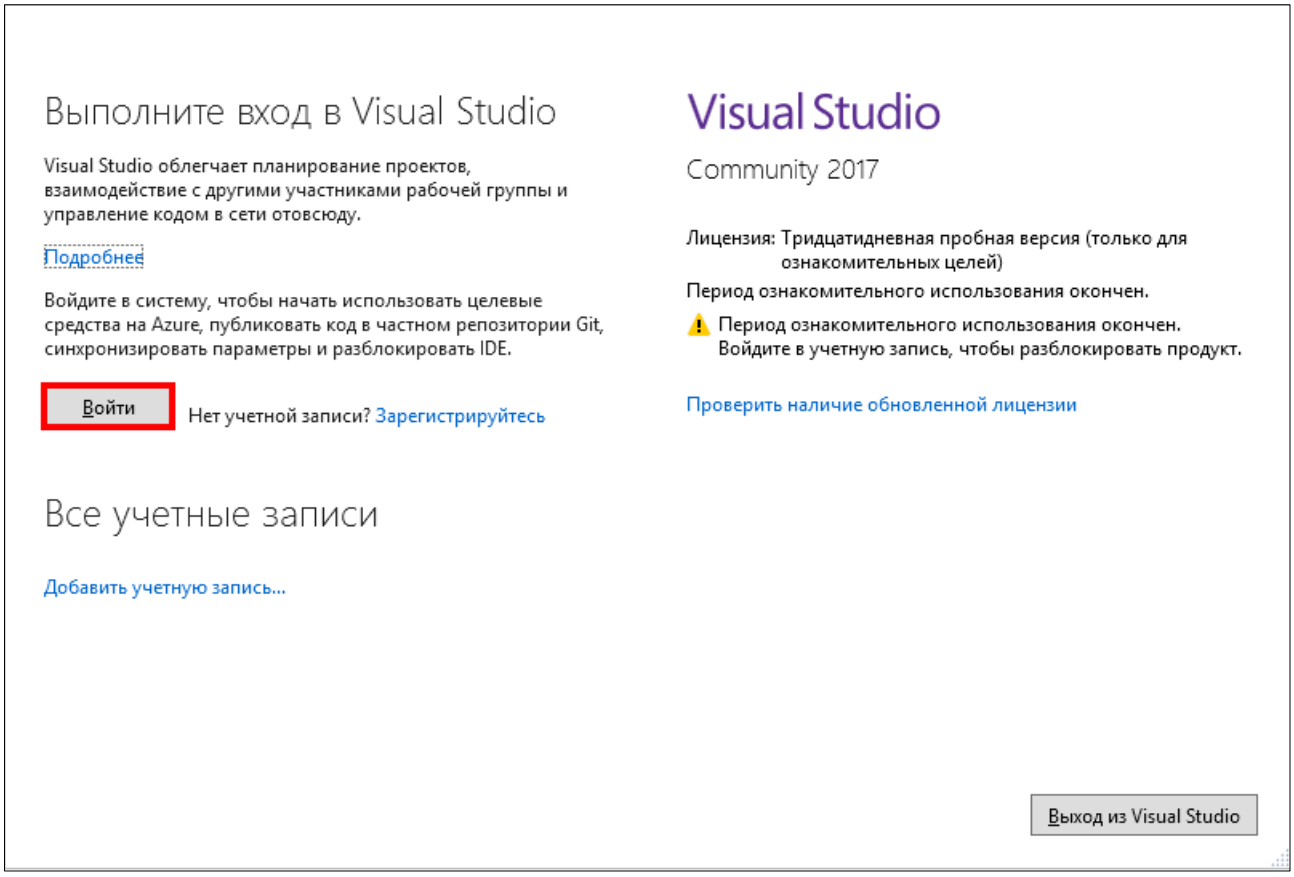

Рисунок 1.7 — Закінчення ліцензії на попередню версії Visual Studio

2. Створення проекту

Для створення першого проекту необхідно виконати наступні дії:

1. Запустити Visual Studio з панелі задач або з робочого столу
2. Обрати “Створити проект” на Початковому екрані (рисунок 2.1)

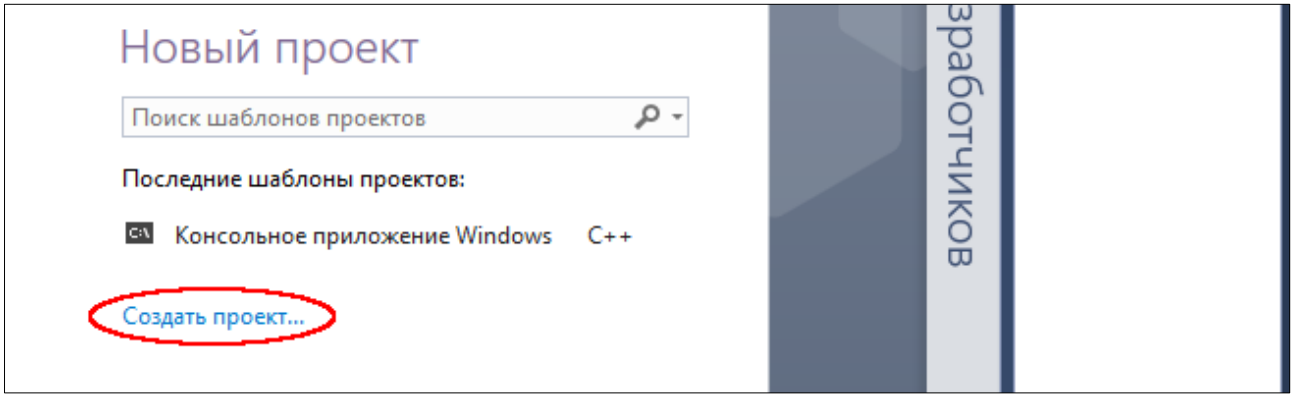

Рисунок 2.1 — Створення нового проекту

3. Обрати вкладку «Visual C++» (рисунок 2.2)
4. Далі обрати «Порожній проект» (рисунок 2.2)
5. У вкладці “Ім'я” написати Прізвище Ім'я По батькові та групу у форматі “Прізвище_ІБ_Группа_номер” (Іванов_ІІ_ПМІ_17) (рисунок 2.2)
6. Обрати директорію для створення проекту “Projects” (шлях c:\Users\Student\Documents\Visual Studio 2017\Projects) (рисунок 2.2)

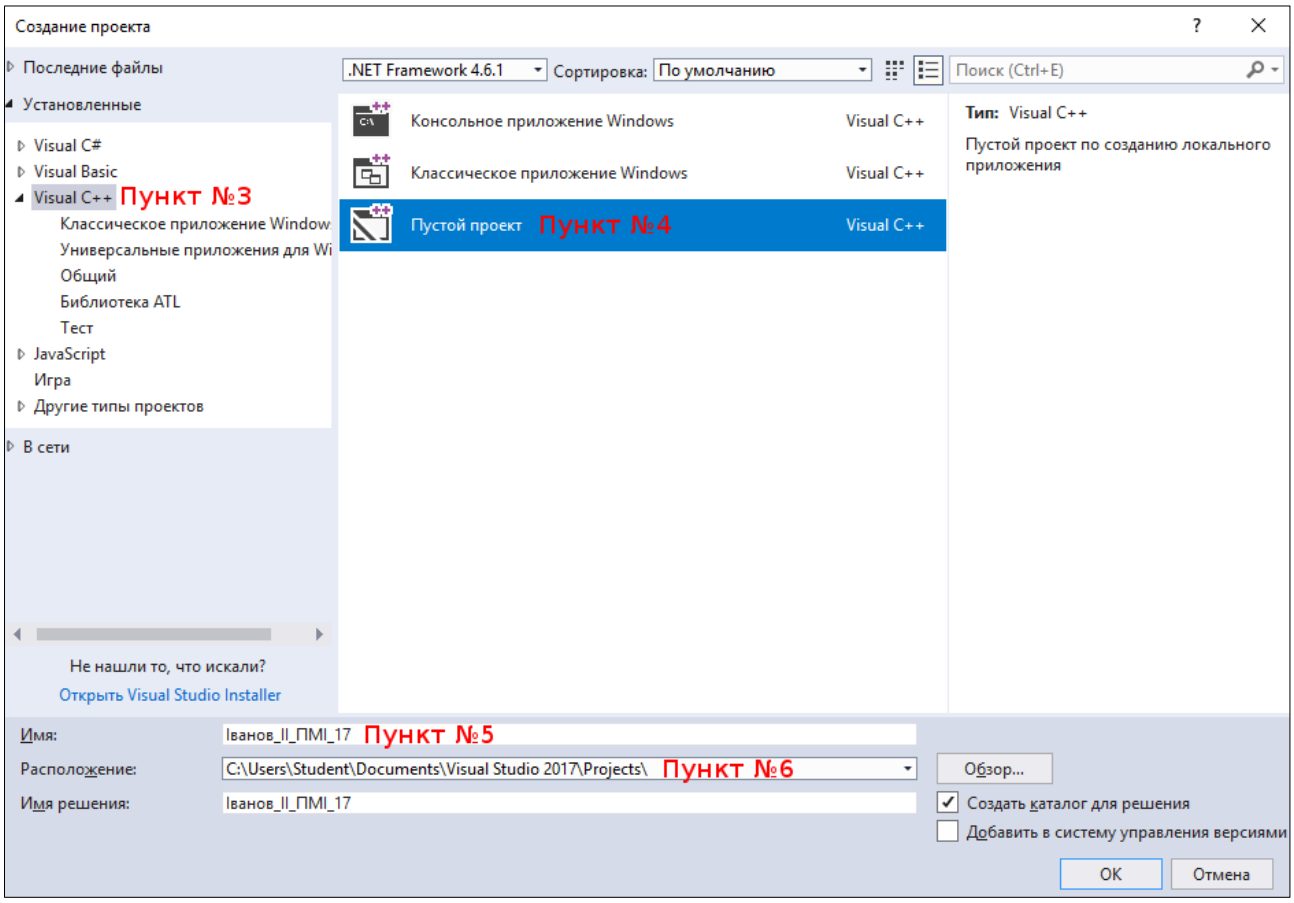

Рисунок 2.2 — Створення порожнього проекту

7. В “Оглядачі рішень” правою кнопкою миші натиснути на “Іванов_II_ПМІ_17”, вибрати Додати/Створити елемент (рисунок 2.3)

8. У наступному вікні обрати “Файл C++”, та назвати, наприклад, “Start.cpp” (рисунок 2.4)

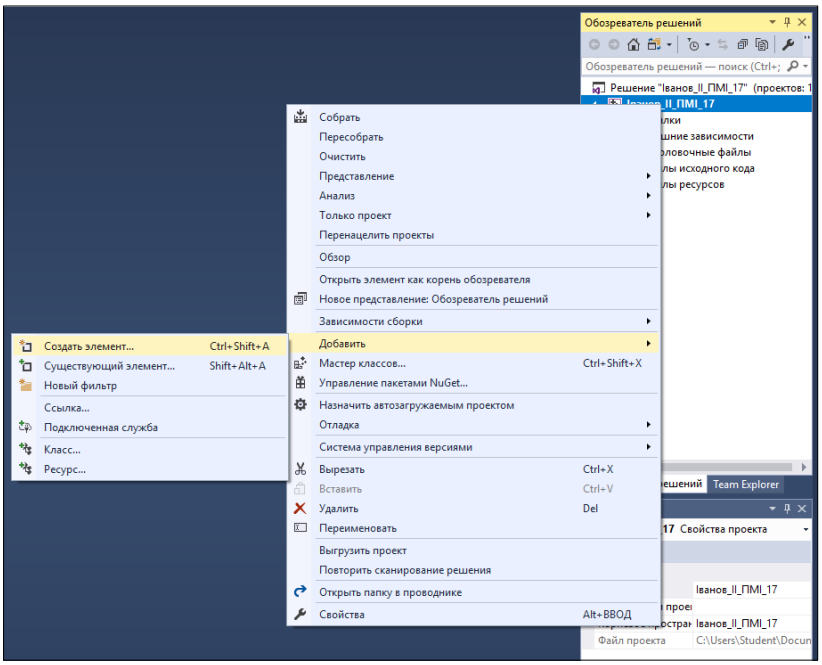

Рисунок 2.3 — Створення нового елементу

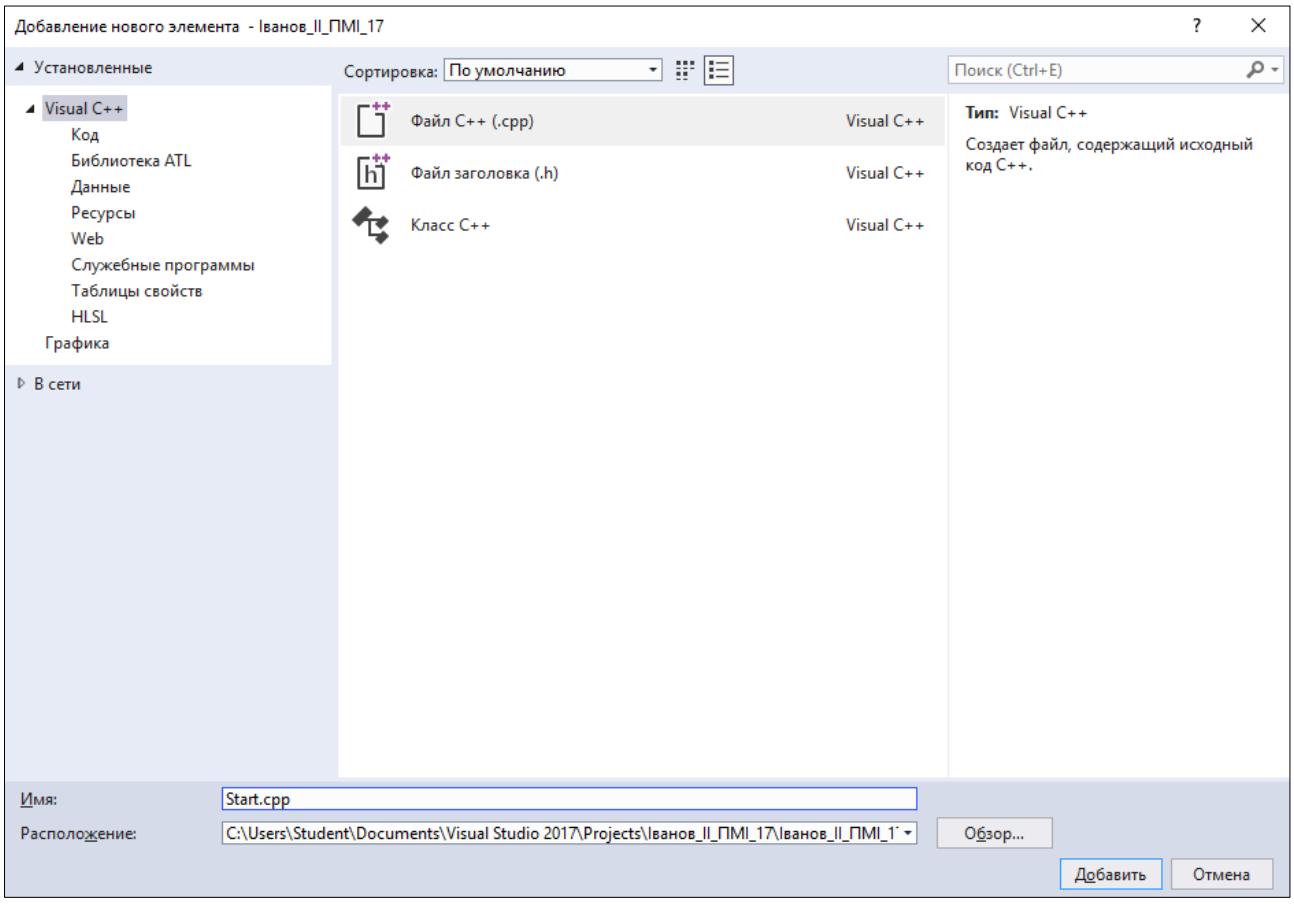

Рисунок 2.4 — Вибір файлу C++

9. Скопіювати наступний код у створений Start.cpp файл:

```
#include <windows.h>
#include <stdlib.h>
#include <string.h>
#include <tchar.h>

// Global variables
// The main window class name.
static TCHAR szWindowClass[] = _T("win32app");

// The string that appears in the application's title bar.
static TCHAR szTitle[] = _T("Win32 Guided Tour Application");
HINSTANCE hInst;

// Forward declarations of functions included in this code module:
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

// ОСНОВНАЯ ФУНКЦИЯ ПРИЛОЖЕНИЯ - АНАЛОГ MAIN () для С
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine,
int nCmdShow)
{
	WNDCLASSEX wcex;
	wcex.cbSize		= sizeof(WNDCLASSEX);
	wcex.style		= CS_HREDRAW | CS_VREDRAW;
	wcex.lpfnWndProc	= WndProc;
	wcex.cbClsExtra		= 0;
	wcex.cbWndExtra		= 0;
	wcex.hInstance		= hInstance;
	wcex.hIcon		= LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_APPLICATION));
	wcex.hCursor		= LoadCursor(NULL, IDC_ARROW);
	wcex.hbrBackground	= (HBRUSH)(COLOR_WINDOW+1);
	wcex.lpszMenuName	= NULL;
	wcex.lpszClassName	= szWindowClass;
	wcex.hIconSm		= LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_APPLICATION));
	if (!RegisterClassEx(&wcex))
		{
		MessageBox(NULL, _T("Call to RegisterClassEx failed!"), _T("Win32
Guided Tour"), NULL);
		return 1;
		}
	hInst = hInstance;

	// Store instance handle in our global variable
	// The parameters to CreateWindow explained:
	// szWindowClass: the name of the application
	// szTitle: the text that appears in the title bar
	// WS_OVERLAPPEDWINDOW: the type of window to create
	// CW_USEDEFAULT, CW_USEDEFAULT: initial position (x, y)
	// 500, 100: initial size (width, length)
	// NULL: the parent of this window
	// NULL: this application does not have a menu bar
	// hInstance: the first parameter from WinMain
	// NULL: not used in this application


	HWND hWnd = CreateWindow( szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT, 500, 200, NULL, NULL, hInstance, NULL );
	if (!hWnd)  //Выполняется, если ошибка при  создании окна
		{
```

```
            MessageBox(NULL, _T("Call to CreateWindow failed!"), _T("Win32 Guided
Tour"), NULL);
            return 1;
            }
      // The parameters to ShowWindow explained:
      // hWnd: the value returned from CreateWindow
      // nCmdShow: the fourth parameter from WinMain

      ShowWindow(hWnd, nCmdShow);
      UpdateWindow(hWnd);

      // Main message loop:
      MSG msg;

      HDC hdc = GetDC(hWnd);

      while (GetMessage(&msg, NULL, 0, 0))
            {
                  TranslateMessage(&msg);
                  DispatchMessage(&msg);

            // * * * *  ЗДЕСЬ НАЧИНАЕМ ПИСАТЬ КОД ЛАБОРАТОРНОЙ РАБОТЫ * * * *

            // * * * *  ЗДЕСЬ КОНЕЦ КОДА ЛАБ РАБОТ * * * *
            }
      return (int)

      msg.wParam;
}

// **** ФУНКЦИЯ ОКОННОЙ ПРОЦЕДУРЫ  ****** //
// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
// PURPOSE:  Processes messages for the main window.
// WM_PAINT: Paint the main window
// WM_DESTROY: Post a quit message and return
//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
      {
            PAINTSTRUCT ps;
            HDC hdc;
            TCHAR greeting[] = _T("ПІБ");
            switch (message)
                  {
                        case WM_PAINT:
                              hdc = BeginPaint(hWnd, &ps);
                              // Here your application is laid out.
                              // For this introduction, we just print out
"Hello, World!"
                              // in the top left corner.
                              TextOut(hdc, 50, 50, greeting,
_tcslen(greeting));
                              // End application-specific layout section.
                              EndPaint(hWnd, &ps);
                              break;

                        case WM_DESTROY:
                              PostQuitMessage(0);
                              break;
                        default:
                              return DefWindowProc(hWnd, message, wParam,
lParam);
                              break;
                  }
```

```
        return 0;
}
```

10. Запустити за допомогою поєднання клавіш “CTRL+F5”
11. Відкриється наступне вікно (рисунок 2.4)

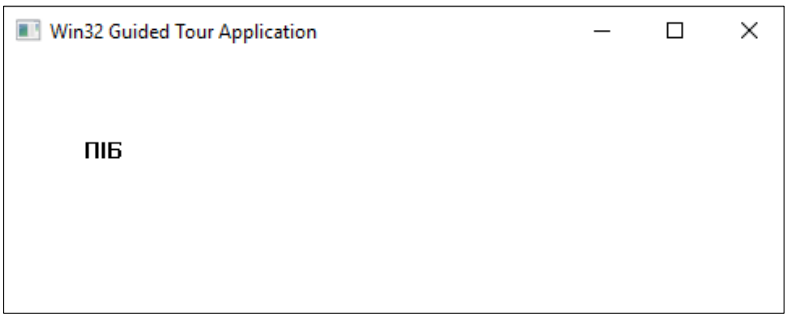

Рисунок 2.4 — Вікно для роботи

12. Особисті рішення для лабораторних робіт виконувати в наданій області коду (рисунок 2.5)

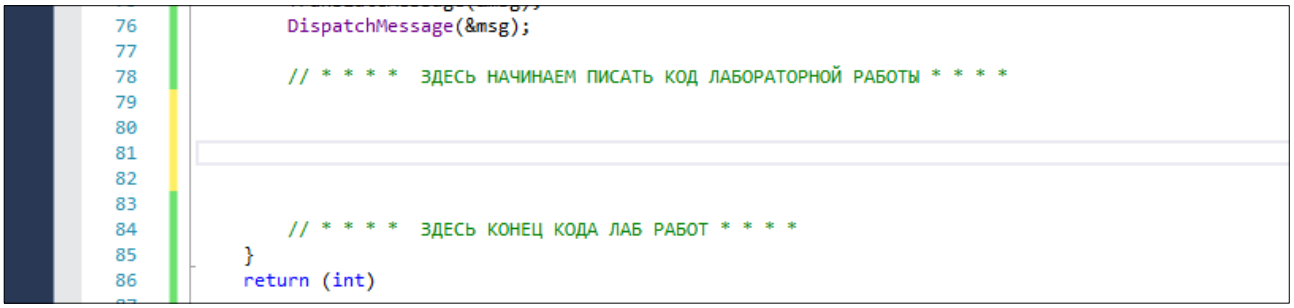

Рисунок 2.5 — Місце для створення рішень лабораторної роботи

3. Відтворення одиночного пікселя

Операційна система Windows надає програмісту можливості інтерфейсу графічних пристроїв (Graphics Device Interface, або скорочено GDI). З GDI тісно пов'язане поняття "Device Context" (DC) - контексту графічного пристрою - на чому виконується відтворення зображення. DC - універсальний пристрій виведення для якого можна використовувати однакові функції GDI

для екрану, принтера та інших пристроїв виведення. Програма користувача звертається до контексту пристрою через функції GDI.

Точка (піксель) базовий графічний об’єкт, за допомогою якого можливо малювання в вікні додатку. Для виводу в вікно поодинокого пікселя (точки) необхідно використовувати функцію GDI

```
SetPixel(HDC hdc, int X, int Y, COLORREF color)
```

hdc - хендел контексту пристрою (в нашому випадку -вікна), до якого виконується вивід пікселя,

X, Y – координати пікселя в вікні,

Color – колір пікселя, який задається як сукупність трьох (**R** (червоний)**, G** (зелений)**, B** (синій)) базових кольорів.

В найпростішому випадку кожний базовий колір це є ціле від 0 до 255. Для задання кольору пікселя рекомендується використовувати макрос

```
#define RGB(r, g, b) ((COLORREF) (((BYTE)(r)
    | ((WORD) ((BYTE)(g)) <8 ))
    | (((DWORD) (BYTE) (P)) <16 )))
```

Наприклад, виклик:

```
SetPixel(hdc, 100, 100, RGB (255, 0, 0));
```

в вікні максимально яскравим красним кольором відображає піксель (точку) с координатами X = 100 , Y = 100.

Для перевірки правильності роботи створеного проекту додайте вище приведений виклик функції, створить програму та запустить її на виконання.

Перевірте правильність роботи програми, порівнявши результат з рисунком 3.1.

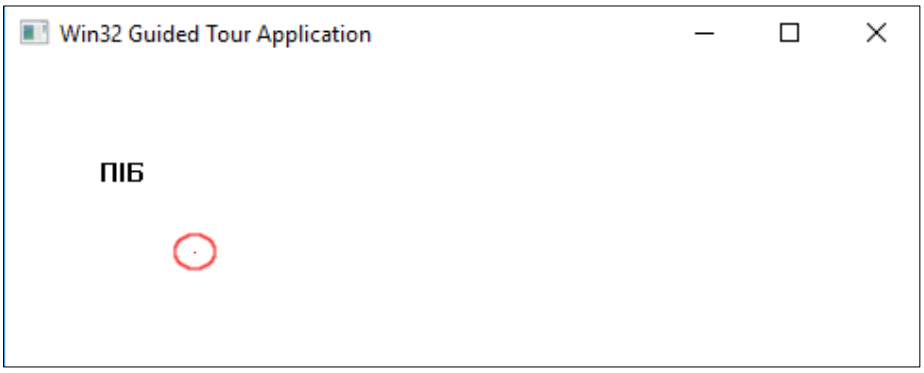

Рисунок 3.1 — Вивід одиночного пікселя

4. Індивідуальне завдання до лабораторної роботи

Написати код, який відображає на екрані 4 пікселя з вказаними координатами вказаного кольору.

Замість ПІБ у вікні вписати **Прізвище, ім'я, по-батькові, групу, номер варіанту.**

Варіант 1
Розмір вікна 700 X 700 пікселів

X	Y	R	G	B
100	100	200	180	32
300	80	80	230	100
250	250	240	0	0
80	200	40	40	230

Варіант 2
Розмір вікна 800 X 800 пікселів

X	Y	R	G	B
300	200	40	240	70
550	150	240	230	0
500	550	140	225	50
200	600	60	200	230

Варіант 3
Розмір вікна 900 X 900 пікселів

X	Y	R	G	B
150	760	185	16	62
360	350	70	156	38

30	670	210	12	19
430	320	45	40	159

Варіант 4
Розмір вікна 800 X 600 пікселів

X	Y	R	G	B
700	410	123	77	139
240	780	197	166	112
100	470	232	126	235
680	320	216	225	135

Варіант 5
Розмір вікна 512 X 384 пікселів

X	Y	R	G	B
365	300	230	45	212
465	30	160	47	110
95	240	214	88	108
465	105	34	224	165

Варіант 6
Розмір вікна 640 X 512 пікселів

X	Y	R	G	B
115	155	237	36	186
520	440	100	253	67
495	35	188	239	101
205	405	185	170	177

Варіант 7
Розмір вікна 640 X 480 пікселів

X	Y	R	G	B
570	365	219	17	162
330	220	97	35	102
315	425	176	104	169
110	355	7	61	33

Варіант 8
Розмір вікна 800 X 480 пікселів

X	Y	R	G	B
500	365	102	195	39

60	185	169	76	204
195	95	33	150	122
585	270	252	136	14

Студент вибирає варіант завдання відповідно до номеру студента в групі.

5. Звіт з виконання лабораторної роботи

Звіт з лабораторної роботи повинен містити:

- Вступну частину.
- Текст коду студента.
- Копію екрану з результатами виконання індивідуального завдання.
- Висновки.

Список використаних джерел

1. https://uk.wikipedia.org/wiki/Microsoft_Visual_Studio
2. https://www.visualstudio.com/
3. https://www.visualstudio.com/ru/license-terms/mlt558734/
4.

ЛАБОРАТОРНА РОБОТА №2. СИСТЕМИ КООРДИНТ ТА ГЕОМЕТРИЧНІ ТРАНСФОРМАЦІЇ

Мета лабораторної роботи: Ознайомитися з системами координат, що використовуються в комп'ютерної графіці, відновити знання з геометричними перетворюваннями на площині. Опрацювати набуті знання створивши програму в середовищі VisualStudio перетворення систем координат, включаючи поворот на довільний кут.

1. Системи координат

В сучасній комп'ютерній графіці для побудови графічних образів використовуються наступні системи координат:

- Об'єктна (локальна) система координат (3D).
- Світова система координат (3D)
- Система координат спостерігач (3D)
- Нормована система координат (2D)
- Система координат порту виводу (2D)
- Фізична система координат пристрою графічного виводу (2D).

Математична модель 2D графічного зображення (графічний образ) формується на **нормованій** картинній площині з прямокутною системою координат (рисунок 2.1). В стандартах комп'ютерної графіки прийнято, що координати всіх елементів (примітивів) графічного образу на картинній площині повинні бути в межах від -1.0 до +1.0. Всі елементи графічного образу, що виходять за межи картинної площини (мають координати більш ніж |1|) відкидаються і не відображаються. Одиниці виміру – безрозмірна величина (**float** або **double**).

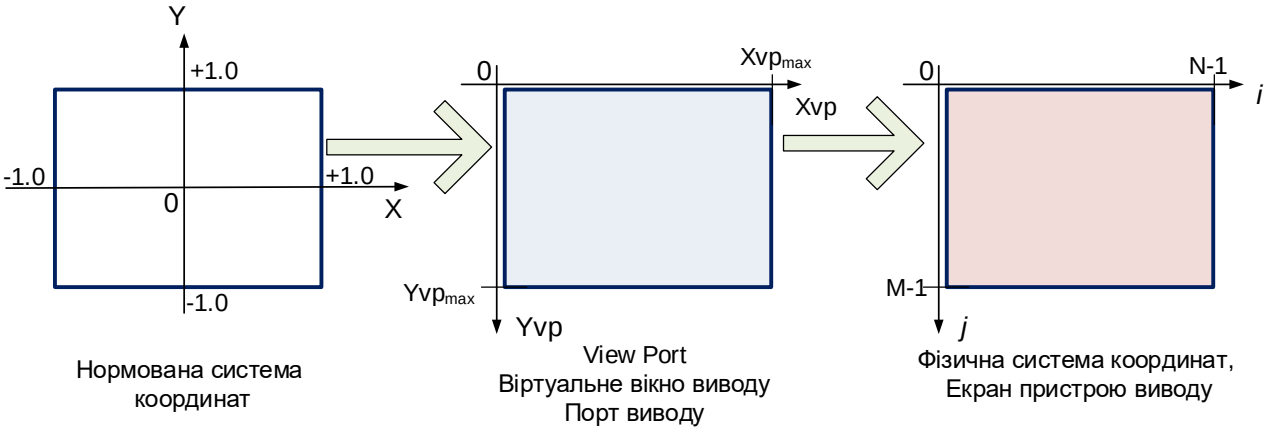

Рисунок 2.1. 2D системи координат графічного виводу

З іншого боку всі технічні пристрої графічного виводу мають власну **фізичну** систему координат. Початок фізичної системи координат - лівій верхній кут екрану. Вісь абсцис направлена зліва на право, вісь координат - з верху вниз. Одиниці виміру - пікселі екрану. Розмір фізичного екрану $N * M$ пікселів. Тобто **цілечисельні** (**int**) координати елементів в фізичній системі повинні бути в межах $0 \le i \le N-1 \quad 0 \le I \le M-1$.

Для узгодження нормованої системи координат з системами координат різноманітних пристроїв графічного виводу використовується спеціальне віртуальне вікно виводу – порт виводу або видовий порт (View Port). С портом виводу пов'язана спеціальна система координат. Її початок - лівій верхній кут порту виводу. Вісь абсцис направлена зліва на право, вісь координат - з верху вниз. Одиниці виміру - пікселі екрану. Розмір порту виводу повинен задає користувач за допомогою цілечисельних змінних Xvp_{max} , Yvp_{max} . Розмір порту виводу відповідно $Xvp_{max} * Yvp_{max}$ пікселів. Позначимо ширину порту виводу як **Width** та висоту порту виводу як **Height.**

Важливо розуміти, що перехід від системи координат порту виводу в фізичну систему координат пристрою виводу виконує операційна система за допомогою вбудованих програм – драйверів відповідного пристрою графічного виводу. Проте, перетворення нормованої системи координат до координат порту виводу повинна виконати програма користувача (рисунок 2.2).

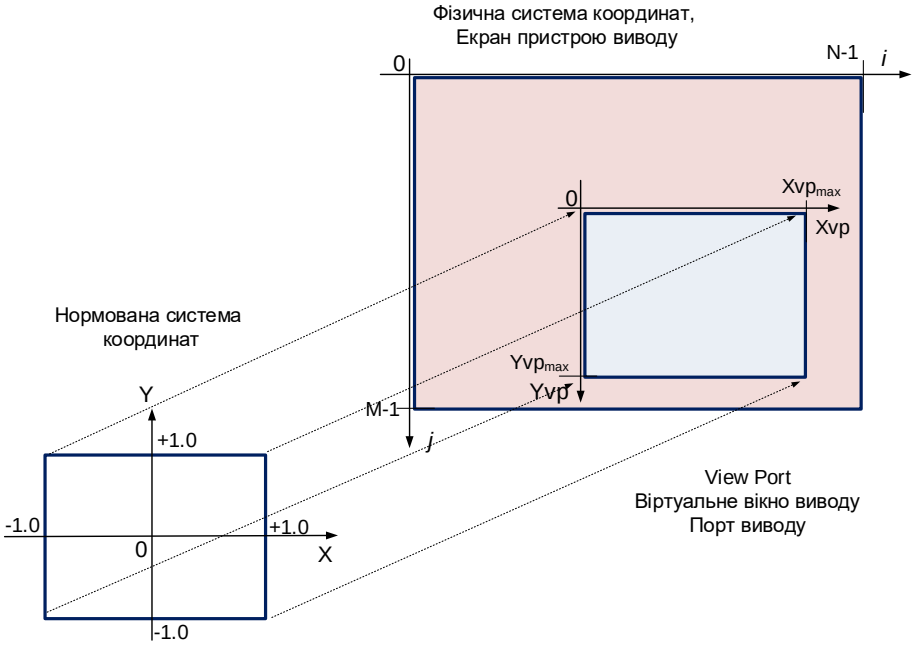

Рисунок 2.2. Перетворення 2D систем координат при графічному виводі

Позначимо координати деякої точки $\mathbf{Pnd} = \begin{bmatrix} Pndx \\ Pndy \end{bmatrix}$ в нормованій системі координат та координати відповідної точки $\mathbf{Pvp} = \begin{bmatrix} Pvpx \\ Pvpy \end{bmatrix}$ в системі координат порту виводу. Тоді перетворення виконується відповідно до співвідношення

$$\mathbf{Pvp} = \begin{bmatrix} Pvpx \\ Pvpy \end{bmatrix} = \begin{bmatrix} (Pndx + 1.0) * \dfrac{\boldsymbol{width}}{2} + Xvpmin \\ -(Pndy - 1.0) * \dfrac{\boldsymbol{height}}{2} + Yvpmin \end{bmatrix}$$

2. Типові геометричні перетворення на площини

Геометричні об'єкти як на площині так і в просторі можна піддавати ряду різних перетворень. Найбільш вживаними в задачах комп'ютерної графіки є:

• переміщення (паралельний перенос);

• зміна розмірів (масштабування);

• повороти навколо деякої точки на площині або деякої осі в просторі (обертання).

Розглянемо типові геометричні перетворення на площині, або 2D перетворення. В загальному випадку будемо вважати в прямокутній декартовій системі координат задана довільна точка **P** своїми координатами $\begin{bmatrix} Px \\ Py \end{bmatrix}$ (рисунок 2.3).

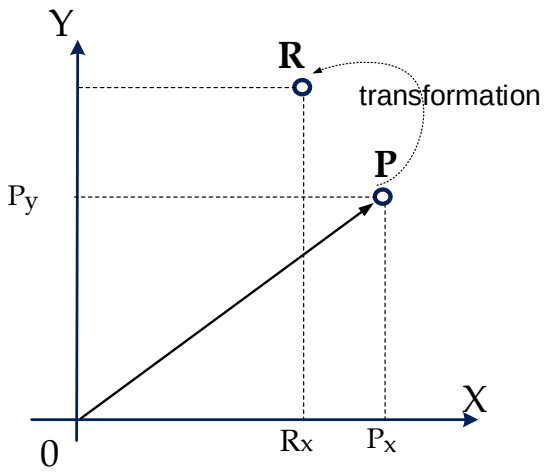

Рисунок 2.3. Загальний вигляд геометричного перетворення на площині

Нову (трансформовану, перетворену) точку позначимо як **R** з координатами $\begin{bmatrix} Rx \\ Ry \end{bmatrix}$.

2.1. Паралельний перенос

При паралельному переносі точка **P** зміщується на відстань ***d*** в напрямку, який задано нормованим вектором **V** з координатами $\begin{bmatrix} Vx \\ Vy \end{bmatrix}$ (рисунок 2.4) . Нормований вектор, це вектор, який має одиничну довжину, тобто $\sqrt{V_x^2 + V_y^2} = 1$.

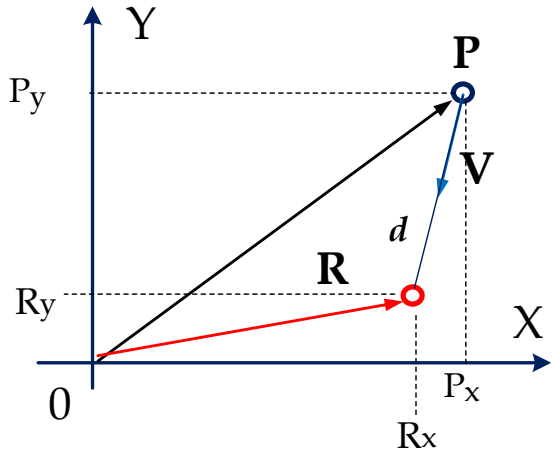

Рисунок 2.4 Трансформація паралельного переносу

З урахуванням нормованості вектору переносу трансформована точка може бути знайдена як

$$\begin{bmatrix} Rx \\ Ry \end{bmatrix} = \begin{bmatrix} Px \\ Py \end{bmatrix} + \boldsymbol{d} * \begin{bmatrix} Vx \\ Vy \end{bmatrix} \quad .$$

2.2. Масштабування

Масштабування з коефіцієнтом $\boldsymbol{k}$ використовується для збільшення ($\boldsymbol{k}$ >1) або зменшення (0< $\boldsymbol{k}$ <*1*) зображення. Для цього необхідно радіус вектори всіх точок зображення збільшити (зменшити) в $\boldsymbol{k}$ раз. Тобто

$$\begin{bmatrix} Rx \\ Ry \end{bmatrix} = \boldsymbol{k} * \begin{bmatrix} Px \\ Py \end{bmatrix} \quad .$$

2.3. Поворот навколо начала координат

При обертанні на площині точка переміщується по дузі кола, центр якого знаходиться в початку координат на кут $\boldsymbol{\alpha}$ (рисунок 2. 5). При цьому довжина радіус вектору точки не змінюється.

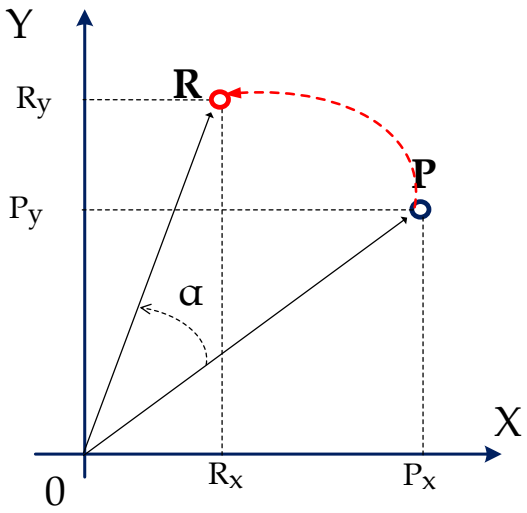

Рисунок 2.5 Трансформація повороту

Координати точки **R** можна знайти як

$$\begin{bmatrix} Rx \\ Ry \end{bmatrix} = \begin{bmatrix} Px * \cos(\alpha) - Py * \sin(\alpha) \\ Px * \sin(\alpha) + Py * cos(\alpha) \end{bmatrix}$$

3. Приклад коду

Нижче наведено фрагмент коду, в якому в нормованій системі координат задано дві точки: центральна з нульовими координатами, вхідна точка з довільними координатами та кут повороту (alfa). В нормованій системі координат виконується поворот точки на заданий кут, перетворення центральної точки, вхідної точки та повернутої точки до системи координат порту виводу та вивід на екран дисплею. Результат роботи приведено на рисунку 3.1.

// Початок фрагменту коду

```
float Xns_Centr = 0.0, Yns_Centr = 0.0;
float Xns_P = 0.5, Xns_P_transf;
float Yns_P = 0.5, Yns_P_transf;
float alfa_grad, alfa_rad;
int Xvp_Center, Yvp_Center;
int Xvp_P, Yvp_P;
```

```
int Xvp_P_transf, Yvp_P_transf;

// Кут повороту
alfa_grad = 45.0;
alfa_rad = 3.14/180.0*alfa_grad;

// Поворот в нормованій системі координат
Xns_P_transf = Xns_P*cos(alfa_rad) - Yns_P*sin(alfa_rad);
Yns_P_transf = Xns_P*sin(alfa_rad) + Yns_P*cos(alfa_rad);

// Перетворення  центральної точки до ViewPort
Xvp_Center = ((Xns_Centr +1.0)*VP_width/2)+VP_start_X;
Yvp_Center = (-(Yns_Centr - 1.0)*VP_height / 2) + VP_start_Y;

// Перетворення  вхідної  точки до ViewPort
Xvp_P = ((Xns_P + 1.0)*VP_width / 2) + VP_start_X;
Yvp_P = (-(Yns_P - 1.0)*VP_height / 2) + VP_start_Y;

// Перетворення  повернутої  точки до ViewPort
Xvp_P_transf = ((Xns_P_transf+ 1.0)*VP_width / 2) + VP_start_X;
Yvp_P_transf = (-(Yns_P_transf - 1.0)*VP_height / 2) + VP_start_Y;

// Вивід  точок
SetPixel(hdc, Xvp_Center, Yvp_Center, RGB(0, 0, 0));
SetPixel(hdc, Xvp_P, Yvp_P, RGB(0, 0, 0));
SetPixel(hdc, Xvp_P_transf, Yvp_P_transf, RGB(0, 0, 0));
// Кінець фрагменту коду
```

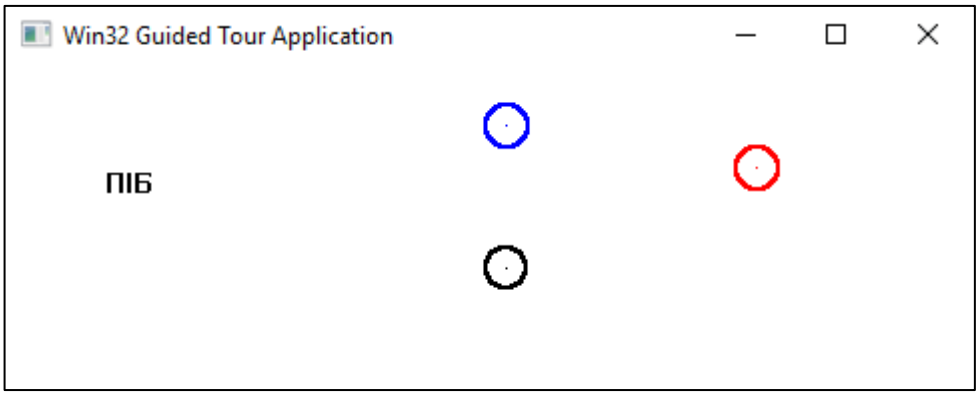

Рисунок 3.1 – Результат роботи програми

4. Завдання

Написати код, який відображає на екрані 7 точок (пікселя):

- Центральна (колір **Black**).
- Вхідна точка (колір **Red**).
- Повернута на кут ***alfa*** точка (колір **Blue**).
- Чотири точки, які показують кути вікна та пряму, яка з'єднує дві кутових точки(колір **Red**).

У вікні вписати **Прізвище, ім'я, по-батькові, групу, номер варіанту.**

Студент вибирає варіант завдання відповідно до номеру студента в групі.

Варіант	Pndx	Pndy	*Alfa (град)*	Xvp_{max}	Yvp_{max}
1	-0,25	0,7	30	350	350
2	0,5	-0,4	-45	400	400
3	0,7	0,5	60	450	450
4	-0,7	-0,25	-75	400	300
5	0,25	-0,75	90	512	384
6	-0,5	0,75	-105	640	512
7	-0,75	-0,5	120	640	480
8	1	0,25	-135	800	480

5. Звіт з виконання лабораторної роботи

Звіт з лабораторної роботи повинен містити:

- Вступну частину
- Текст коду студента, який включає в собі дві функції (функція створення вхідної точки та функція повороту);
- Копію екрану з результатами виконання індивідуального завдання (обвести відповідним кольором)
- Висновки

Список використаних джерел

1. Комп’ютерна графіка. Курс лекцій. Вінниця, 2016 р.

2. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.

3. Вельтмандер П.В. Машинная графика: Учебное пособие в 3-х книгах. Книга 2. Основные алгоритмы / Новосиб. Ун-т. Новосибирск, 1997.- 193 с.

ЛАБОРАТОРНА РОБОТА № 3. ГНЕРАЦІЯ ВІДРІЗКУ ПРЯМОЇ КАРКАСНЕ ПРЕДСТАВЛЕННЯ ПОЛІГОНУ

Мета лабораторної роботи: Ознайомитися з алгоритмами генерації відрізків прямих для плоских зображень. Опрацювати набуті знання створивши в середовищі VisualStudio процедуру генерації відрізку прямою за алгоритмом Брезенхема та вивести на екран каркасне представлення трикутного полігону.

1. Алгоритм Брезенхема генерації відрізку прямої

Алгоритм Брезенхема (Bresenham) — алгоритм, який визначає які точки в 2-вимірному растрі мають бути накреслені для формування близького наближення для прямої лінії між двома заданими точками. Він загально використовується для малювання ліній на екрані через те, що використовує тільки цілочисельні операції суми, віднімання та бітові операції. Всі ці операції дуже «дешеві» в стандартній архітектурі комп'ютерів. Незначно розширений початковий алгоритм Брезенхема також використовується для малювання кіл та еліпсів.

Хоча є інші алгоритми, що також використовується в сучасній комп'ютерній графіці (наприклад, алгоритм Ву через підтримку згладжування [ ]), алгоритм Брезенхема залишається вживаним завдяки його швидкості і простоті. Через свою простоту алгоритм часто реалізується або як вбудована програма або як апаратне забезпечення сучасних відеокарт. Також його можна знайти в багатьох програмних графічних бібліотеках. Ім’я "Брезенхем" використовується сьогодні для низки алгоритмів, що розширюють або модифікують базовий алгоритм.

Суть алгоритму полягає в наступному. Необхідно знайти растрове уявлення прямолінійного відрізку, заданого початковою $\boldsymbol{S}(x_S,y_S)$ та кінцевою $\boldsymbol{E}(x_E,y_E)$ вершинами. Природньо, для розрахунків використовуються

цілочисельні координати (центри пікселів), рисунок 3.1. Тобто початкова та кінцева вершини співпадають з центрами пікселов: $\boldsymbol{S} = (i_S, j_S)$, $\boldsymbol{E} = (i_E, j_E)$ та, звісно $\boldsymbol{S} \neq \boldsymbol{E}$. Також, для спрощення, будемо розглядати відрізок з кутом нахилу менш 45^o , $|i_E - i_S| > |j_E - j_S| > 0$.

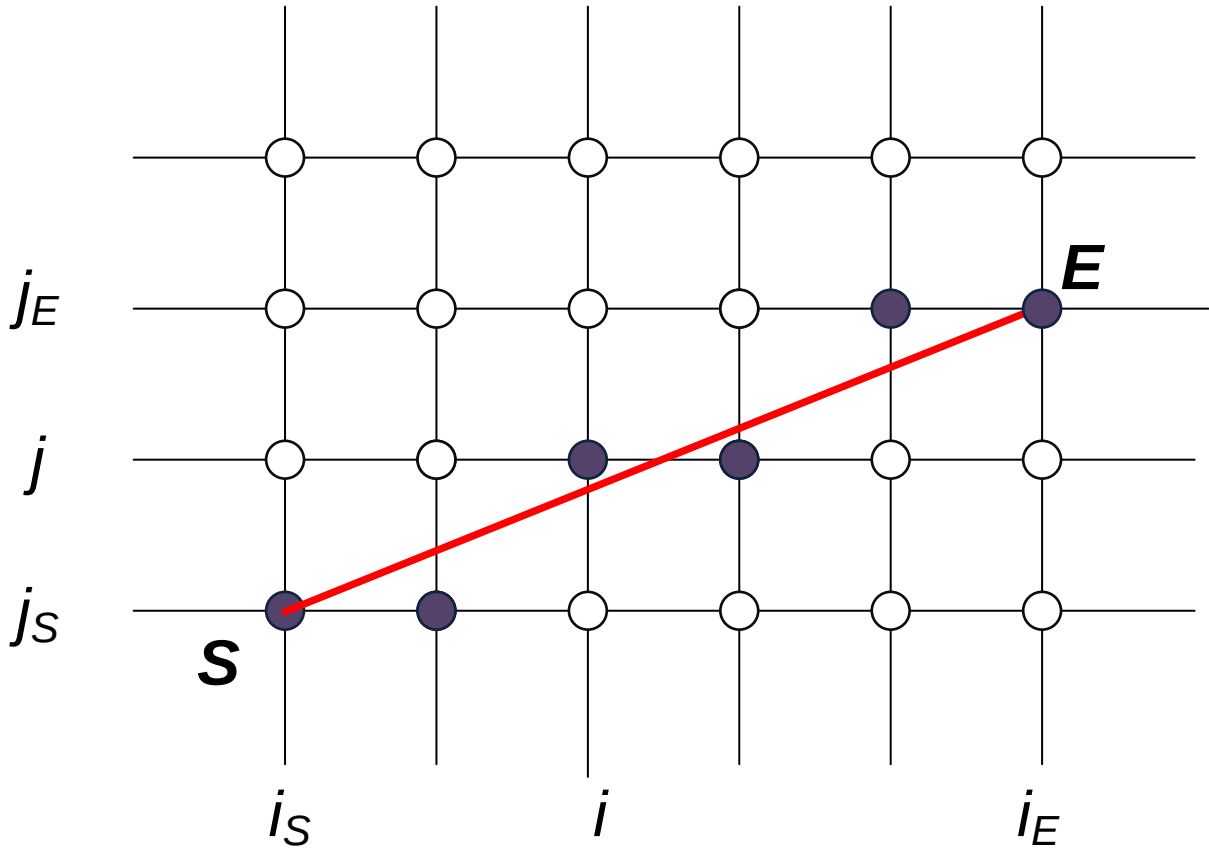

Рисунок 3.1. Растрове наближення відрізку прямої.

Зрозуміло, що у цьому випадку, цілочисельна координата по координатній висі $\boldsymbol{X}$ має змінюватись від $\boldsymbol{i}_S$ до $\boldsymbol{i}_E$ з шагом +1. Алгоритм Брезенхема визначає, як знайти координату $\boldsymbol{j}$ для любого проміжного $\boldsymbol{i}$.

Для цього алгоритм Брезенхема обирає наступне ціле $\boldsymbol{j}$ таким чином, щоб центр обраного пікселя був найближчим до ідеального (дробового) $\boldsymbol{y}$ для того ж $\boldsymbol{x} = \boldsymbol{i}$. Тобто $\boldsymbol{j}$ може залишитися тим самим, тобто $\boldsymbol{j}_i = \boldsymbol{j}_{i-1}$, або бути збільшеним на одиницю $\boldsymbol{j}_i = \boldsymbol{j}_{i-1}+1$. Відомо, що загальне рівняння лінії через дві точки таке:

$$y = \frac{y_E - y_S}{x_E - x_S}(x_E - x_S) + y_S \text{ або } \quad y = kx + b$$

де $k = \frac{y_E - y_S}{x_E - x_S}$ кутовий коефіцієнт, $b = y_S - k * x_S$ постійна константа.

Важно, що кутовий коефіцієнт $k < 1$ залежить тільки від координат точок ***S*** та ***E*** і може бути обчислений заздалегідь.

Найдемо Δy для $\Delta x = x_i - x_{i-1}$

$$\Delta y = y_{(x=i)} - y_{(x=i-1)} = k * i + b - k * (i - 1) - b = k.$$

Таким чином, на кожному кроці по ***i*** точне значення ***y*** повинно збільшуватись на постійне значення ***k***.

Для вибору координати j_i наступного пікселю необхідно оцінити значення похибки $Err = y_{(x=i)} - j = C_{i-1} + \Delta y$, де C_{i-1} відхилення попередньо обраного пікселю від прямої. Якщо $Err < 0.5$ необхідно вибрати $j_i = j_{i-1}$. Якщо $Err \geq 0.5$ необхідно вибрати $j_i = j_{i-1}+1$ (дивись рисунок 3.2).

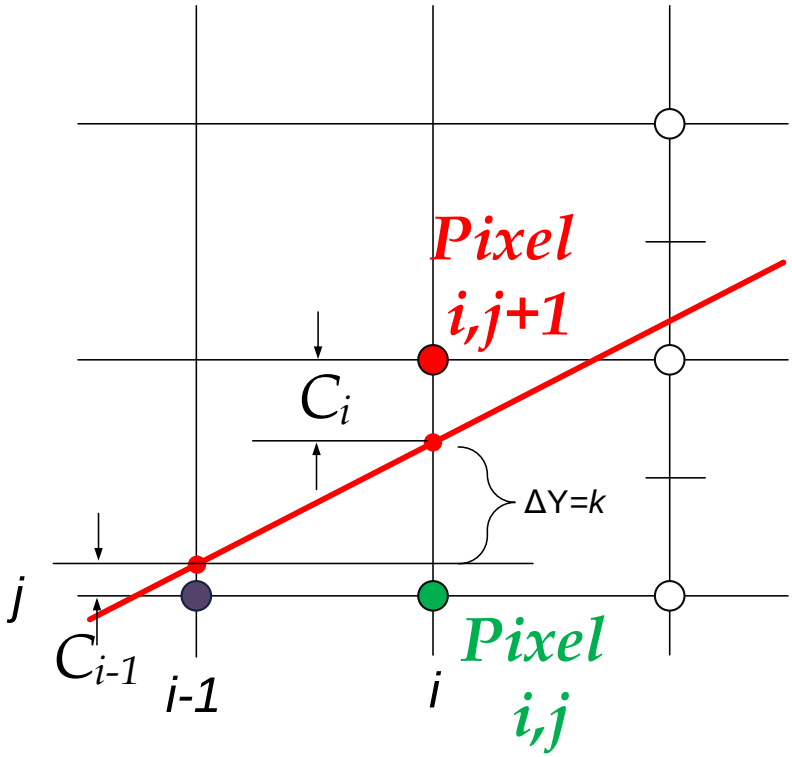

Рисунок 3.2. До вибору наступного пікселя алгоритмом Брезенхема.

З другого боку для S значення $C_{iS} = 0$, тобто наступне $C_{iS+1} = \Delta y = k$. Також ясно, що якщо вибрано $j_i = j_{i-1}$, то $C_i = C_{i-1} + k$. Якщо вибрано $j_i = j_{i-1} + 1$, то $C_i = C_{i-1} + k - 1$.

Далі наведено найпростіший варіант алгоритму Брезенхема для випадку $|i_E - i_S| > |j_E - j_S| > 0$. (на псевдомові).

```
function line(is, ie, js, je)
    int deltai := ie - is
    int deltaj := je - js
    real c := 0
    real k := deltaj / deltai
    int j := js
    for i from is to ie
       out_pixel (i,j)
       c := c + k
       if abs(c) ≥ 0.5 then
          j := j + 1
          c := c - 1.0
```

3. Приклад коду

Один з варіантів «розширеного» алгоритму Брезенхема для відображення відрізку в любому квадранті .

```
// * * * *  ЭТО ОТРИСОВКА ПРЯМОЙ АЛГОРИТМОМ БРЕЗЕНХЕМА * * * *
int X1 = 300; // НАЧАЛЬНАЯ ТОЧКА  X
int Y1 = 300; // НАЧАЛЬНАЯ ТОЧКА  Y
int X2 = 400; // КОНЕЧНАЯ ТОЧКА   X
int Y2 = 400; // КОНЕЧНАЯ ТОЧКА   Y
int deltaX = abs(X2 - X1);  // ПРИРАЩЕНИЕ ПО  X
int deltaY = abs(Y2 - Y1);  // ПРИРАЩЕНИЕ ПО  Y
int signX = X1 < X2 ? 1 : -1;   // НАПРАВЛЕНИЕ СМЕЩЕНИЯ ПО  X
int signY = Y1 < Y2 ? 1 : -1;   // НАПРАВЛЕНИЕ СМЕЩЕНИЯ ПО  Y
int Err = deltaX - deltaY;      // ОШИБКА

SetPixel(hdc, X2, Y2, RGB(0, 0, 255));   // ВЫВОДИМ КОНЕЧНУЮ ТОЧКУ

while (X1 != X2 || Y1 != Y2)   // ПОКА ТОЧКИ НЕ СОВПАДАЮТ
     {
```

```
SetPixel(hdc, X1, Y1, RGB(0, 0, 255)); // ВЫВОДИМ ТЕКУЩУЮ ТОЧКУ
int Err2 = Err * 2;

if (Err2 > -deltaY)
      {
      Err -= deltaY;
      X1 += signX;
      }
if (Err2 < deltaX)
      {
      Err += deltaX;
      Y1 += signY;
      }
}
```

4. Індивідуальне завдання до лабораторної роботи

Завдання 1.

Оформити алгоритм Брезенхема у вигляді функції та відобразити на екрані чотирикутник, який задано чотирма вершинами (розмір вікна та координати точок співпадають з даними першою лабораторною роботи). Ребра відобразити заданим кольором.

Замість ПІБ у вікні вписати **Прізвище, ім'я, по-батькові, групу, номер варіанту.**

Варіант 1

Розмір вікна 700 X 700 пікселів. Колір - **червоний**.

Вершина	X	Y
1	100	100
2	300	80
3	250	250
4	80	200

Варіант 2

Розмір вікна 800 X 800 пікселів. Колір - **помаранчевий**.

Вершина	X	Y
1	300	200
2	550	150
3	500	550
4	200	600

Варіант 3

Розмір вікна 900 X 900 пікселів. Колір - **жовтий**.

Вершина	X	Y
1	150	760
2	360	350
3	30	670
4	430	320

Варіант 4

Розмір вікна 800 X 600 пікселів. Колір - **зелений**.

Вершина	X	Y
1	700	410
2	240	780
3	100	470
4	680	320

Варіант 5

Розмір вікна 512 X 384 пікселів. Колір - **голубий**.

Вершина	X	Y
1	365	300
2	465	30
3	95	240
4	465	105

Варіант 6

Розмір вікна 640 X 512 пікселів. Колір - **синій**.

Вершина	X	Y
1	115	155
2	520	440
3	495	35
4	205	405

Варіант 7

Розмір вікна 640 X 480 пікселів. Колір - **фіолетовий**.

Вершина	X	Y
1	570	365
2	330	220
3	315	425

4	110	355

Варіант 8

Розмір вікна 800 X 480 пікселів. Колір - **чорний**.

Вершина	X	Y
1	500	365
2	60	185
3	195	95
4	585	270

Завдання 2 (для вищої оцінки)

В нормованій системі координат задано три точки. Використовуючи функцію з лабораторної роботи №2, перетворити точки до ViewPort і з допомогою створеної функції алгоритму Брезенхема відобразити трикутник. За допомогою алгоритма Брезенхема створити лінії системи координат (чорний колір).

Варіант 1.

Розмір вікна Xvp_{max} = 800 піх, Yvp_{max} = 480 піх. Колір - **червоний**.

Вершина	X	Y
1	0,5	0,5
2	0,0	-0,8
3	-0,4	+0,4

Варіант 2.

Розмір вікна Xvp_{max} = 640 піх, Yvp_{max} = 480 піх. Колір - **червоний**.

Вершина	X	Y
1	0,3	0,2
2	0,5	-0,1
3	-0,5	+0,5

Варіант 3.

Розмір вікна Xvp_{max} = 640 піх, Yvp_{max} = 512 піх. Колір - **червоний**.

Вершина	X	Y
1	0,1	0,7
2	0,3	-0,3

3	-0,6	0,0

Варіант 4.

Розмір вікна Xvp_{max} = 512 піх, Yvp_{max} = 384 піх. Колір - **червоний**.

Вершина	X	Y
1	0,7	0,4
2	0,2	-0,7
3	-0,1	+0,4

Варіант 5.

Розмір вікна Xvp_{max} = 800 піх, Yvp_{max} = 600 піх. Колір - **червоний**.

Вершина	X	Y
1	-0,3	0,3
2	0,4	0,0
3	-0,5	-0,5

Варіант 6.

Розмір вікна Xvp_{max} = 900 піх, Yvp_{max} = 900 піх. Колір - **червоний**.

Вершина	X	Y
1	0,1	0,1
2	0,5	-0,4
3	-0,4	0,0

Варіант 7.

Розмір вікна Xvp_{max} = 800 піх, Yvp_{max} = 800 піх. Колір - **червоний**.

Вершина	X	Y
1	0,5	0,3
2	0,3	-0,2
3	-0,3	0,4

Варіант 8.

Розмір вікна Xvp_{max} = 700 піх, Yvp_{max} = 700 піх. Колір - **червоний**.

Вершина	X	Y
1	0,5	0,5
2	0,0	-0,1
3	-0,1	0,0

Студент вибирає варіант завдання відповідно до номеру студента в групі.

5. Звіт з виконання лабораторної роботи

Звіт з лабораторної роботи повинен містити:

- Вступну частину
- Текст коду студента
- Копію екрану з результатами виконання індивідуального завдання
- Висновки

Список використаних джерел

1. Комп’ютерна графіка. Курс лекцій. Вінниця, 2016 р.
2. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.
3. Вельтмандер П.В. Машинная графика: Учебное пособие в 3-х книгах. Книга 2. Основные алгоритмы / Новосиб. Ун-т. Новосибирск, 1997.- 193 с.
4. https://uk.wikipedia.org/wiki/Алгоритм_Брезенхейма
5. https://ru.wikipedia.org/wiki/Список_цветов_в_X11

ЛАБОРАТОРНА РОБОТА № 4. ЗАФАРБУВАННЯ ПЛОСКИХ ПОЛІГОНІВ

Мета лабораторної роботи: Ознайомитися з системами координат, що використовуються в компютерной графиці, відновити знання з геометричними переторюваннями на площині. Опрацювати набуті знання створивши програму в середовищи VisualStudio, яка зафарбовує плоский полігон.

1. Задача зафарбування плоского полігону.

При опису об’єктів відображення як множини плоских полігонів виникає типова задача зафарбування відповідних областей екрану, обмежених деякою замкнутою лінією (рис. 4.1). Типовими є області обмежені довільною замкнутою лінією (рис. 4.1.б), та області у вигляді випуклих многокутників, у найпростішому вигляді опуклого трикутника (рис. 4.1.б). З математичної точки зору задача полягає в пошуку всіх пікселів екрану які лежать усередині області. В лабораторній роботі розглядається випадок опуклого трикутника, який задається координатами його трьох вершин. Звісно, кожна вершина не повинна співпадати ні з якою іншою.

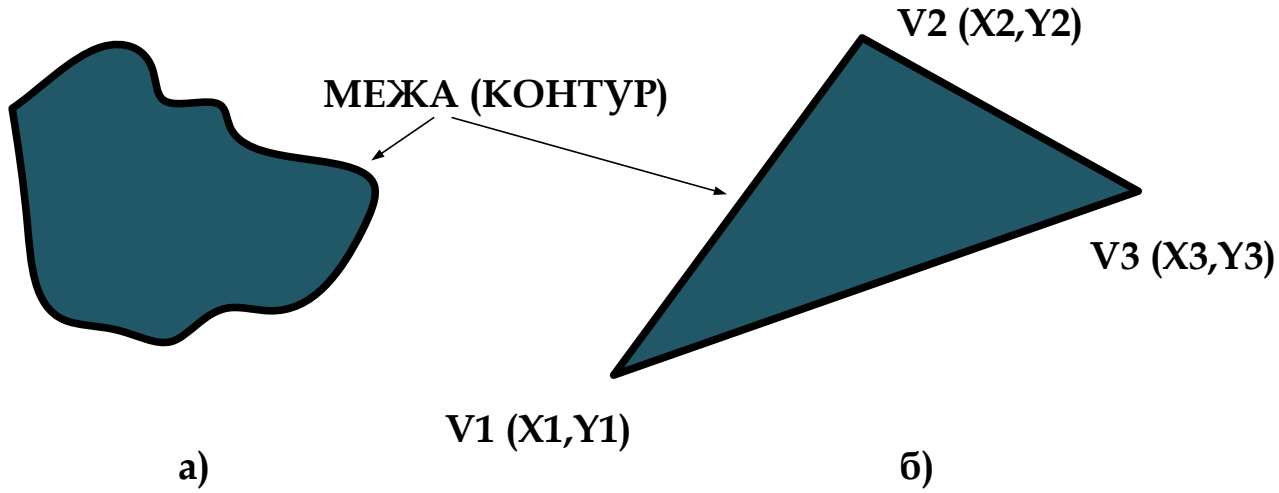

Рисунок 5.1. До задачі зафарбування полігонів

Для вирішення цієї спрощеної задачі застосовуються два основних підходи: затравочне заповнення і растрова розгортка. Методи першого типу виходять з того, що задана деяка точка (запал) всередині контуру полігону і заданий деякий критерій приналежності точки кордону області. В алгоритмах шукають точки, сусідні з затравочною і розташовані всередині контуру. Якщо виявлена сусідня точка, що належить внутрішній області контуру, то вона стає затравочною і пошук триває рекурсивно.

Методи растрової розгортки засновані на скануванні рядків растра й визначенні, чи лежить точка всередині заданого контуру області. Сканування здійснюється найчастіше "зверху вниз", а алгоритм визначення приналежності точки заданої області залежить від виду її кордону (межі).

2. Алгоритм растрової розгортки зафарбування полігону.

В загальному вигляді алгоритм полягає в наступному. Нехай контур трикутника задається координатами трьох вершин (рисю 4.2).

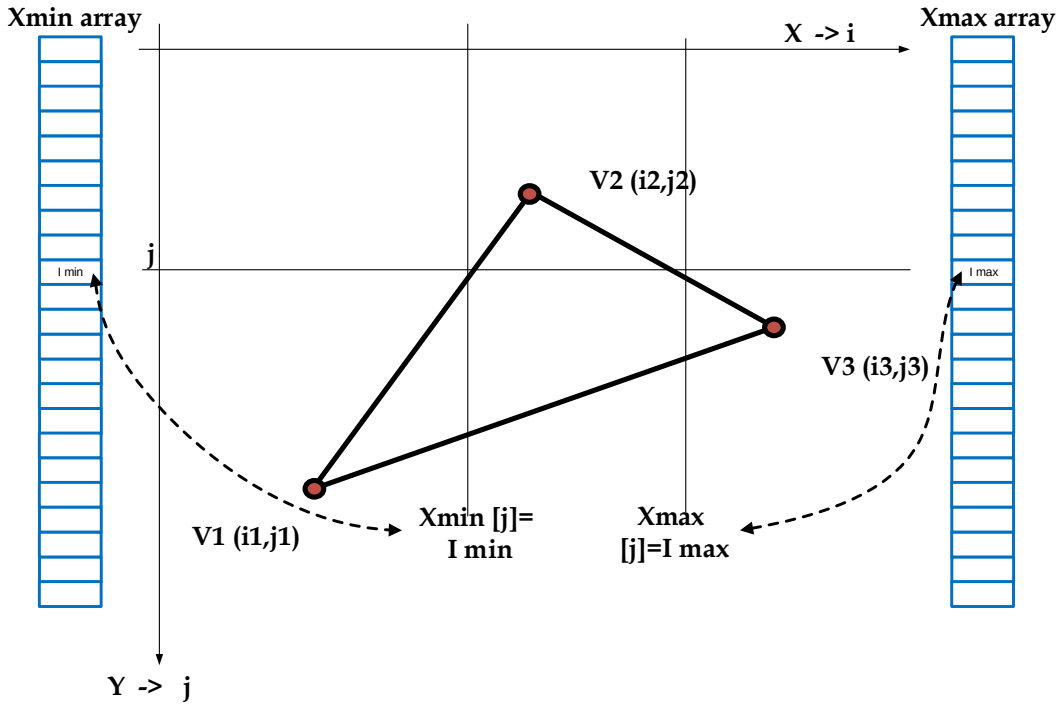

Рисунок 4.2 До алгоритму растрової розгортки

Побудова починається з ініціалізації цілих масивів кордонів полігону Xmin [*M*] і Xmax [*M*], де M – розмір вікна по вертикалі (пікселів). Масив лівого

кордону Xmin [*M*] заповнюється деяким цілим від'ємним числом, наприклад -1, а масив правого кордону Xmax [*M*] – числом **N**, що дорівнює кількості пікселів растра по горизонталі. Далі, використовуючи послідовно для кожної пари вершин (3-х ребер трикутника) алгоритм Брезенхема, обчислюються та записуються координати кордону 3-х відрізків в масиви Xmin [*M*] і Xmax [*M*]. Можна модифікувати алгоритм з лабораторної роботи 3, при цьому не виводити пікселі на екран, а записувати значення координати ***i*** поточного пікселя відрізку до Xmin [***j***] або Xmax [***j***] порівнюючи поточне значення *i* з поточними значеннями Xmin [***j***] та Xmax [***jj***]:

- якщо Xmin [***j***] == -1 та Xmax [***j***] == **N**, значення ***i*** записуємо в кожний масив (Xmin [***j***] = ***i*** **,** Xmax [***j***] = ***i*** ;
- якщо ***i*** < Xmin [***j***], значення ***i*** записуємо в масив Xmin [***j***] = ***i*** ;
- якщо ***i*** > Xmax [***j***], значення ***i*** записуємо в масив Xmax [***j***] = ***i*** ;

Після закінчення генерації всіх сторін трикутника для кожної строк з індексом ***j*** маємо наступні варіанти. Якщо Xmin[***j***] = -1 всі пікселі ***j***-ї строки растру не належать трикутнику. Якщо Xmin[***j***] ≥0, то пікселі *j*-ї строки, починаючи з I_min = Xmin [***j***] до I_max = Xmax [***j***] належать трикутнику і їх треба відобразити в вікні с заданим кольором.

Алгоритм легко поширюється на випадок довільного опуклого багатокутника.

3. Приклад коду зафарбування

Наведено основний код алгоритму растрової розгортки зафарбування трикутника з використанням модифікованої функції Брезенхема для генерації ребер.

```
// ОБЯВЛЕНИЕ ФУНКЦИИ БРЕЗЕНХЕМА
int Brezenhem (int, int, int, int, int arr1[], int arr2[], int);
// первый аргумент – координата X первой вершины
// второй аргумент – координата Y первой вершин
```

```
// третий аргумент – координата X второй вершины
// четвертый аргумент – координата Y второй вершин
// пятый аргумент – указатель на массив X_min[ ]
// шестой аргумент – указатель на массив X_max[ ]
// пятый аргумент – размер порта по горизонтали


// * * * *  ЗДЕСЬ НАЧИНАЕМ ПИСАТЬ КОД ЛАБОРАТОРНОЙ РАБОТЫ * * * *

        int Xmin [win_size_Y];
        int Xmax [win_size_Y];
        int i, j;
        // Инициализация массивов
        for (j =0; j<win_size_Y; ++j)
                {
                Xmin[j] = -1;
                Xmax[j] = win_size_X;
                }

        SetPixel(hdc, win_start_X, win_start_Y, RGB(255, 0, 0));
        SetPixel(hdc, win_end_X, win_start_Y, RGB(255, 0, 0));
        SetPixel(hdc, win_start_X, win_end_Y, RGB(255, 0, 0));
        SetPixel(hdc, win_end_X, win_end_Y, RGB(255, 0, 0));

        // Вызовы Брезенхема
        Brezenhem( ,  , , , Xmin, Xmax, win_size_X); // Первое ребро
        Brezenhem( ,  , , , Xmin, Xmax, win_size_X); // Второе ребро
        Brezenhem( ,  , , , Xmin, Xmax, win_size_X); // Третье ребро
        // Отрисовка полигона
        for (j = 0; j<win_size_Y; ++j)
                {
                if (Xmin[j]>-1)
                        {
                        for (i = Xmin[j]; i <= Xmax[j]; ++i)
                                {
                                SetPixel(hdc, i, j, RGB(255, 0, 0));
                                }
                        }
                }
```

4. Завдання до лабораторної роботи.

Модифікувати функцію Брезенхема з лабораторної роботи №3 для заповнення масивів Xmin [*M*] і Xmax [*M*]. Координати точок многокутника з лабораторної роботи №3

Варіант 1. Колір ребра – **червоний**, колір зафарбування – **чорний**.

Варіант 2. Колір ребра – **помаранчевий**, колір зафарбування – **фіолетовий**.

Варіант 3. Колір ребра – **жовтий**, колір зафарбування – **синій**.

Варіант 4. Колір ребра – **зелений**, колір зафарбування – **голубий**.

Варіант 5. Колір ребра – **голубий**, колір зафарбування – **зелений**.

Варіант 6. Колір ребра – **синій**, колір зафарбування – **жовтий**.

Варіант 7. Колір ребра – **фіолетовий**, колір зафарбування – **помаранчевий**.

Варіант 8. Колір ребра – **чорний**, колір зафарбування – **червоний**.

5. Звіт з виконання лабораторної роботи

Звіт з лабораторної роботи повинен містити:

- Вступну частину
- Текст коду студента
- Копію екрану з результатами виконання індивідуального завдання
- Висновки

Список використаних джерел

1. Комп’ютерна графіка. Курс лекцій. Вінниця, 2016 р.
2. Маценко В.Г. Комп’ютерна графіка: Навчальний посібник. – Чернівці: Рута, 2009 – 343 с.
3. Вельтмандер П.В. Машинная графика: Учебное пособие в 3-х книгах. Книга 2. Основные алгоритмы / Новосиб. Ун-т. Новосибирск, 1997.- 193 с.
4. https://ru.wikipedia.org/wiki/Список_цветов_в_X11