Міністерство освіти і науки України

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Технології та засоби розробки комп’ютерної графіки та мультимедіа

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського як навчальний посібник для здобувачів ступеня бакалавра
спеціальності F2 Інженерія програмного забезпечення
спеціальності F6 Інформаційні системи та технології
спеціальності F7 Комп’ютерна інженерія

Укладачі: М. С. Хмелюк

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2025

УДК 004.92 (075.8)

Укладачі: *Хмелюк Марина Сергіївна*

Рецензент *Лісовиченко О.І.,* к.т.н., доцент кафедри інформатики та програмної інженерії КПІ ім. Ігоря Сікорського

Відповідальний редактор *Амонс О.А.,* к.т.н., доцент кафедри інформаційних систем та технологій КПІ ім. Ігоря Сікорського

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 05.03.2026р.)
за поданням вченої ради факультету Інформатики та обчислювальної техніки
(протокол № 7 від 23.02.2026 р.)

Технології та засоби розробки комп'ютерної графіки та мультимедіа. [Електронний ресурс] : лаб. практикум : навч. посіб. для здобувачів ступеня бакалавра за спец. F2 Інженерія програмного забезпечення, F6 Інформаційні системи та технології, F7 Комп'ютерна інженерія / КПІ ім. Ігоря Сікорського ; уклад.: М. С. Хмелюк. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2025. – 148 с.

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Технології та засоби розробки комп'ютерної графіки та мультимедіа». Посібник містить роз'яснення щодо виконання лабораторного практикуму, передбаченого робочою програмою дисципліни «Технології та засоби розробки комп'ютерної графіки та мультимедіа». В посібнику викладено інформаційні матеріали та приклади виконання завдань Посібник може бути корисним студентам відповідних спеціальностей при вивченні дисциплін, пов'язаних обробкою різних видів інформації: зображень, тексту, відео, звуку..

УДК 004.92 (075.8)

Реєстр. № НП 25/26-232. Обсяг 7,4 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

ЗМІСТ

ВСТУП

Найважливішою функцією комп'ютера є обробка інформації. Окремо можна виділити обробку інформації, пов'язану із зображеннями. Вона поділяється на три основні напрямки:

– комп'ютерна графіка (КГ);

– обробка зображень;

– розпізнавання зображень.

Завданням комп'ютерної графіки є візуалізація, тобто створення зображення. Візуалізація виконується виходячи з опису (моделі) того, що потрібно відображати. Існує багато методів та алгоритмів візуалізації, які різняться між собою, залежно від того, що і як відображати.

Комп'ютерна графіка поділяється на:

– статичну (нерухома);

– динамічну (рухому).

Кожна з яких у свою чергу ділиться на 2-вимірну та 3-вимірну графіку.

Залежно від методу формування зображень, комп’ютерна графіка поділяється на [1]:

– растрову (машинна графіка, в якій зображення представляється двовимірним масивом точок (елементів растру), колір та яскравість кожної з яких задається незалежно;

– векторну (об'єктно-орієнтована графіка, яка описує зображення, яке складається з простих об'єктів, такий як, контури, графічні примітиви та ін. за допомогою математичних формул);

– фрактальну (обчислювана графіка, зображення будується за рівнянням або системою рівнянь, але відрізняється векторної графіки тим, що об'єкти в пам'яті комп'ютера не зберігаються).

Більше 90% інформації здорова людина отримує через зір або асоціює з геометричними просторовими уявленнями. Комп'ютерна графіка має величезний потенціал для полегшення процесу пізнання і творчості, вона дозволяє розвивати просторову уяву, практичне розуміння, художній смак.

Комп'ютерною графікою останнім часом займаються багато людей, що обумовлено високими темпами розвитку обчислювальної техніки та застосуванням в багатьох областях людської діяльності, таких як, поліграфія, web-дизайн, мультимедіа, двовимірна та тривимірна графіка, відеомонтаж, навчальні системи, геоінформаційні системи, системи автоматизованого проєктування та ін.

В залежності від методів формування зображень та виду діяльності є свої технології за засоби розробки комп’ютерної графіки.

Мультимедіа (Multimedia) у перекладі - багато середовищ і означає об'єднання декількох засобів подання інформації та технологій (способів організації та виконання деяких дій з метою одержання продукту з заданими властивостями) в одній системі.

Мультимедіа - поєднання мультимедійних об'єктів: тексту, графіки, звуку, відео, анімації та просторових моделей та інших ефектів в одному файлі.

Мультимедіа технології тісно пов'язані такими інформаційними процесами, як зберігання інформації, передача та обробка.

Важливої особливістю мультимедіа є інтерактивність, тобто можливість користувача керувати відтворенням інформації.

В залежності від наповнення, мультимедійний контент може бути навчальним, ігровим, розважальним, інформаційним та ін. Прикладом мультимедійного продукту може бути віртуальна реальність (віртуальні музеї, інтерактивні тренажери).

Дисципліна «Технології та засоби розробки комп'ютерної графіки та мультимедіа» належить Вибіркових освітніх компонент спеціальностей: F2 Інженерія програмного забезпечення, F6 Інформаційні системи та технології, F7 Комп'ютерна інженерія. Вивчення дисципліни базується на знаннях, навичках та уміннях, отриманих під час вивчення студентами дисциплін «Вища математика», «Програмування»».

Основна мета вивчення дисципліни – формування та закріплення у студентів здатності до розуміння предметної області та професійної діяльності, здатності вчитися і оволодівати сучасними знаннями, оброблення та узагальнення інформації з різних джерел, здатності застосовувати технології та засоби для створення графічних об'єктів: растрової й векторної графіки, двовимірної та тривимірної графіки й анімації, управління графічними об'єктами, здатності створення та оброблення мультимедійної інформації, яку можна використовувати при розробці програмного забезпечення, створенні мультимедіа продукції.

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Посібник призначений для виконання комп1ютерних практикумів з дисципліни «Технології та засоби розробки комп'ютерної графіки та мультимедіа». Він складаються з 8 тем комп'ютерного практикуму, де для кожної із тем систематизовані теоретичні матеріали, приклади практичного засвоєння теоретичного матеріалу, завдання, та рекомендована література. В процесі вивчення даної навчальної дисципліни студенти знайомляться з методами представлення графічний об'єктів, колірними моделями, двовимірними перетвореннями, тривимірними перетвореннями, технологіями та засобами створення та відображення графічних об'єктів, управління графічними об'єктами використовуючи середовище розробки Microsoft Visual Studio, технологію .NET та мову C# , OpenGL ; мультимедіа технологіями, засобами для обробки мультимедійної інформації та створення мультимедіа продукції (Gimp, CapCut).

При вивченні конкретної теми слід уважно ознайомитися з поставленим завданням, пропрацювати матеріали комп'ютерного практикуму по цій темі. Лабораторний практикум відпрацьовується студентами групи індивідуально, протягом практикуму студенти виконують всі заплановані практикуми один за одним. Кожна практична робота комп'ютерного практикуму оформляється у вигляді звітного документу. Звіт повинен містити наступні розділи:

– стандартний титульний лист (вказати: ВНЗ, факультет, кафедру; тему та номеру поточної роботи; код групи та П.І.Б. виконавця; П.І.Б. викладача що перевірятиме роботу);

- назва практикуму та мета роботи;
- завдання до практикуму;
- тези теоретичних відомостей;
- результати виконання;

– висновки по роботі.

Microsoft Visual Studio це інтегроване середовище розробки (IDE), яке надає набір інструментів для створення, тестування та розгортання програм для різних платформ (Windows, Linux, macOS, мобільних системах, веб платформах) [2]. Є версія Visual Studio Community. яка розповсюджується на безоплатній основі, її можна встановити з офіційного сайту Microsoft.

IDE Visual Studio містить такі компоненти:

– єдина інтерактивна оболонка, яка забезпечує виклик всіх інших компонентів, не виходячи із середовища;

– текстовий редактор для набору та редагування вихідних текстів програм;

– система підтримки збірки (build), тобто компіляції проєктів з вихідних кодів, що включає компілятор з вихідної мови, що реалізується;

– налагоджування (debugger) для налагодження програм в середовищі за допомогою типового набору команд: встановити контрольну точку зупинки; зупинитися на заданому методі, візуалізувати значення змінних або регістрів та областей пам'яті;

– профілювальник (profiler);

– рефакторинг (refactoring);

– генератор тестів (unit test generator);

– система керування версіями вихідних кодів та ін;

.NET є кросплатформенна платформа з відкритим вихідним кодом від Microsoft для створення різноманітних застосунків.

Ідеї та принципи . NET [3]:

– це компіляція з будь-якої мови в проміжний двійковий код MSIL (Microsoft intermediate language);

– генерація компілятором так званих метаданих - інформації про типи, що визначаються та використовуються у програмному модулі для .NET;

– проміжний код, метадані та список вмісту бінарного коду складають збірку (assembly) - одиницю подання бінарного коду в .NET.

Збірка є базовою структурною одиницею в .NET, на рівні якої проходить контроль версій, розгортання і конфігурація програми.

Збірки кристалізують всю бібліотеку класів .NET - при написанні коду і створенні збірки своєї програми ми використовуємо простори імен, які розміщені в інших збірках .NET.

Збірки мають такі складові:

– маніфест, який містить метадані збірки;

– метадані типів. Використовуючи ці метадані, збірка визначає розташування типів у файлі програми, а також місця розміщення їх у пам'яті;

– код програми мовою MSIL, в який компілюється код C# (якщо ви використовуєте для написання коду мову C#);

– ресурси.

Всі ці компоненти можуть знаходитися в одному файлі, і тоді збірка представляє єдиний файл у форматі exe або dll (рисунок 0.1).

Рисунок 0.1 – Збірка - файл у форматі dll

Але також можливо, що ці компоненти зберігаються в окремих файлах. Наприклад, є основний файл exe, який має метадані збірки та типів і який використовує додаткові файли ресурсів – різні зображення, звукові файли, допоміжні модулі, елементи інтерфейсу, локалізовані для різних культур (рисунок 0.2).

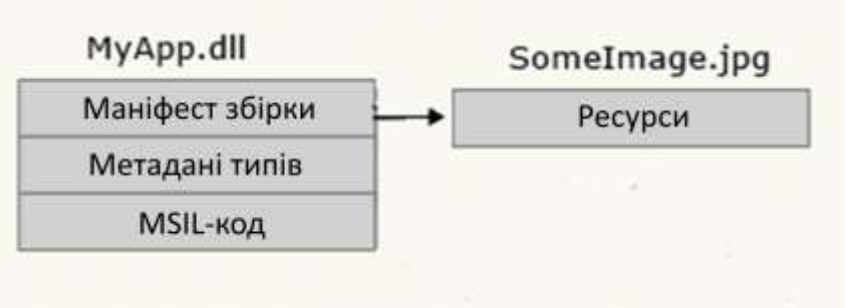


Рисунок 0.2 – Збірка - файл у форматі dll, ресурси не входять в dll

Маніфест збірки призначений для наступних завдань:

– перерахування файлів, які є в збірці;

– зіставлення посилань на типи та ресурси збірки з файлами, що містять оголошення та реалізації цих типів та ресурсів;

– перелік інших збірок, від яких залежить ця збірка;

– надання власного опису збірки.

Середовище VS при розробці програм оперує двома категоріями сутностей: рішеннями (solutions) і проєктами (projects). Рішення - це більша одиниця: рішення може складатися з одного або кількох проєктів. Можливе також створення порожнього рішення, до якого поступово додаються проєкти.

Код програмного проєкту може мати складну структуру та складатися з кількох файлів вихідного коду та конфігураційних файлів.

У зв'язку з цим, середовище Visual Studio полегшує створення проєктів за допомогою шаблонів (**templates**) (рисунок 0.3).

Шаблон задає типову структуру коду проєкту та його конфігураційних файлів, і розробнику залишається лише додати в шаблон конкретний код.

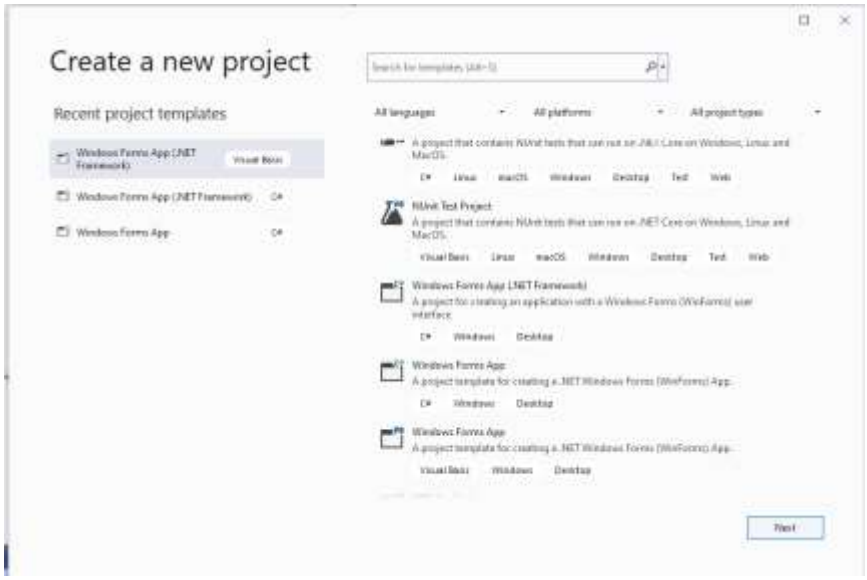


Рисунок 0.3 – Шаблони в VS

В браузері рішень Solution Explorer можна побачити структуру рішення, яка згенерована за замовчуванням (рисунок 0.4).

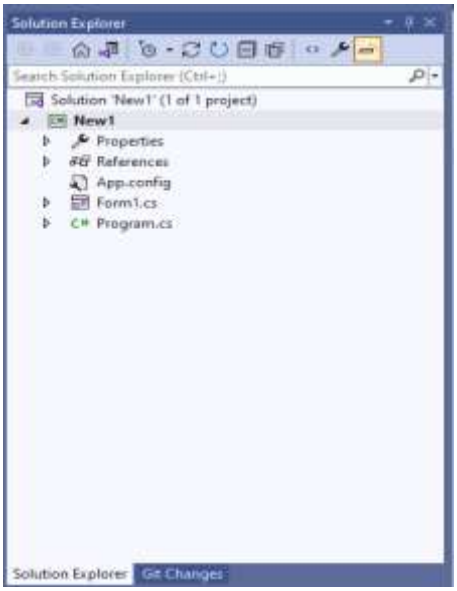


Рисунок 0.4 – Solution Explorer

Наприклад для WinForm App [4]:

– вузол Properties або Властивості - зберігає файли властивостей програми);

– вузол References - вузол містить збірки dll, які додані в проєкт за замовчуванням. Ці збірки містять класи бібліотеки .NET, які буде використовувати C#. Збірки можна додавати та видаляти;

– файл конфігурації App.config;

– файл коду програми Program.cs.

За замовчуванням Visual Studio при створенні проєкту додає файл AssemblyInfo.cs (рисунок 0.5), який можна знайти у вузлі Properties:

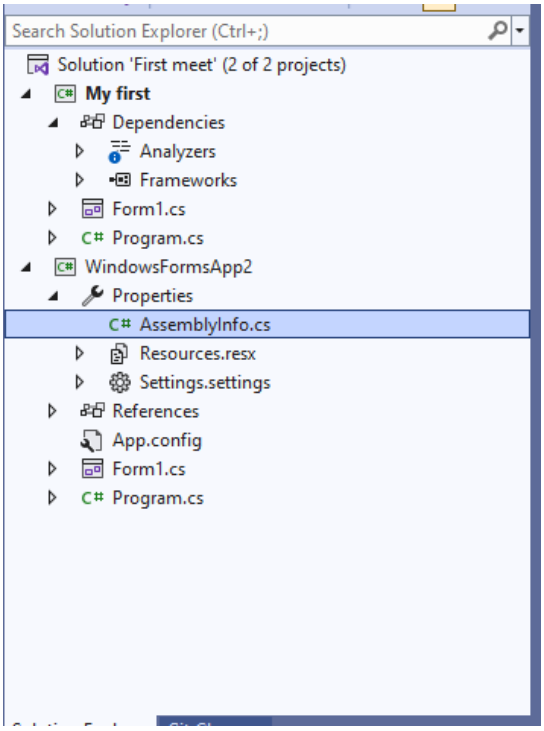


Рисунок 0.5 – Файл AssemblyInfo.cs в структурі проєкту

Сенс цього файлу полягає в тому, що він задає налаштування маніфесту збірки (рисунок 0.6).

Через атрибути типу **[assembly: AssemblyVersion("1.0.0.0")]** можна встановити значення у маніфесті. Префікс **assembly:** перед атрибутом вказує, що це атрибут рівня збірки, у даному випадку атрибут номера версії збірки.

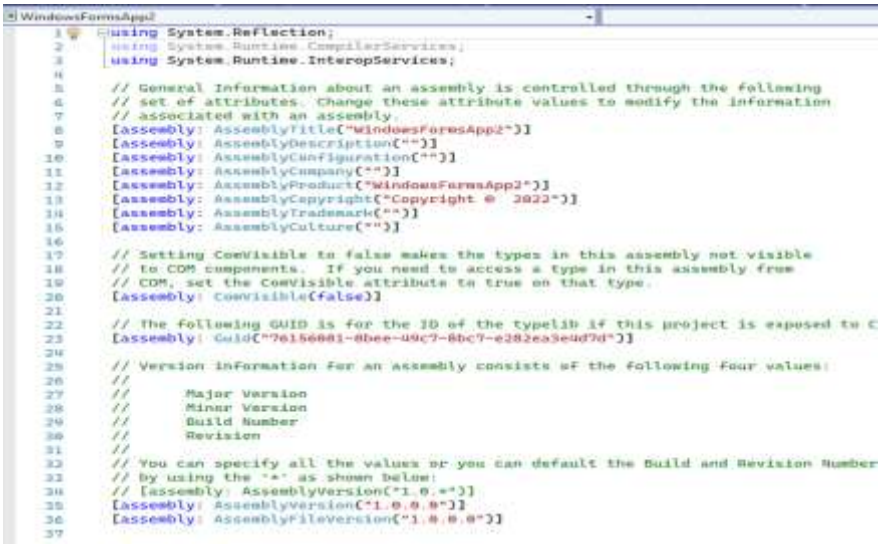


Рисунок 0.6 – Вміст файлу AssemblyInfo.cs

Призначення деких атрибутів:

– AssemblyCompany - назва компанії;

– AssemblyConfiguration -конфігурація збірки (Release або Debug);

– AssemblyCopyright - авторське право на програму;

– AssemblyDescription - короткий опис збірки;

– AssemblyCulture - задає мову та регіональні параметри, що підтримуються збіркою (наприклад, встановлення української культури: [assembly:AssemblyCultureAttribute("UA")]);

– AssemblyVersion - версія збірки.

У вікні Solution Explorer виділяємо назву проекту, клацаємо правою кнопкою миші та вибираємо в меню пункт Properties. У вікні містяться всі властивості поточного проєкту (рисунок 0.7):

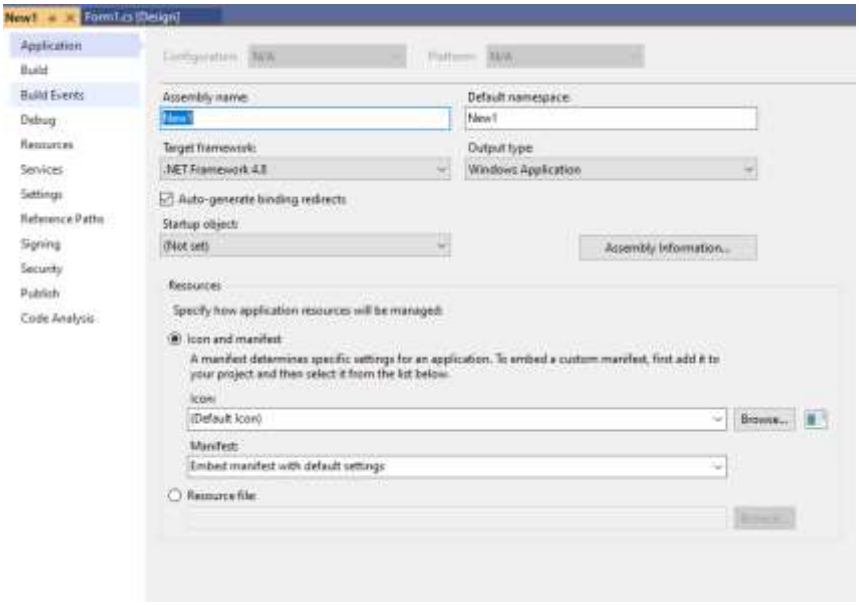


Рисунок 0.7 – Налаштування конкретного проєкту

При створенні/ налаштуванні проєкту апелюють поняттями **Assembly name** та **Namespace**. Namespace (простір імен) є логічною угодою, яка використовується під час розробки, в той час як assembly (ЗБІРКА) встановлює область видимості імені в процесі виконання.

Простір імен та збірка (файл, що містить реалізацію компонентів (типів)). Різні типи, що належать одному простору імен, можуть бути реалізовані в кількох збірках. Наприклад, тип resxfileref відноситься до простору імен System.Resources, але його реалізація знаходиться в збірці System.Windows.Forms.dll. Щоб виконати компіляцію коду, що посилається на тип resxfileref, слід додати до вихідного коду директиву using System.Resources.

Можна створити посилання на такі типи компонентів (рисунок 0.8):

– посилання на застосунок Магазину Windows;
– бібліотеки класів або збірки .NET Framework;
– компоненти COM;
– інші збірки та бібліотеки класів проєктів у тому самому рішенні;
– веб-служби XML.

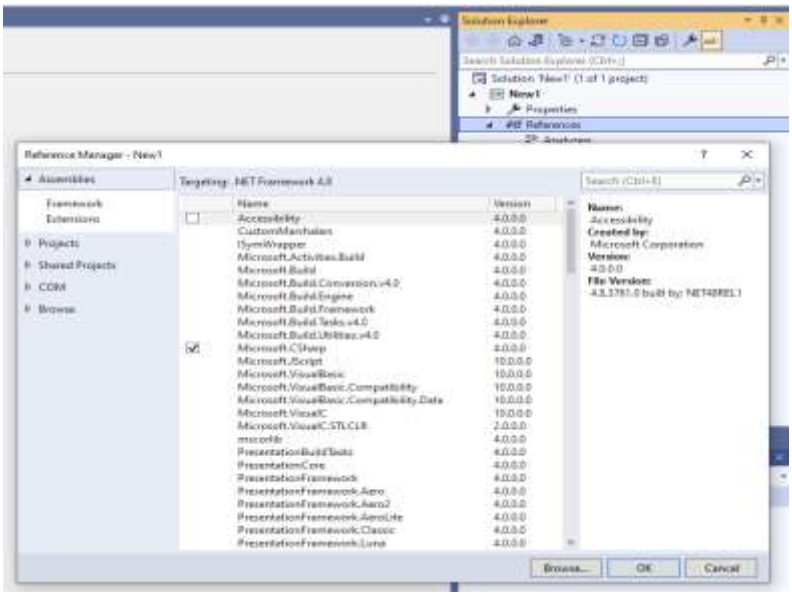


Рисунок 0.8 – Посилання на інші компоненти

Є два способи, залежно від того, які бібліотеки потрібно додати:

– якщо потрібно додати посилання на збірку, яка є у стандартній поставці Visual Studio, то на початку коду дописати using <назва> (рисунок 0.9);

– якщо треба додати зовнішнє посилання, то в дереві проєкту (браузер рішень) у розділі "Посилання/References" натиснути правою кнопкою миші та обрати пункт "Додати посилання/Add reference" і там jбрати те, що потрібно (рисунок 0.10).

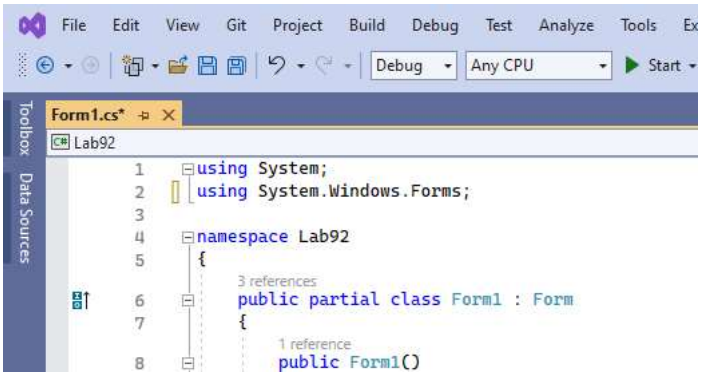


Рисунок 0.9 – Додавання посилань

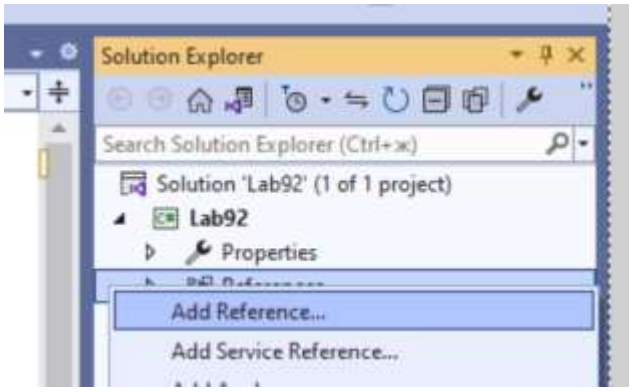


Рисунок 0.10 – Додавання посилань

Побудова Рішення включає в себе наступні можливості (рисунок 0.11) :

– Build компілює та лінкує тільки ті збірки, які змінилися (змінилися вихідні коди) з моменту попереднього білда + ті збірки, які залежать від даних збірок;

– Rebuild необхідний для того, щоб спочатку очистити проєкт, а потім виконати побудову всіх файлів та компонентів проєкту;

– Clean очищає рішення, щоб видалити всі проміжні та вихідні файли.

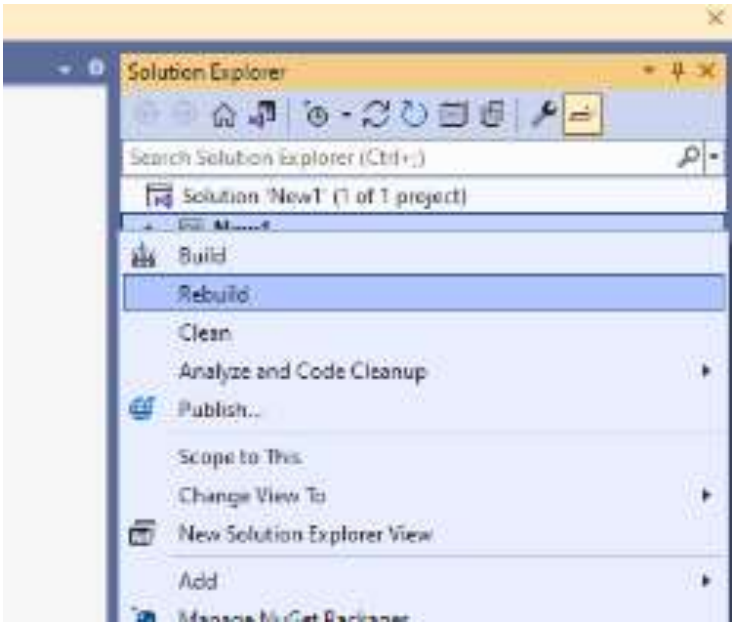


Рисунок 0.11 – Побудова Рішення

Налагодження програми - це той етап розробки програми, на якому програміст шукає і усуває всілякі помилки коду. Нас цікавлять помилки, що виникають на етапі виконання програми, а не на етапі компіляції. Саме вони можуть призвести до серйозних проблем та збоїв програми, виявити причину яких буває непросто.

У Microsoft Visual Studio для такої роботи є Debugger.

Приклад дебагінгу: На форму помістили кнопку та за подією Click для кнопки написали наступний код (рисунок 0.12):

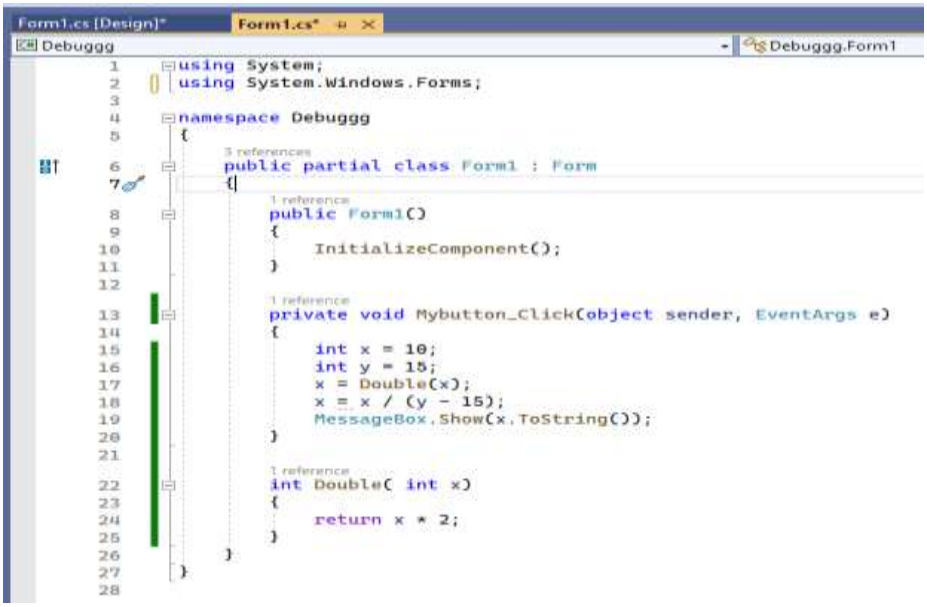


Рисунок 0.12 – Початковий код програми

У цьому коді відбувається виклик методу Double. Це не якийсь стандартний метод, його потрібно написати у коді форми (рисунок 0.13):

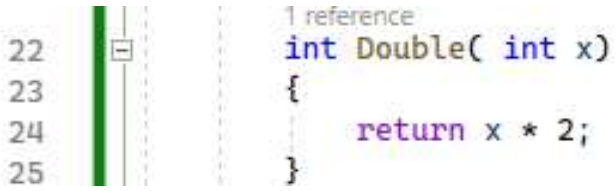


Рисунок 0.13 – Метод Double

Запустити програму із середовища розробки можна 2 способами:

– без налагодження (start without Debugging) CTRL+F5;

– з налагодженням (start Debugging) F5.

Для того, щоб налагодження програми було доступне, програма має бути ще запущена в режимі налагодження та з середовища розробки. Натискання клавіші F5 запускає програму в налагоджувальному режимі.

Для налагодження програма повинна бути скомпілована в конфігурації Debug, яка включає до файлу додаткову інформацію, необхідну при налагодженні програми.

Конфігурацію можна вибрати на панелі інструментів трохи правіше за кнопку запуску програми (рисунок 0.14).

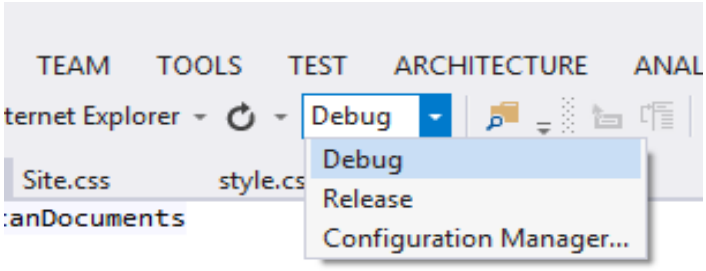


Рисунок 0.14 – Вибір конфігурації

Старт дебагу (рисунок 0.15):

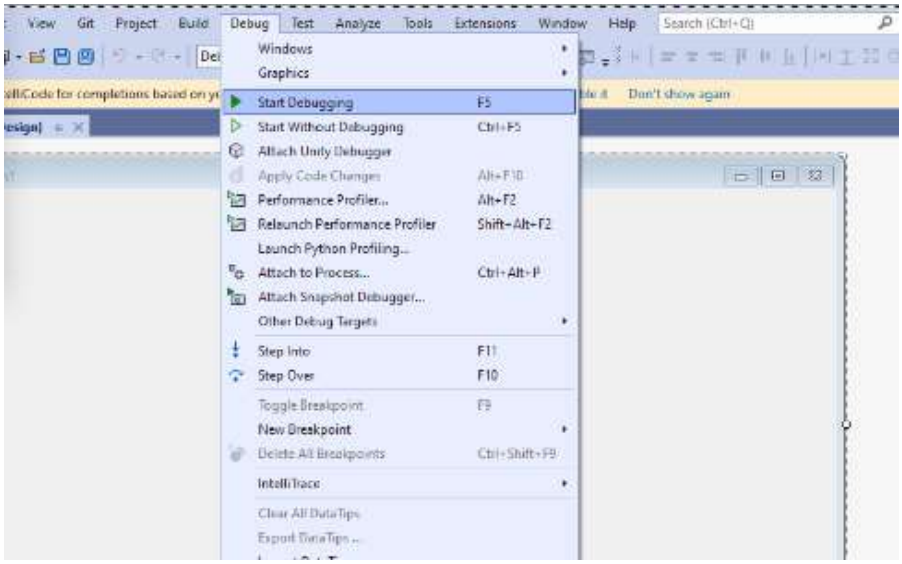


Рисунок 0.15 – Старт дебагу

Виникла помилка і середовище розробки взяло управління на себе, щоб ви змогли переглянути інформацію про помилку та спробували визначити її джерело.

У редакторі коду жовтим кольором виділено рядок коду, в якому і сталася помилка. Від цього рядка йде стрілка до вікна, в якому виняткова ситуація описана більш докладно (рисунок 0.16).

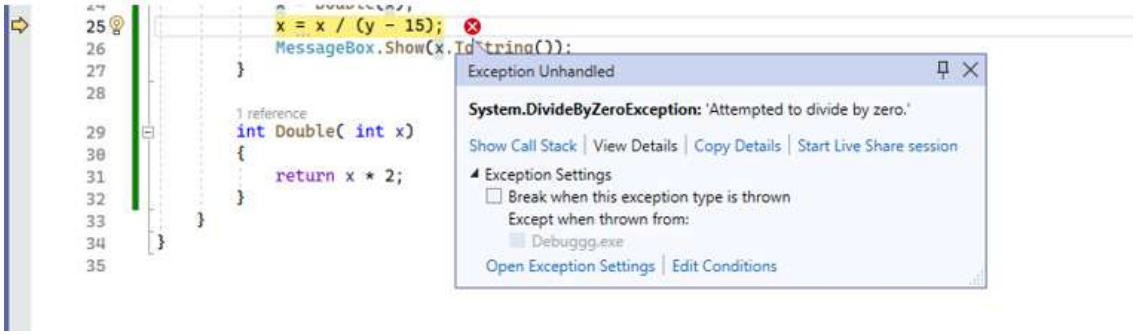


Рисунок 0.16 – Рядок з помилкою

У даному випадку, потрібно перевірити, щоб змінна y не дорівнювала 15 (рисунок 0.17):

Рисунок 0.17 – Додаємо перевірку

Отже, нам потрібно налагодити метод, і ми зробимо це з самого початку. Для цього потрібно поставити точку зупинки (переривання, breakpoint) у потрібному нам місці.

Для створення точки необхідно перейти на потрібний рядок:

1) натиснути F9;

2) вибрати меню Debug | Toggle Breakpoint;

3) двічі клацнути на смужці сірого кольору, ліворуч від рядка тексту у вікні редактора коду (рисунок 0.18).

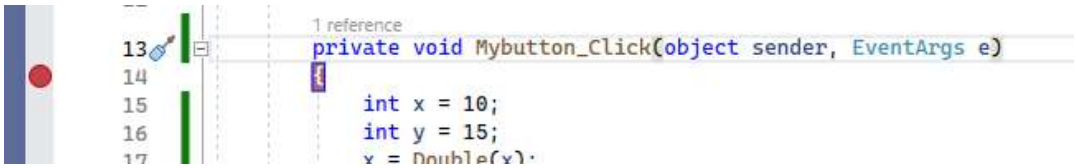


Рисунок 0.18 – Створення точки зупинки

Так як ми налагоджуємо метод із самого початку, то ставимо курсор на рядок з ім'ям методу та натиснемо F9.

Оскільки ми поставили точку зупинки, середовище розробки перехопить на себе виконання і виділить оператор, який можна виконати наступним кроком жовтим кольором (назвемо цю точку курсором покрокового виконання програми) (рисунок 0.19):

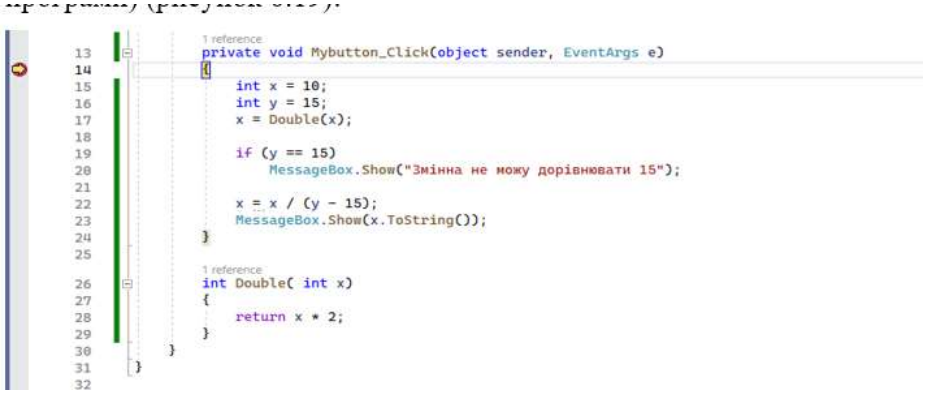


Рисунок 0.19 – Точка зупинки

Внизу вікна середовища розробки з'явилося дві панелі:

– Locals - де у вигляді списку представлені всі змінні, доступні у поточному методі. У цьому списку змінні представлені у вигляді трьох колонок: Name – назва змінної, Value – значення змінної, Type – тип;

– Autos - представлені змінні, пов'язані з поточною та попередньою інструкцією (рисунок 0.20):

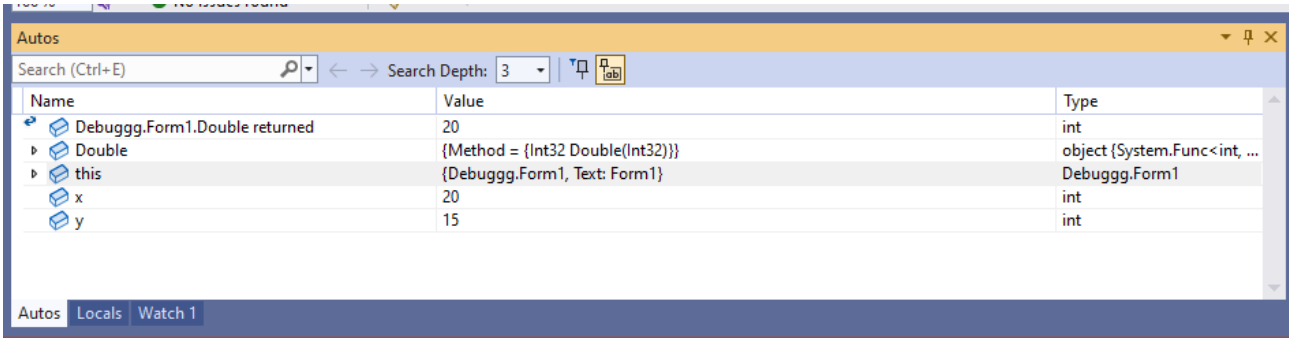


Рисунок 0.20 – Вкладка Autos

– Call stack - стек викликів. У цьому вікні перелічені методи, які були викликані раніше (рисунок 0.21).

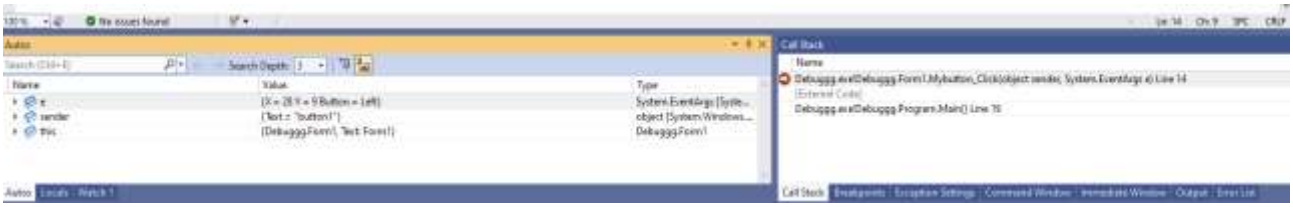


Рисунок 0.21 – Вкладка Call stack

Отже, наш курсор покрокового виконання зупинився на якійсь точці, і тепер хочемо розпочати тестування коду. з'явилася нова панель налагодження, там же ми знайдемо необхідні команди (рисунок 0.22):

Рисунок 0.22 – Панель налагодження

– Continue (F5) - продовжити виконання програми;

– Restart (Ctrl+Shift+F5) - перезапустити програму. Виконання програми буде перервано та запуститься заново;

– Stop debugging (Shift+F5) - зупинити налагодження. При цьому зупиниться виконання програми. виконання програми перерветься у тій точці, де зараз і перебуває, тобто, воно не буде завершено коректно і нічого не збережеться, якщо ви щось робили;

– Show Next Statement (Alt + Num*) - показати наступний оператор, тобто перемістити курсор редактора коду в курсор покрокового виконання. Курсор переміститися на початок оператора, який має бути виконаний наступним. У редакторі він виділений жовтим;

– Step Into (F11) - виконати черговий оператор. Якщо це метод, то перейти на початок цього методу, щоб почати налагодження. Наприклад, якщо ви знаходитесь на рядку: x = Double(x), то курсор покрокового виконання перейде на початок методу Double і ви зможете налагодити цей метод;

– Step Over (F10) - виконати черговий оператор. Якщо це метод, він буде повністю виконаний, тобто, курсор виконання не входитиме всередину методу;

– Step out (Shift + F11) - вийти з методу. Якщо ви налагоджуєте метод і натиснете цю кнопку, то метод виконається до кінця і курсор покрокового виконання вийде з методу та зупиниться на наступному рядку після виклику даного методу.

Результат знаходиться після вертикальної межі і він дорівнює нулю (рисунок 0.23).

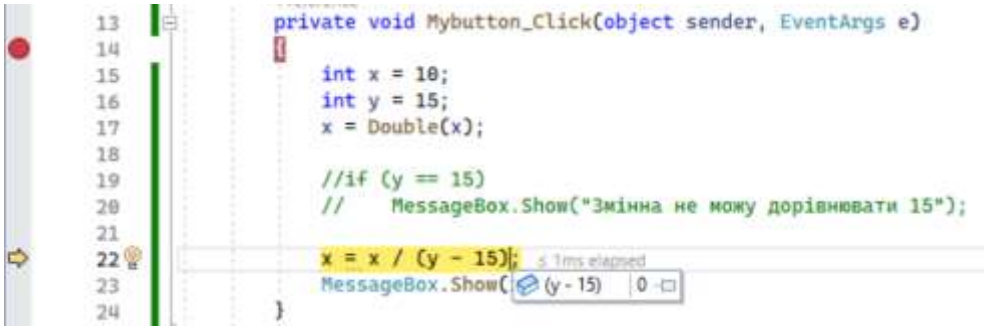


Рисунок 0.23 – Результат

Ви можете наводити мишкою на будь-яку змінну і debugger покаже вам її значення у вигляді підказки.

Панель Locals, закладка Watch (рисунок 0.24):

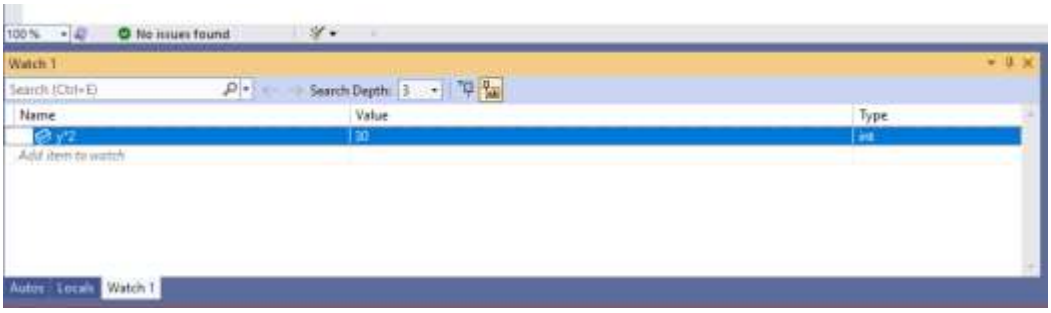


Рисунок 0.24 – Вкладка Call stack

Тут знаходиться список, в який ви можете вносити свої змінні та вирази, за значеннями яких ви хочете спостерігати.

На вкладці Locals видно змінні, які є актуальними для даного методу (рисунок 0.25).

Сюди включаються змінні, оголошені всередині методу і параметри методу, а також змінна this. Розкриваючи дерево об'єкта this ви можете побачити значення всіх властивостей і навіть об'єктів на поточній формі

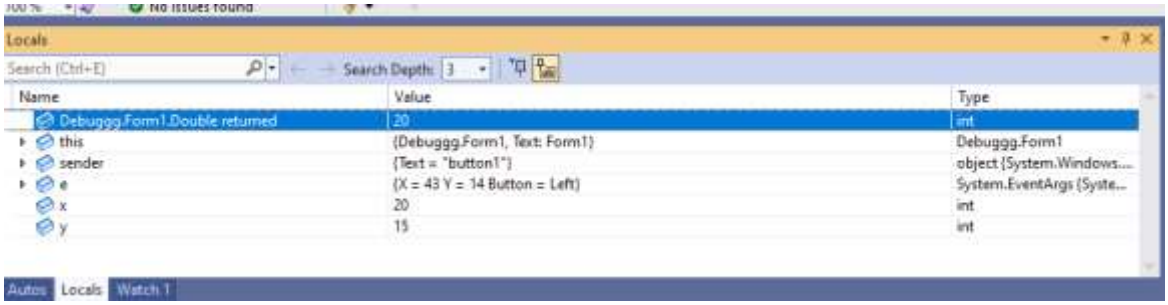


Рисунок 0.25 – Вкладка Locals

Можна використовувати вікно Точки зупинки (Налагодження / Вікна / Точки зупинки) для перегляду всіх точок зупинки рішення (рисунок 0.26):

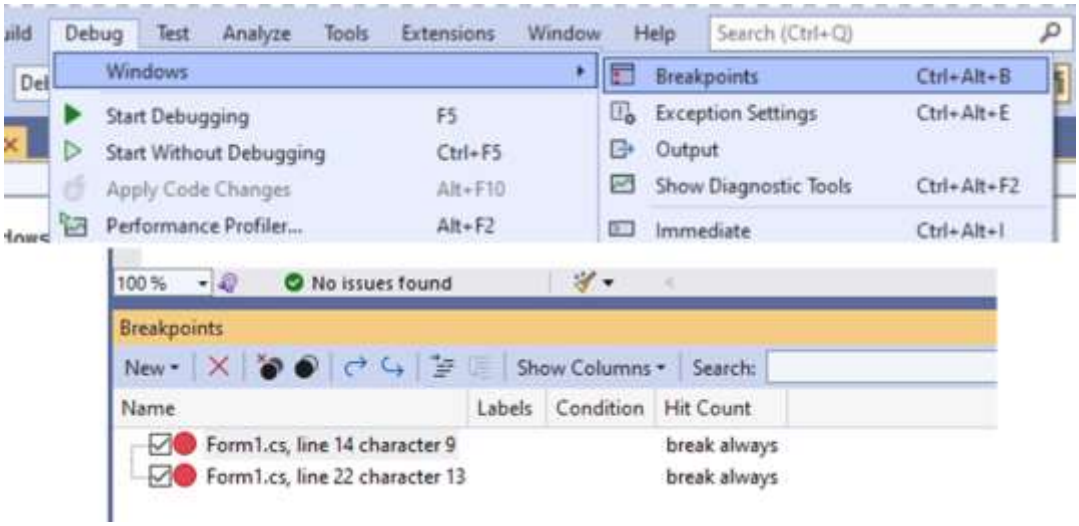


Рисунок 0.26 – Вікно «Точки зупинки»

Visual Studio дозволяє налаштовувати точки зупинки так, щоб вони з’являлися лише у разі дотримання певних умов.

Щоб налаштувати точку зупинки, клацніть на ній правою кнопкою миші та виберіть одну з наведених нижче опцій (рисунок 0.27, 0.28):

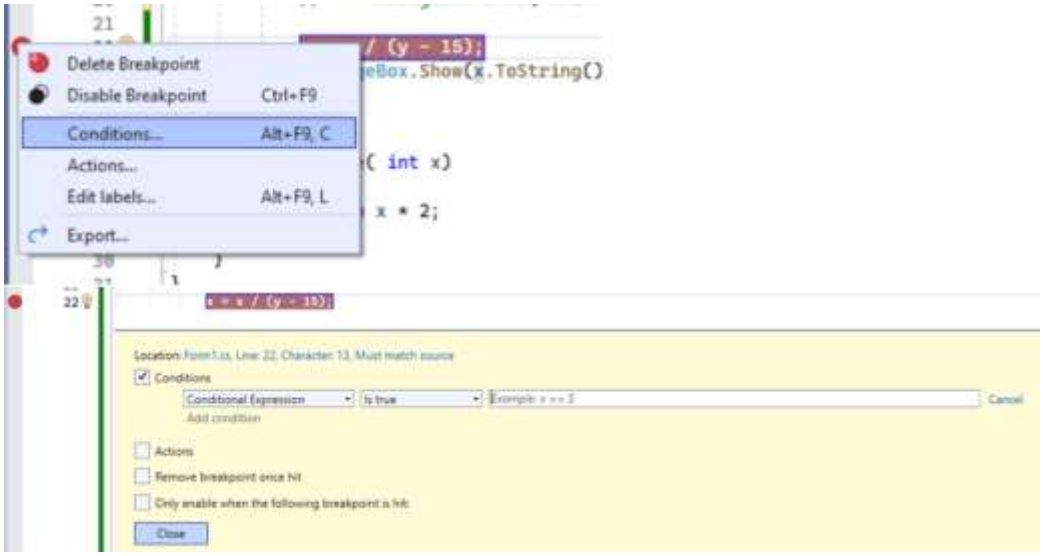


Рисунок 0.27 – Налаштування точок зупинки (умови)

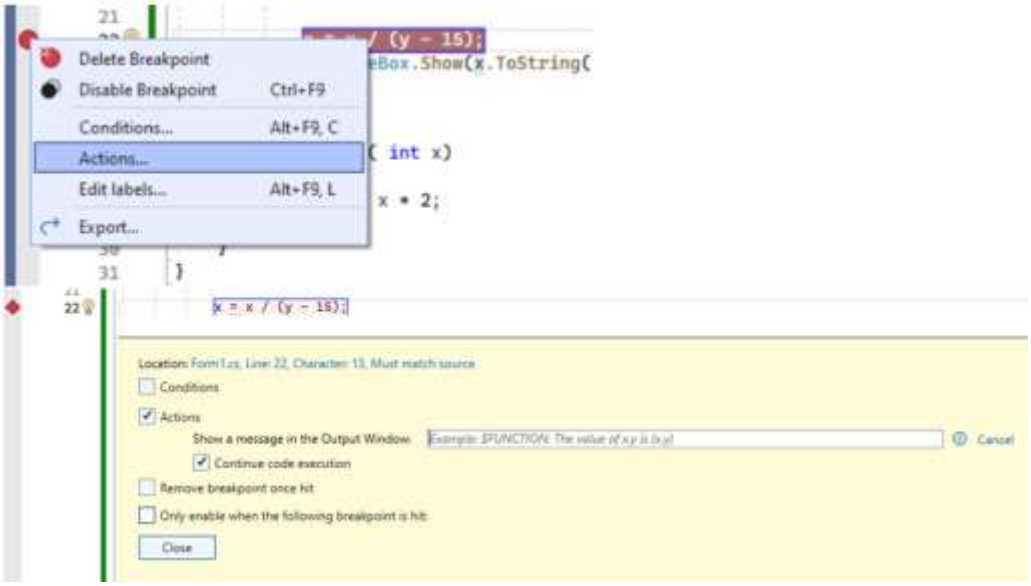


Рисунок 0.28 – Налаштування точок зупинки (дії)

ЛАБОРАТОРНИЙ ПРАКТИКУМ №1 Базові поняття комп'ютерної графіки. Ініціалізація графічного середовища

Мета: познайомитись з Visual Studio, класом Graphics, для роботи з графікою, навчитись створювати графічне середовище, застосовувати методи класу Graphics: DrawLine(), DrawRectangle(), DrawEllipse(), DrawPolygon(), DrawArc(), DrawCurve(), FillRectangle(), FillEllipse(), FillPolygon(), DrawString() та класи: Pen, Brush, Color для малювання об’єктів та відображення на пристрої.

Завдання

Створити Windows Forms-застосунок [4]:

– намалювати іконку і встановити її в формі;

– встановити фон форми;

– на формі повинен бути текст «Лабораторна робота №1. Виконав студент групи ... ПІБ »;

– до тесту застосувати різні стилі;

– вивести на екран Прізвище виконавця за допомогою ліній, кривих використовуючи класи Graphics, Pen, Brush C#.

– реалізувати закриття форми при натисканні на кнопку;

– написати звіт (текст програми, результат роботи, висновки).

Теоретичні відомості

Форма являє собою екранний об'єкт, зазвичай прямокутної форми, який можна застосовувати для надання інформації користувачу і для обробки введення інформації від користувача. Форма - це об'єкт, який задається властивостями, які визначають їх зовнішній вигляд, методами, що визначають їх поведінку, і подіями, що визначають їх взаємодію з користувачем.

Форми, як і всі об'єкти в .NET, є екземплярами класів, успадкованих від System.Windows.Forms.Form. Форма, яку ви створюєте за допомогою Visual Studio Designer, є класом. Коли ви будете відображати форму під час виконання програми, цей клас буде використовуватися як шаблон для відображення вікна.

Форму можна створювати повністю в коді програми, однак простіше використовувати для цього Visual Studio Designer.

Те, що ви бачите на екрані при створенні нової програми, називається вікном дизайнера. У цьому вікні, по суті, в графічному вигляді відображається код вашої програми. Дизайнер призначений для зручного та інтуїтивного створення інтерфейсу програми, призначеного для користувача інтерфейсу Основні елементи дизайнера форм:

- Properties Window (пункт меню View /Properties Window);
- Layout Toolbar (пункт меню View/Toolbars/Layout);
- Toolbox (пункт меню View/Toolbox).

У вікні дизайнера форм відображається тільки графічне представлення візуальних компонент форми. Всі дані вашої програми зберігаються кодом програми на мові C #.

Кнопкою називається елемент управління, вся взаємодія користувача з яким обмежується одним дією - натисканням. Все, що вам необхідно зробити при роботі з кнопкою - це помістити її в потрібному місці форми і призначити їй відповідний обробник події. Обробник призначається для події Click.

Елемент управління Label призначений для створення надписів до інших елементів управління або для виведення інформаційних повідомлень прямо на поверхні форми.

GDI і GDI+ [5]

Однією з сильних сторін Windows та й усіх сучасних операційних систем - є їх здатність абстрагуватися від деталей, які характеризують конкретні пристрої, без вказівок розробника. Наприклад, вам не потрібно розуміти роботу драйвера

пристрою жорсткого диска у тому, щоб програмно читати або записувати файли на диску. Ви просто викликаєте відповідні методи у відповідних класах .NET (або функції Windows API).

Цей принцип також справедливий по відношенню до малювання. Коли комп'ютер малює щось на екрані, він робить це шляхом відправки команд відеокарті. Однак на ринку присутні багато сотень різноманітних відеокарт, більшість з яких мають відмінний від інших набір команд і можливостей. Якщо б вам довелося брати це до уваги і писати специфічний код для кожного відеодрайвера, то написання звичайного застосунку стало б практично неможливим завданням. Ось чому, починаючи з ранніх версій Windows, з'явився інтерфейс графічних пристроїв (graphical device interface - GDI).

GDI забезпечує рівень абстракції, приховуючи різницю між різними відеокартами. Ви просто викликаєте функцію Windows API, щоб виконати специфічне завдання, а GDI всередині себе самостійно вирішує, як змусити певну клієнтську відеокарту виконати те, що необхідно при запуску певного фрагмента коду. Tfкож, якщо у клієнта є кілька пристроїв відображення, наприклад, моніторів і принтерів, GDI забезпечує практично однаковий результат при виведенні одного і того ж зображення на екран і принтер. Якщо клієнт бажає надрукувати щось замість відображення його на екрані, ваш зпастосунок просто має повідомити системі, що пристроєм виведення буде принтер, після чого викликати ті ж функції API.

GDI пропонує високорівневий програмний інтерфейс для розробників. У комп'ютерній графіці, в тому числі і в Windows є така структура даних - «Контекст пристрою» (device dontext - DC), яка створюється інтерфейсом GDI, для представлення підключених пристроїв, вона містить інформацію про графічні об'єкти (лінії, криві, пензлі для малювання та заповнення, шрифти) та режими, необхідні для малювання на певному пристрої (екрані, принтері). Наприклад, щоб відобразити зображення на екрані, застосунку Windows

необхідний контекст пристрою екрану, а для виводу на друк, необхідний контекст іншого пристрою - принтера. Тобто, роль контексту пристрою полягає в забезпеченні обміну даними між застосунком і пристроєм. Також роллю DC є зберігання набору встановлених графічних атрибутів (наприклад, колір лінії, шрифт тексту). Значення будь-якого графічного атрибуту можна змінити, викликаючи певні функції GDI.

Наприклад, апаратним пристроям необхідно повідомляти, де слід малювати об'єкти, і звичайно їм потрібні координати, відлічувані відносно верхнього лівого кута екрана (або іншого вихідного пристрою). Однак програми зазвичай відображають щось в клієнтської області (області, зарезервованої для малювання) власного вікна, можливо, використовуючи власну систему координат. Оскільки вікно може бути позиціоновано в будь-якому місці екрану, і користувач в будь-який момент може його перемістити, перетворення між цими системами координат можуть виявитися непростим завданням. Однак DC завжди знає, де знаходиться ваше вікно, і може виконувати такі перетворення автоматично.

В GDI + контекст пристрою поміщений в оболонку базового класу .NET з ім'ям System.Drawing.Graphics. Велика частина малювання виконується викликом методів екземпляра Graphics.

GDI+ - це рівень, що знаходиться між GDI і застосунком, який надає більш інтуїтивно зрозумілу об’єктну модель. Хоча GDI + - це в основному оболонка навколо GDI, тим не менш, Microsoft за допомогою GDI + пропонує ряд нових можливостей і збільшену продуктивність у порівнянні з деякими старими засобами GDI.

Частина базової бібліотеки класів .NET, пов'язана з GDI +, велика. Важливо зрозуміти фундаментальні принципи, що стосуються малювання, щоб мати можливість, при необхідності, легко розширити свої знання. Повні списки всіх класів і методів, доступних в GDI +, містяться в документації SDK.

Основні простори імен, у яких знаходяться базові класи GDI+ :

System.Drawing містить більшість класів, структур, перерахувань і делегатів, які забезпечують базову функціональність малювання.

System.Drawing.Drawing2D представляє основну підтримку для двовимірної і векторної графіки, включаючи згладжування, геометричні трансформації і графічні шляху.

System.Drawing.Imaging містить різні класи, що забезпечують маніпуляції з графічними зображеннями (бітові карти, файли GIF тощо).

System.Drawing.Printing містить класи, що мають відношення до друку і попереднього перегляду виводяться на друк зображень.

System.Drawing.Design включає деякі зумовлені діалогові вікна, таблиці властивостей і інші елементи інтерфейсу, що мають відношення до розширення користувальницького інтерфейсу часу проєктування.

System.Drawing.Text включає класи для виконання більш складних маніпуляцій зі шрифтами і родинами шрифтів.

Клас Graphics надає методи малювання на пристрої відображення.

Щоб намалювати лінію за допомогою інтерфейсу GDI+, потрібно створити два об'єкти: об'єкт Graphics і об'єкт Pen. Об'єкт Graphics містить методи, які безпосередньо виконують малювання, а об'єкт Pen служить сховищем атрибутів, таких як колір, ширина і стиль лінії.

Клас Graphics реалізує методи малювання об'єктів на пристрої дисплея. Найпростіший і популярніший спосіб отримати об'єкт цього класу - використовувати обробник події Paint. Потрібно створитиь програму і для форми створити обробник події Paint. Середовище розробки зробить метод для обробника події:

```
private void Form1_Paint(object sender, PaintEventArgs e)
 {
 }
```

Перший параметр sender - надає посилання на об'єкт, який викликав подію. Другий параметр e - це змінна класу PaintEventArgs, передає об'єкт, що відноситься до оброблюваної події. Посилаючись на властивості об'єкта (і іноді його методи), можна отримати такі відомості, як розташування миші для подій миші або даних, що передаються подіях перетягування. Через цю змінну ми отримуємо важливі властивості:

ClipRectangle – область, яку потрібно перемалювати;

Graphics - екземпляр класу Graphics, який є поверхнею малювання. Саме на ній треба малювати.

Виходить, що для малювання на формі ми повинні використовувати об'єкт e.Graphics та його методи. Але це лише в обробнику події Paint. В інших методах і обробниках доведеться створювати екземпляр класу самому.

Малювання на формі можна здійснювати по кліку на Кнопку. Для цього на форму потрібно додати елемент управління Button і для кнопки створити обробник події Click. Середовище розробки зробить метод для обробника події:

```
private void FormOnForm_Click(object sender, EventArgs e)
 {
 }
```

Потрібно створити графічний об'єкт

```
Graphics graph = CreateGraphics();
```

Примітка. Стандартним чином створити об'єкт-полотно не вдається.

```
Graphics g = new Graphics();
```

На цьому операторі генерується помилка:

Для типу System.Drawing.Graphics не визначений конструктор.

```
private void FormOnForm_Click(object sender, EventArgs e)
 {
  Graphics graphics = e.Graphics;
```

```
graphics.DrawLine(new Pen(Color.Red), 0, 0, 200, 300);
}
```

Малювання можна здійснювати на елементі управління Panel. Для цього на форму потрібно додати елемент управління Panel. В цьому випадку графічну поверхню потрібно створити на елементі Panel:

```
private void btnDraw_Click(object sender, EventArgs e)
{
  Point my1 = new Point(x1, y1);
  Point my2 = new Point(x2, y2);

  graph = panelDraw.CreateGraphics();
  graph.DrawBezier(myPen, my1, my2, new Point(x3, y3), new Point(500,
300));
  }
```

Ще один варіант завдання графічного контексту на елементі PictureBox через растровий об'єкт класу Bitmap, який наслідується від абстрактного класу Image. У класі форми створимо два об'єкти:

```
Graphics graph; // графічний об'єкт - якесь полотно
Bitmap mybit; // буфер для Bitmap-зображення
```

При створенні об'єкта Bitmap аргументами конструктора потрібно вказати висоту і ширину зображення в пікселях. Розмір зображення обмежується висотою і шириною елементу PictureBox.

```
mybit = new Bitmap(pictureBoxDraw.Width, pictureBoxDraw.Height); // з
розмірами
pictureBoxDraw.Image = mybit;
```

Потім слід створити графічний контекст GDI+ зображення, представлений об'єктом System.Drawing.Graphics. Щоб створити об'єкт Graphics з Bitmap, використовуйте статичний метод Graphics.FromImage().

```
graph = Graphics.FromImage(mybit); // ініціалізація graph
```

За допомогою методів класу Graphics у растровому зображенні можна намалювати текст, форми та зображення.

```
public partial class FormOnBitmap : Form
{
    Graphics graph;
    Bitmap mybit;
      …
    private void btnDraw_Click(object sender, EventArgs e)
    {
            …
      mybit = new Bitmap(pictureBoxDraw.Width, pictureBoxDraw.Height);
      pictureBoxDraw.Image = mybit;
      graph = Graphics.FromImage(mybit);
      graph.DrawPolygon(new Pen(Color.BlueViolet), new Point[] { my1, my2,
my3, my4});
    }
}
```

Основні методи та властивості класу Graphics

Властивості:

Clip – регіон (прямокутна область), який визначає область малювання;

ClipBounds - область малювання у вигляді чотирикутника (клас RectangleF);

CompositingMode - спосіб малювання композитних картинок;

CompositingQuality - дозволяє задати якість відображення композитних зображень;

DpiX - горизонтальна роздільна здатність поверхні;

DpiY - вертикальна роздільна здатність поверхні;

PageScale - масштабування;

PageUnit — одиниці виміру на поверхні.

Методи класу Graphics:

Clear() — очистити поверхню малювання та залити її кольором, вказаним як параметр;

DrawArc() - намалювати дугу;

DrawBezier() - намалювати криву Безье;

DrawBeziers() - намалювати серію (декілька) кривих Без'є;

DrawCurve() - намалювати криву;

DrawClosedCurve() - намалювати замкнуту криву, кінець якої буде з'єднаний з початком кривої;

DrawEllipse() - намалювати еліпс;

DrawIcon() - намалювати значок;

DrawImage() - намалювати картинку;

DrawLine() - намалювати лінію;

DrawLines() - намалювати серію ліній;

DrawPolygon() - намалювати багатокутник по масиву точок;

DrawRectangle() - намалювати прямокутник;

DrawString() - відобразити рядок тексту;

FillEllipse(), FillPolygon(), FillRectangle(), FillRegion() - залити кольором еліпс, багатокутник, прямокутник або область;

FromHwnd() - статичний метод створення об'єкта Graphics на основі Hwnd значення компонента;

FromImage() - статичний метод створення об'єкта Graphics на основі картинки;

MeasureString() - розраховує розміри рядка тексту на поверхні під час використання певного шрифту.

Кожен метод приймає набір аргументів, необхідних для малювання об'єктів.

Для представлення точки, використовуються координати X, Y, які можуть бути типу int (1, 2, 3, …), float (1.0f, 2.0f, 3.0f, …). Точка з координатами X=0, Y=0 відповідає верхньому лівому куту форми або об’єкта, який є полотном для малювання.

Також точку можна передати через **Структуру Point**, яка представляє впорядковану пару цілих чисел координат X і Y, які визначають точку на двовимірній площині і **Структуру PointF**, яка представляє впорядковану пару чисел з плаваючою комою, які визначають точку на двовимірній площині.

Для малювання використовується ручка, яку повертає клас Pen, для заливки – клас Brush, який визначає об'єкти, що використовуються для заповнення внутрішніх просторів графічних фігур, таких як прямокутники, еліпси, круги, багатокутники та контури.

Для визначення кольору для ручки та заливки використовується клас Color, який представляє колір в колірній моделі RGB (Red, Green, Blue) або ARGB (Alpha, Red, Green, Blue), де кожен компонент має значення від 0 до 255, що дозволяє задавати прозорість та інтенсивність кольорів для графічних елементів. Прозорість (Alpha) має значення від 0 до 255: 0 - повністю прозорий, 255 - повністю непрозорий.

Клас Pen (System.Drawing.Pen)

При створенні об'єкта Pen можна вказати кілька атрибутів. Наприклад, один з конструкторів Pen дозволяє вказувати при створенні об'єкта його колір і ширину.

```
Pen myPen = new Pen(Color.Blue, 2);
```

Приклади:

```
myGraphics.DrawLine(myPen, 4, 2, 12, 6);
```

Штрихові лінії і завершення відрізків

Об'єкт Pen також дозволяє змінювати значення деяких своїх властивостей, таких як DashStyle.

myPen.DashStyle = DashStyle.Dash;

Змінюючи значення властивостей об'єкта Pen, можна задати багато атрибутів лінії. Властивості StartCap і EndCap визначають вид кінцевих точок відрізка. Кінець відрізка може бути плоским, квадратним, круглим, трикутним або мати довільну форму. Властивість LineJoin дозволяє вказувати, чи повинні з'єднуються лінії з'єднуватися під кутом (з виступаючими гострими кутами), зі скошеними, закругленими або обрізаними краями.

Клас заливки для фігур Brush

Клас Brush абстрактний, не можна створити безпосередньо його екземпляр, тому що він має абстрактні методи, які не реалізовані, але створені для того, щоб їх реалізовували нащадки (наприклад SolidBrush, LinearGradientBrush).

SolidBrush - суцільний, тобто ця заливка заповнює область повністю одним кольором та одним візерунком.

Пензель LinearGradientBrush дозволяє створювати лінійний градієнт. Як конструктор, об'єкт отримує дві точки (початок і кінець градієнта) і два кольори (початковий і кінцевий кольори градієнта). Маючи ці дані, система сама розфарбує область, що заповнюється плавним переходом від одного кольору до іншого.

Brush brush = new SolidBrush(Color.Red);

Brush brush = new LinearGradientBrush(p1, p2, Color.Red, Color.Wheat)

Метод DrawLine класу Graphics перевантажений, тому для нього підтримується кілька способів передачі аргументів. Наприклад, можна створити два об'єкти Point і передати ці об'єкти Point методу DrawLine в якості аргументів.

```
Point myStartPoint = new Point(4, 2);
Point myEndPoint = new Point(12, 6);
myGraphics.DrawLine(myPen, myStartPoint, myEndPoint);
```

Метод DrawString – для відображення тексту, приймає параметри: рядок для малювання, Font - шрифт для малювання, Brush колір, координати X, Y лівого верхнього кута мальованого тексту, форматування тексту.

```
myGraphics ("Text", new Font("Arial", 16), mybrush1, new PointF(200.0F, 50.0F));
```

Приклад створення проєкту та роботи з WinForm

Для створення нового проєкту в VisualStudio .NET вибираємо пункт меню File / New / Project ... (рисунок 1.1).

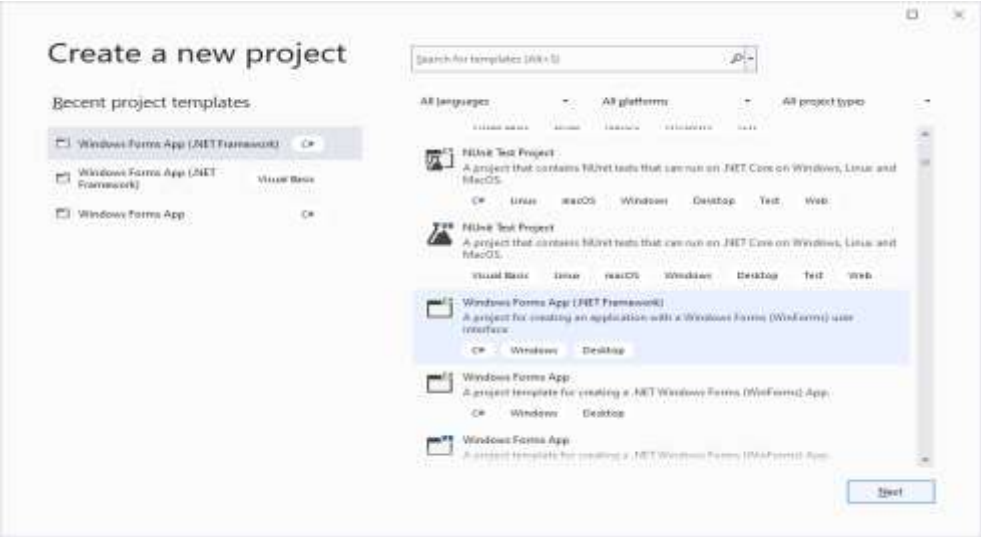


Рисунок 1.1 – Створення проєкту WinFormApp

З'явиться вікно, в якому треба вибрати шаблон, з врахуванням мови програмування, вказати назву Рішення (Solution), назву проєкту, і шлях до проєкту (рисунок 1.2).

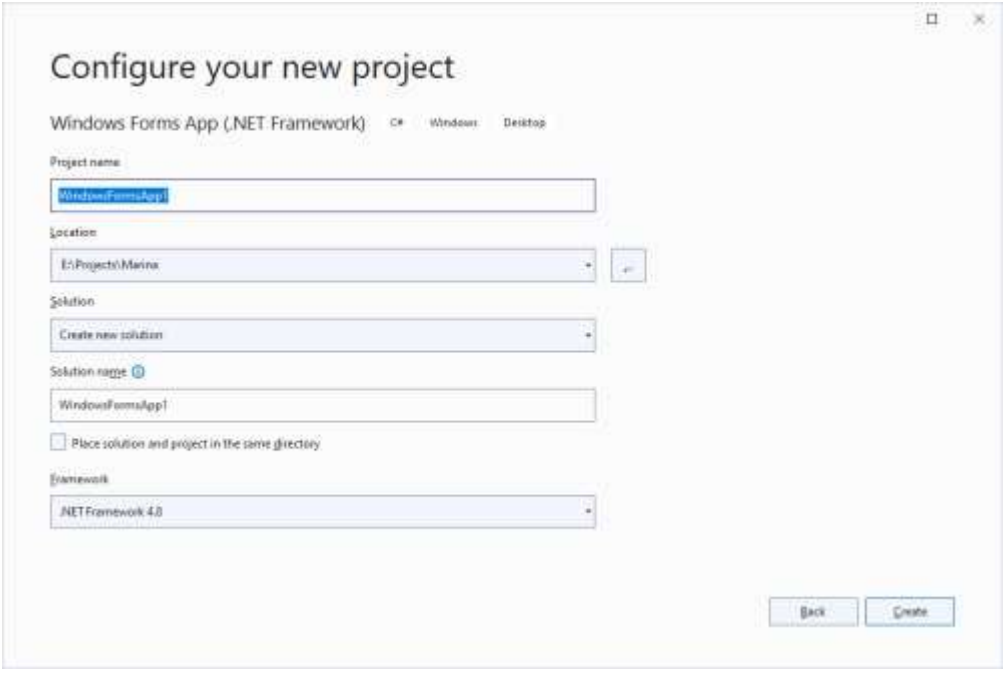


Рисунок 1.2 – Іменування проєкту

Проєкт створено (рисунок 1.3).

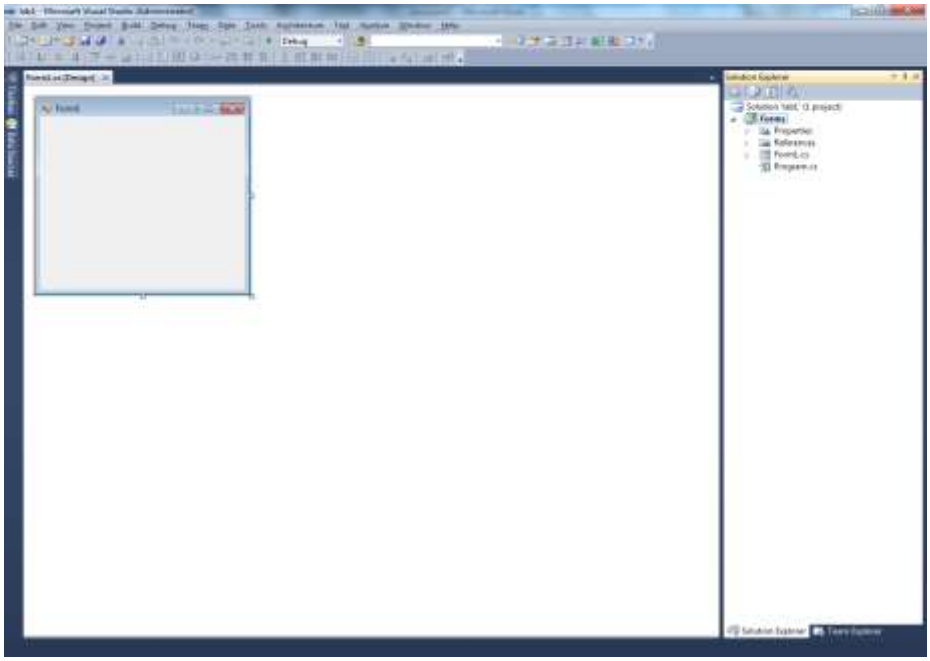


Рисунок 1.3 – Готове Рішення

Змінюємо назву файлу форми Form1 на AboutForm (рисунок 1.4).

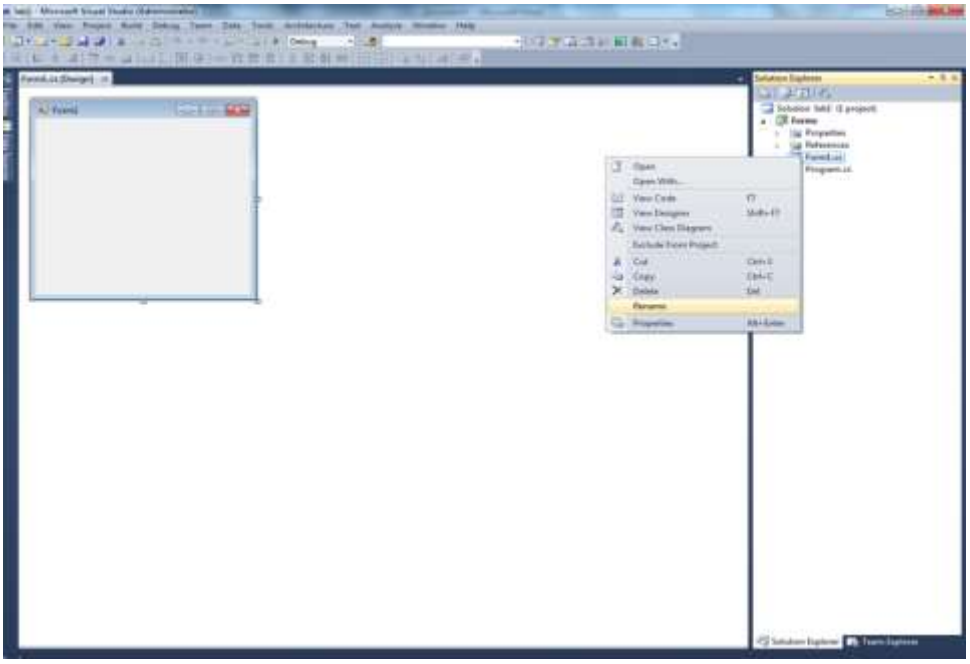


Рисунок 1.4 – Перейменування файлу

Після зміни назви з'являється нижчезазначених повідомлення про перейменування. Натискаємо «Yes» (рисунок 1.5).

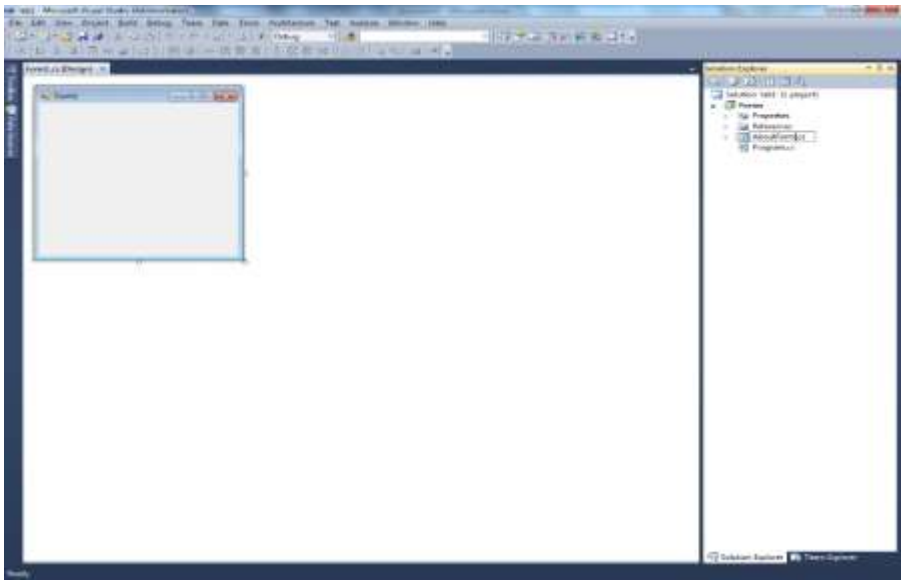


Рисунок 1.5 – Результат перейменування

Вибираємо форму і переходимо на вкладку «Properties». У властивості «Text» вказуємо назву форми (рисунок 1.6).

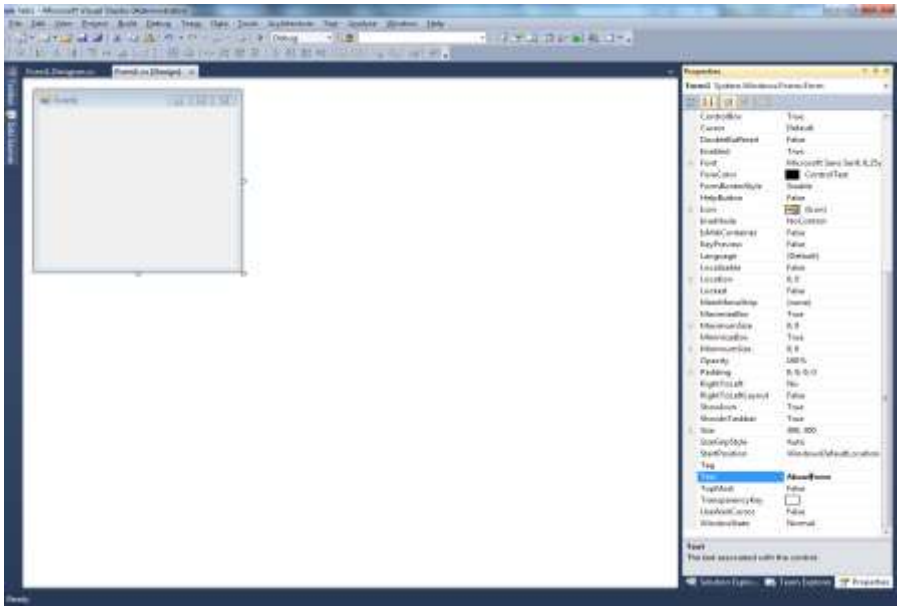


Рисунок 1.6 – Змінюємо текст форми (назву, яка відображається на формі)

З вкладки Toolbox перетягуємо компонент (Label), які дозволять відобразити на формі текст (рисунок 1.7).

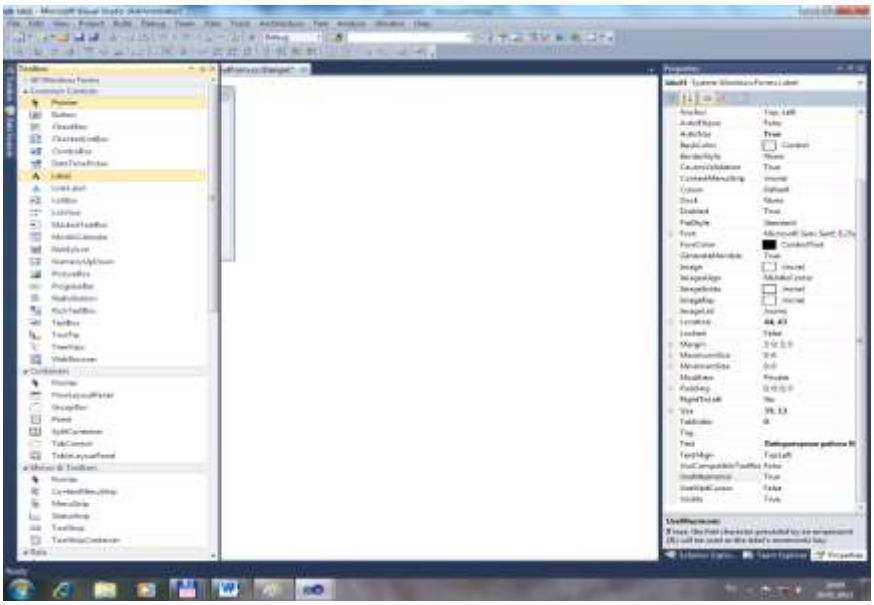


Рисунок 1.7 – Додавання на форму компоненту Label

Обираємо на формі компонент «Label». У властивості «Text» вказуємо потрібний текст і встановлюємо для нього властивості (колір, напрямок, розмір, стиль ...) (рисунок 1.8) .

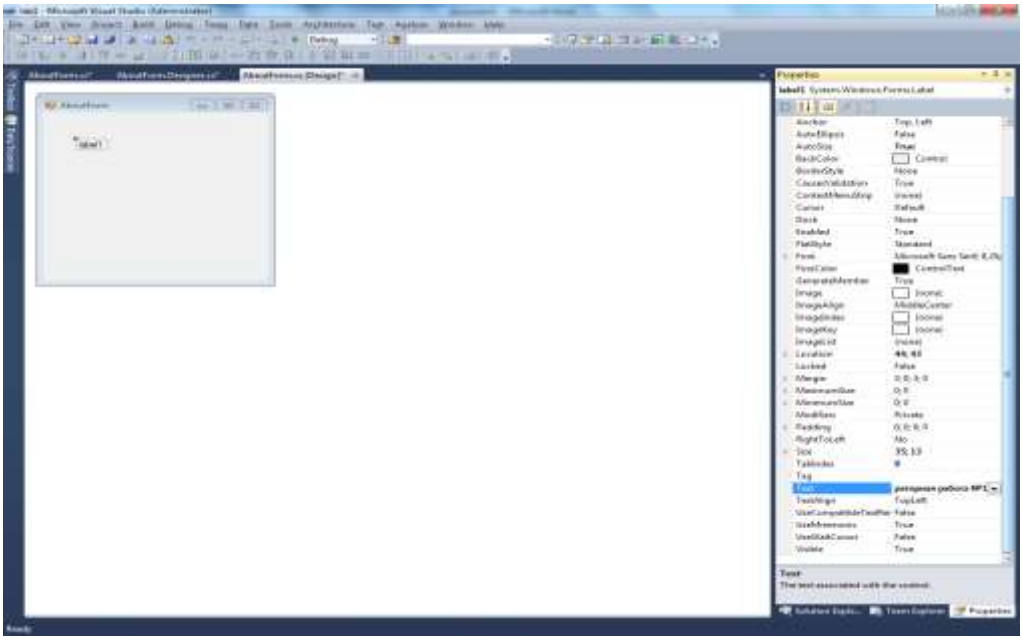


Рисунок 1.8 – Налаштування компоненту Label

Для створення іконки відкриваємо ресурсний файл (Properties / Resuorces.resx) і в пункті меню «AddResource» вибираємо пункт «Add New Icon» (рисунок 1.9):

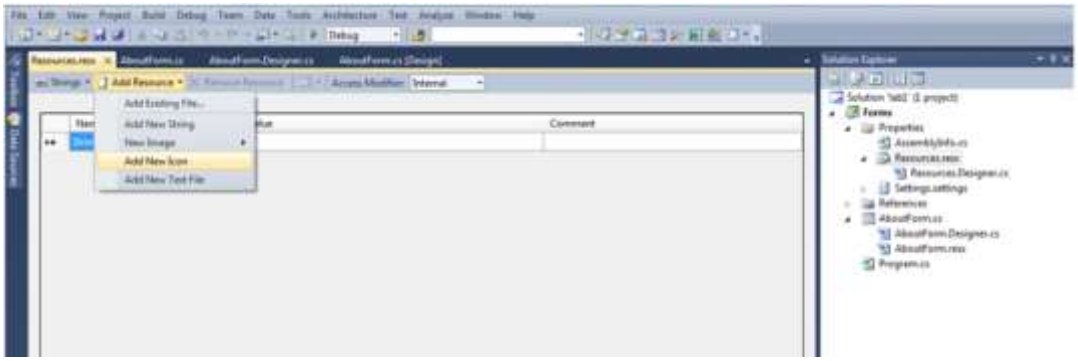


Рисунок 1.9 – Створення іконки

Вказуємо назву іконки (рисунок 1.10):

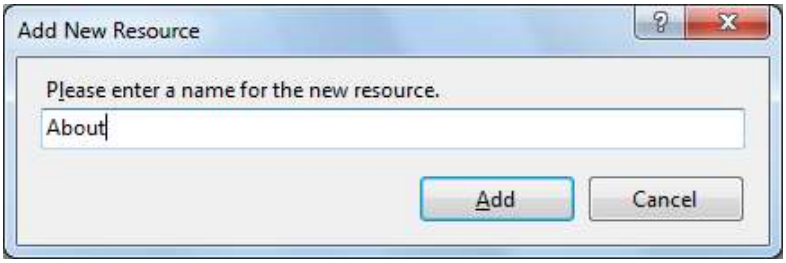


Рисунок 1.10 – Назва іконки

Малюємо іконку і зберігаємо (рисунок 1.11).

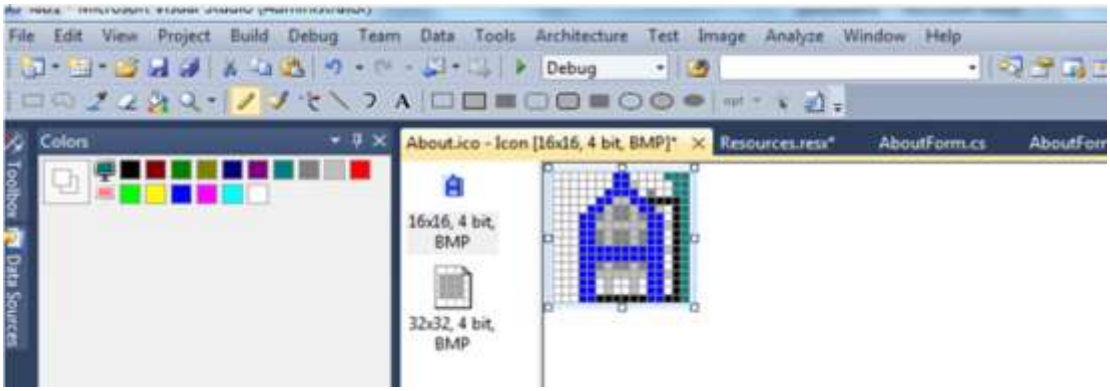


Рисунок 1.11 – Малювання іконки

Для установки іконки заходимо у властивості форми, і у властивості «Icon» встановлюємо нашу іконку (вказуємо шлях знаходження іконки) (рисунок 1.12, 1.13).

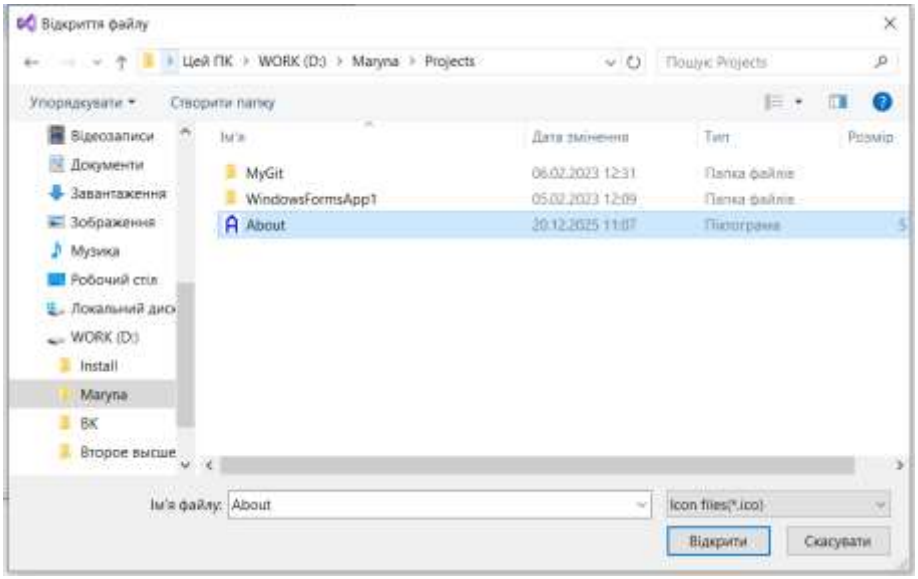


Рисунок 1.12 – Встановлення іконки на форму

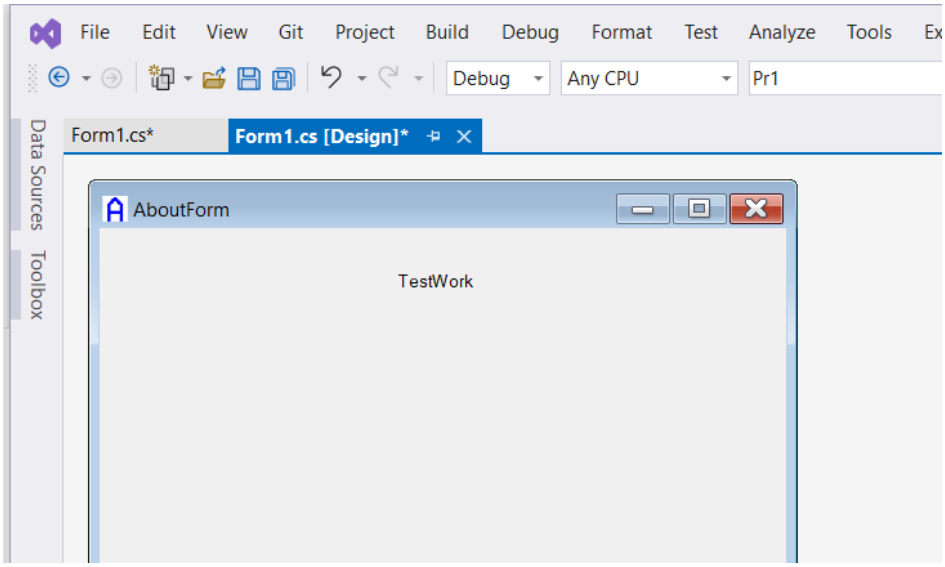


Рисунок 1.13 – Іконка на формі

У властивостях форми вказуємо колір фону форми (можна встановити картинку) (рисунок 1.14).

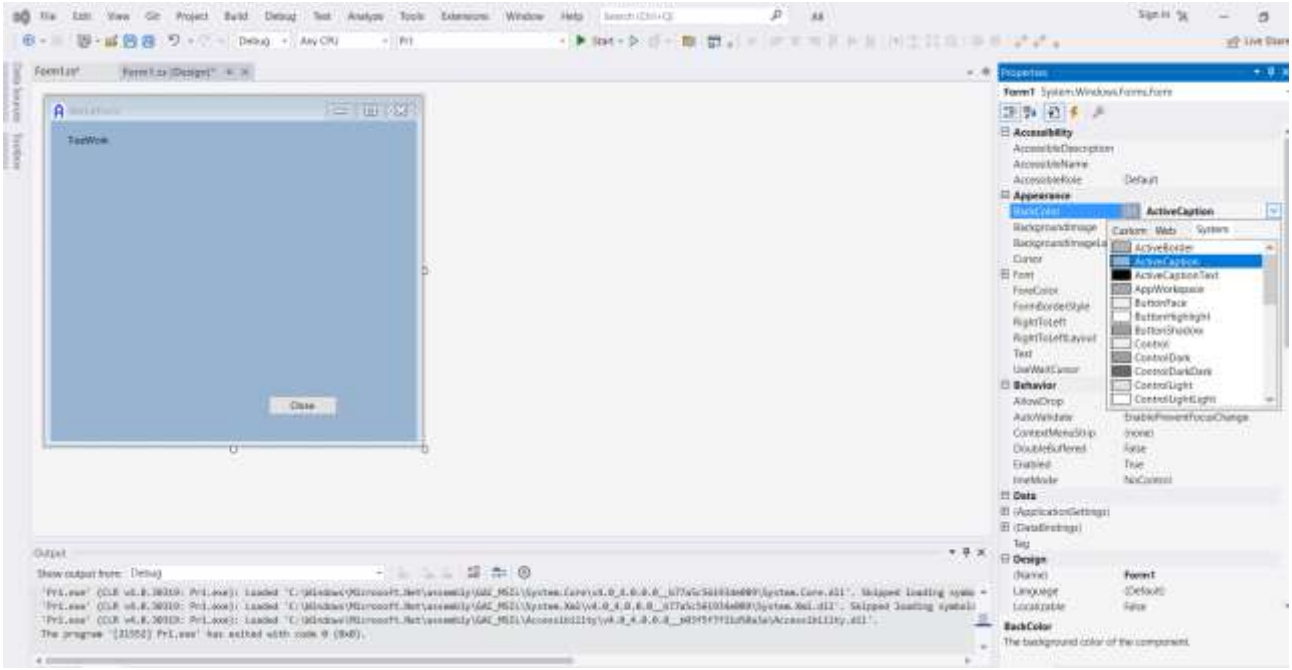


Рисунок 1.14 – Фон форми

З закладки Toolbox перетягуємо компонент «Button» (рисунок 1.15).

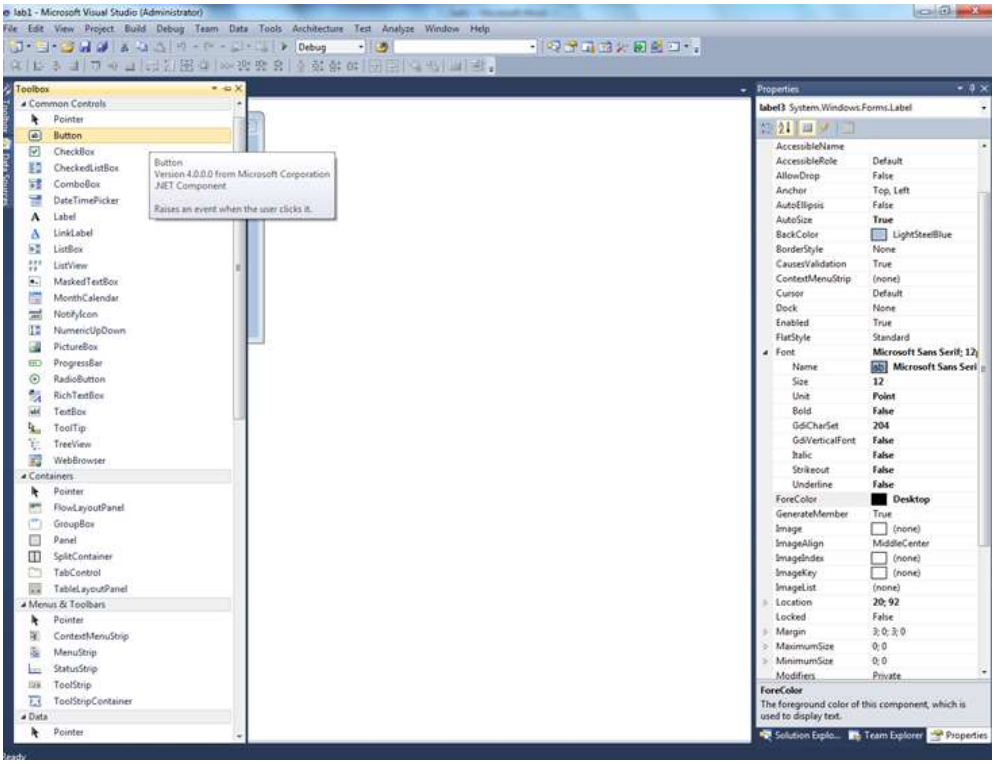


Рисунок 1.15 – Додавання на форму кнопки

У властивостях кнопки встановлюємо «Name» і «Text»

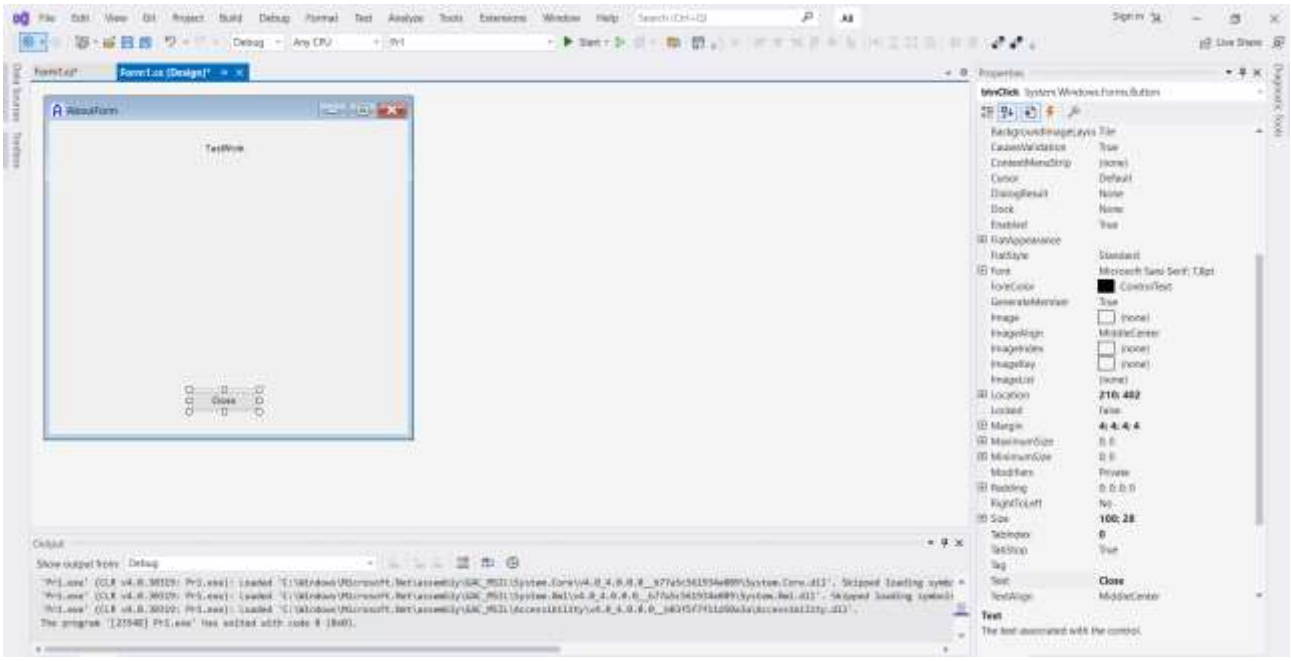


Рисунок 1.15 – Налаштування «Button»

При натисканні на кнопку (одинарний Click), створюється метод – обробник події на об’єкті Кнопка (Button) (рисунок 1.16).

Реалізуємо закриття вікна методом Close():

Метод Close() у батьківській формі закриває всю програму.

Метод Close() у дочірніх формах закриває лише дочірню форму.

Метод Application.Exit() закриває програму, якщо викликається як із батьківської форми, так і з дочірніх форм.

```
private void btnClose_Click(object sender, EventArgs e)
        {
      Close();
            //Application.Exit();
        }
```

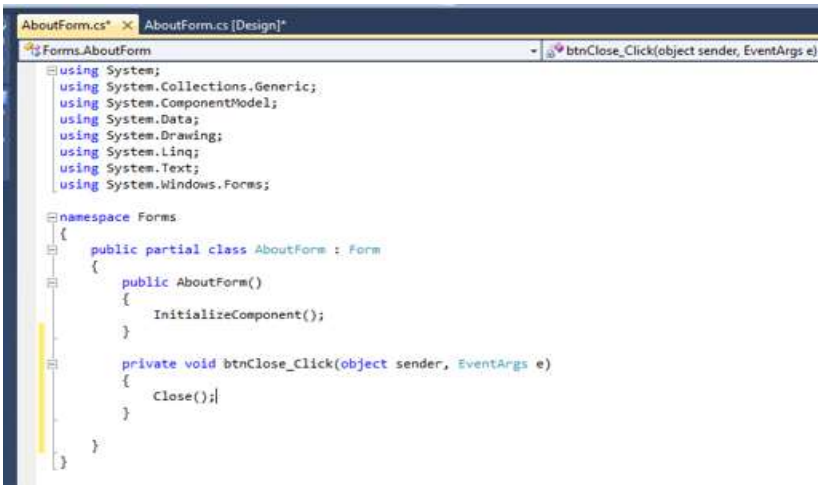


Рисунок 1.16 – Обробник події Click на об’єкті Button

Для малювання об’єктів скористаємось подією Paint, яка виникає під час перемальовування елемента управління. При створенні нового елемента або успадкованого елемента керування з іншим зовнішнім виглядом необхідно надати код для малювання елемента керування шляхом перевизначення методу OnPaint.

У властивостях форми, потрібно перейти на вкладки «Події» (блискавка) і для події Paint створити обробник події - подвійним кліком в полі біля події (рисунок 1.17).

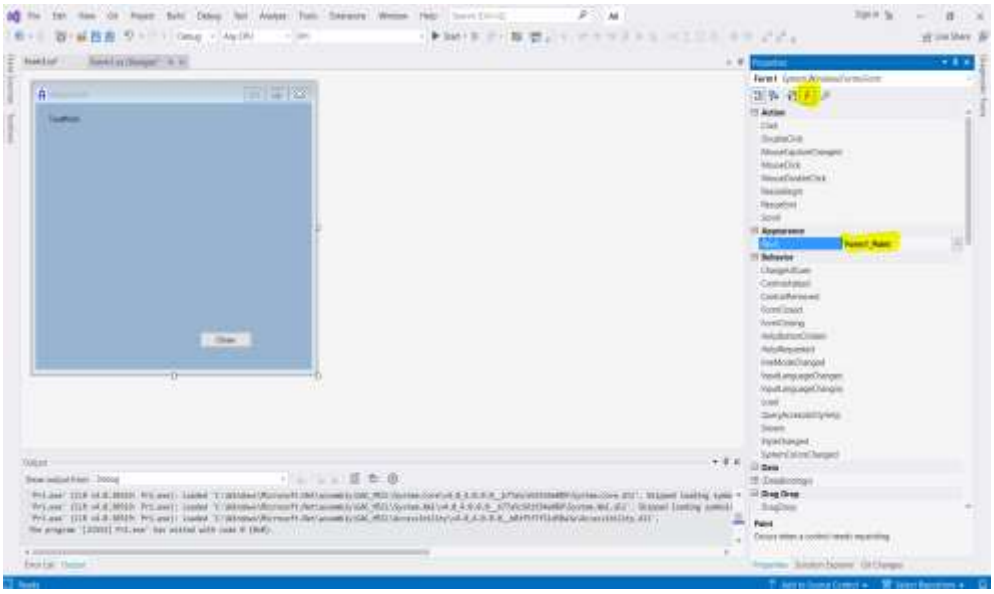


Рисунок 1.17 – Обробник події Paint на об’єкті Form

Далі, використовуючи методи класу Graphics: DrawLine(), FillEllipse(), DrawString(), створимо об’єкти відповідні об’єкти (рисунок 1.18).

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
   Graphics graph = CreateGraphics();
   graph.DrawString("Maryna", new Font("Arial", 16), mybrush1, new
PointF(200.0F, 50.0F));
   graph.DrawLine(new Pen(Color.Green, 5), x1, y1, x2, y2);
   graph.DrawLine(mypen, 300, 100, 300, 300);
   graph.DrawLine(mypen, x1, y1, 200, 200);
   graph.DrawLine(mypen, 200, 200, 300, 100);
   graph.FillEllipse(mybrush, new Rectangle(350, 250, 50,50));
 }
```

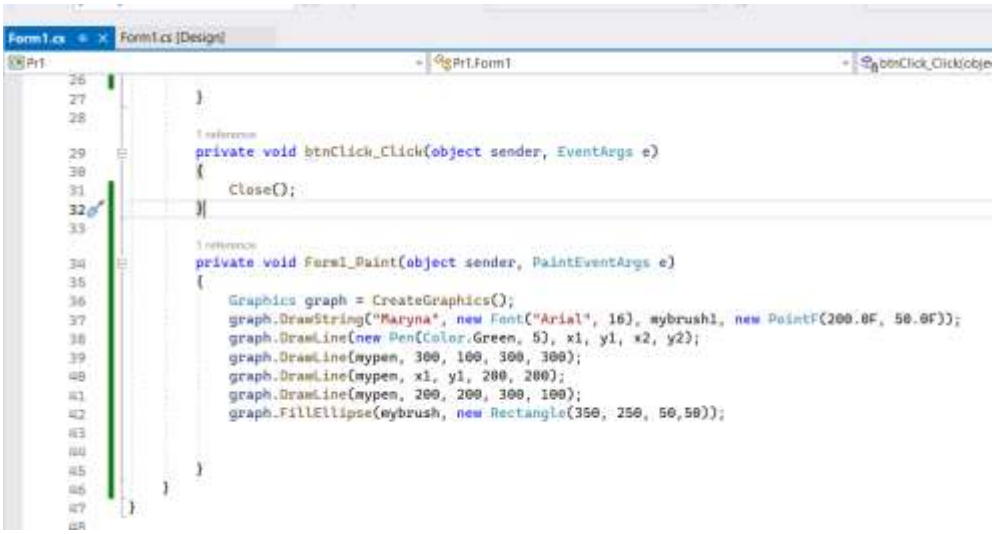


Рисунок 1.18 – Створення об’єктів

Запускаємо програму (рисунок 1.19).

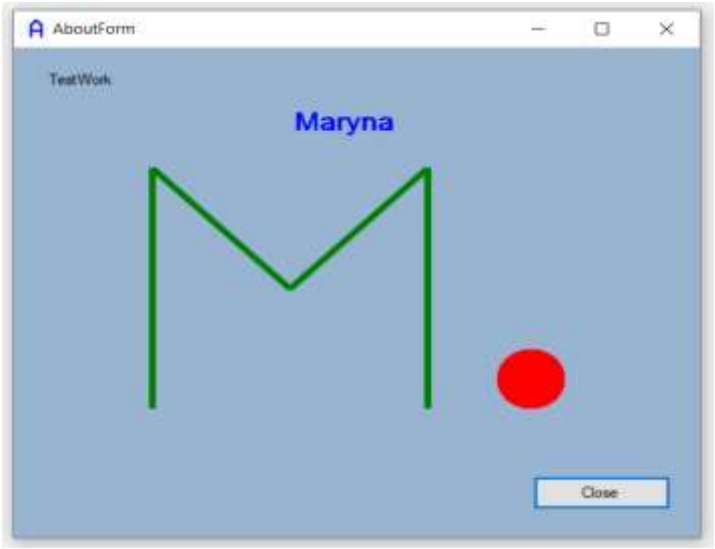


Рисунок 1.19 – Результат

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу.

ЛАБОРАТОРНИЙ ПРАКТИКУМ №2 Візуалізація лінійних зображень. Рекурсивні алгоритми при побудові лінійних зображень

Мета: ознайомитись базовими операторами мови C#, з лінійними та рекурсивними функціями.

Завдання

1) Побудувати та відобразити візерунок, утворений 50 вкладеними квадратами. Сторони першого квадрата паралельні осям координат екрану. Вершини кожного наступного квадрата - це точки на сторонах попереднього квадрата, що ділять ці сторони у відношенні до P = 0,08.

2) Побудувати трикутник Серпінського.

Теоретичні відомості

Оператор умови if.else

Конструкція if/else перевіряє істинність певної умови та залежно від результатів перевірки виконує певний код.

Найпростіша форма складається з блоку if:

```
if (умова)
{
   інструкції
}
```

Після ключового слова if ставиться умова. Умова повинна представляти значення bool. Це може бути значення типу bool або результат умовного виразу або іншого виразу, який повертає значення bool. І якщо ця умова є істинною (true), то спрацьовує код, який поміщений далі після умови всередині фігурних дужок.

Вираз else

Але що, якщо ми хочемо, щоб у разі недотримання умови також виконувались якісь дії, то ми можемо додати блок else:

```
If (умова)
{
   інструкції
}
else
{
   інструкції
}
```

Цикли

for

Оператор for виконує інструкції, поки певний логічний вираз дорівнює true.

```
for (дії до виконання циклу; [умова]; дії після виконання)
{
   інструкції
}

for (int i = 0; i < 10; i++)
{
   інструкції;
}
```

do while

У циклі do спочатку виконується код циклу, потім відбувається перевірка умови в інструкції while. І поки ця умова є істинною, цикл повторюється.

```
do
{
   інструкції;
}
while (умова)

int i = 5;
do
```

```
{
   інструкції;
}
while (i > 0);
```

while

На відміну від циклу do цикл while відразу перевіряє істинність деякої умови, і якщо умова істинна, то код циклу виконується:

```
while (умова)
{
   інструкції;
}
```

foreach

Цикл foreach призначений для перебору набору або колекції елементів.

```
foreach (тип даних змінна in колекція)
{
   інструкції;
}
```

Лінійна функція - це функція, що описується формулою y = kx + b, де x – незалежна змінна, k - кутовий коефіцієнт, число, яке задає кут нахилу графіка, ab – число, яке задає точку перетику графіку з віссю OY. Графіком такої функції є пряма лінія.

Пряма лінія, що проходить через дві точки з координатами (x1, y1) і (x2, y2), може бути описана параметричними рівняннями:

$x = x1 + (x2 – x1) P,$

$y = y1 + (y2 – y1) P.$

При $0 < P < 1$ точка (x, y) лежить всередині відрізка і ділить його у відношенні P / (1-P) (рисунок 2.1)

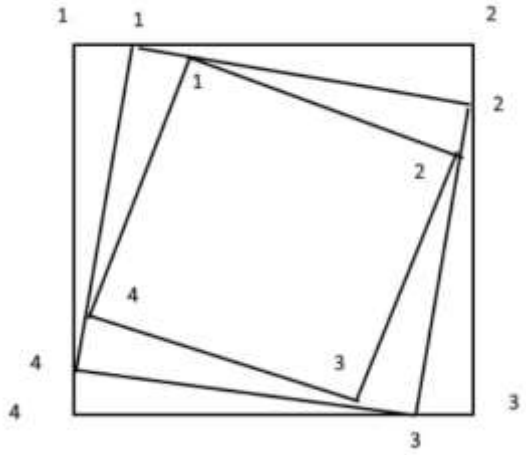


Рисунок 2.1 – Вкладені квадрати

Алгоритм рішення задачі з квадратами:

1. Задати координати вихідного квадрата.
2. Повторити 50 (в умові 50 вкладених квадратів) разів наступні дії:

2.1. Намалювати черговий квадрат.

2.2. За формулами x = x1 + (x2 -x1) P, y = y1 + (y2 - y1) P визначити координати наступного вкладеного квадрата.

Рекурсивна функція - це функція, яка викликає сама себе для розв'язання задачі, поки не буде виконано умову завершення. Умова завершення змусить функцію повернути значення або виконати будь-яку дію, або викликати переповнення стека і збій програми.

Рекурсія дає змогу розбивати складні задачі на підзадачі, а потім розв’язувати їх за допомогою однієї й тієї самої функції. Це дає змогу уникнути використання таких структур, як цикли.

Що необхідно знати для реалізації рекурсивного процесу? З входом в рекурсію здійснюється виклик методу, а для виходу необхідно пам'ятати точку

повернення, тобто місце програми, звідки ми прийшли і куди нам потрібно буде повернутися після завершення методу. Місце зберігання точок повернення називається стеком викликів і для нього виділяється певна область оперативної пам'яті. У цьому стеку запам'ятовуються не тільки адреси точок повернення, а й копії значень всіх параметрів, що повертаються при виклику метода. При рекурсії, за рахунок створення копій параметрів можливе переповнення стека. Це є основним недоліком рекурсивного методу. З іншого боку, рекурсивні методи дозволяють перейти до більш компактного запису алгоритму.

Будь-який рекурсивний метод можна перетворити в звичайний метод. І практично будь-який метод можна перетворити в рекурсивний, якщо виявити рекурентне співвідношення між значеннями які обчислюються в методі.

Існує ще й непряма рекурсія, в якій метод викликає себе в якості допоміжного, не безпосередньо, а через інший допоміжний метод.

Таким чином, при розробці рекурсивного методу слід врахувати його ефективність.

Рекурсія також дає змогу програмістам реалізовувати складні алгоритми, що вимагають повторюваних операцій, які використовуються на побудови, наприклад, трикутника Серпінського.

Трикутник Серпінського, який будується шляхом розбиття трикутника, необов'язково рівностороннього, середніми лініями на чотири подібних трикутника, центрального і трьох кутових, та рекурсивним розбиттям кутових трикутників до отримання елементів бажаного результату (рисунок 2.2).

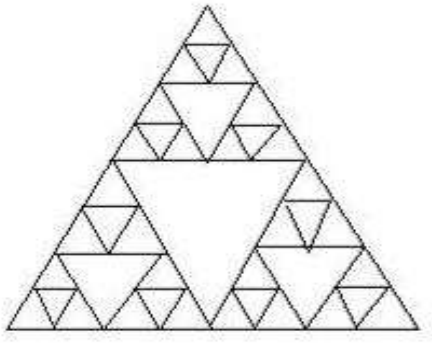


Рисунок 2.2 – Трикутник Серпінського

Перевага використання рекурсії очевидна - без рекурсії побудова такого рисунка, який складається більш ніж з шести рівнів проблематично, а рекурсія дозволяє збільшувати кількість рівнів, не обмежуючись мінімальними розмірами самого нижнього рівня.

Алгоритм рішення задачі трикутника Серпінського:

1. Задати координати вихідного трикутника.
2. Поставити глибину вкладеності, тобто скільки трикутників необхідно намалювати всередині вихідного трикутника.
3. Намалювати вихідний трикутник.
4. За формулами x = x1 + (x2 –x1) P, y = y1 + (y2 – y1) P визначити координати наступного вкладеного трикутника за умови, що P = 0,5.

В якості прикладу рекурсивної функції, можна обчислити факторіал, який знаходиться за формулою n! = 1 * 2 * … * n.

Метод для знаходження факторіалу з рекурсією:

```
int Factorial(int n)
{
    if (n == 1) { return 1; }
```

```
    else { return n * Factorial(n - 1); }
  }
```

Якщо число n дорівнює 1, то повертається 1.

Якщо число n не дорівнює 1, при подальших рекурсивних викликах підфункцій у них передаватиметься щоразу число, менше на одиницю. Таким чином ми дійдемо до ситуації, коли число дорівнюватиме 1, і буде використаний варіант, коли n дорівнює 1.

Метод для знаходження факторіалу без рекурсії:

```
int Factorial(int n)
{
    int result = 1;
    for (int i = 1; i <= n; i++)
  {
      result *= i;
   }
   return result;
  }
```

Задаємо початкове значення для множення result=1. В циклі перебираємо всі числа від 1 до n, при кожній ітерації число збільшується на 1 (1, 2, 3 і т.д.) , На кожній ітерації result множиться на поточне значення i, накопичуючи добуток.

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №3 Візуалізація фракталів

Мета: ознайомитись видами комп'ютерної графіки [1], навчитись будувати та візуалізувати фрактали, використовуючи методи класу Graphics.

Завдання

Побудувати сніжинку Коха.

Теоретичні відомості

Залежно від методу формування зображень, комп'ютерну графіку прийнято ділити на:

- растрову;
- векторну;
- фрактальну.

Растрова графіка – машинна графіка, в якій зображення представляється двовимірним масивом точок (елементів растру), колір та яскравість кожної з яких задається незалежно (рисунок 3.1).

Рисунок 3.1 – Растрове зображення

Переваги растрової графіки

- растрова графіка ефективно представляє реальні образи, так як людське око пристосоване для сприйняття світу, як набору дискретних елементів, які утворюють предмети;
- якісне растрове зображення виглядає реально та природньо.

Недоліки

- великий розмір, займають великий обсяг пам'яті;
- редагування великих растрових зображень потребують великих ресурсів комп'ютера і, отже, потребують більшого часу;
- трудомісткий процес редагування растрових зображень;
- при збільшенні розмірів зображення, погіршується якість.

Растрові зображення використовуються для обробки фотозображень, художньо графіки, реставраційних робіт, роботи зі сканером.

Векторна графіка описує зображення за допомогою математичних формул.

За своєю суттю будь-яке зображення можна розкласти на безліч простих об'єктів, як контури, графічні примітиви і т.д. Будь-який такий простий об'єкт складається з контуру та заливки (рисунок 3.2).

Рисунок 3.2 – Векторне зображення

Основна перевага векторної графіки полягає в тому, що при зміні масштабу зображення, воно не втрачає своєї якості. Звідси випливає й інший висновок - змінюючи розмір зображення засобами векторної графіки, не змінюється розмір файла. Адже формули, що описують зображення, залишаються ті ж самі, змінюється лише коефіцієнт пропорційності.

З іншого боку, такий спосіб зберігання інформації має свої недоліки. Наприклад, якщо робити дуже складну геометричну фігуру (особливо якщо їх багато), то розмір векторного файлу може бути набагато більшим, ніж його растровий аналог, через складність формул, що описують таке зображення. Таким чином, можна зробити висновок, що векторну графіку слід застосовувати для зображень, що не мають великої кількості колірних фонів, напівтонів та відтінків. Наприклад, оформлення текстів, створення логотипів тощо.

Векторну графіку часто називають *об'єктно-орієнтованою графікою* чи креслярською графікою.

Прості об'єкти, такі як кола, лінії, дуги, куби, заповнювачі і т.д., називаються примітивами і використовуються при створенні більш складних об'єктів.

У векторній графіці зображення створюються шляхом комбінації різних об'єктів. Таким об’єктом може бути і растрове зображення, тобто файли векторної графіки можуть містити растрові зображення, як один із типів об'єктів. Такий растровий фрагмент можна, зазвичай, лише масштабувати, але не редагувати.

Існують програми, що підтримують обидва типи об'єктів і дозволяють працювати, як і з растровим так і з векторним зображенням одночасно.

Файли векторної графіки можуть містити декілька різних елементів:

- набори векторних команд;
- таблиці інформації про колір рисунка;
- дані про шрифти, які можуть бути включені в рисунок.

Складність векторних форматів файлів визначається кількістю можливих команд опису об'єктів. Векторні формати файлів можуть відрізнятися способом кодування даних, мати різні колірні можливості. Колір об'єкта зберігається у вигляді його векторного опису.

Переваги векторної графіки

- векторна графіка використовує всі переваги роздільної здатності будь-якого пристрою виведення (використовується максимально можлива кількість точок пристрою), що дозволяє змінювати розміри векторного рисунка без втрати якості;
- векторна графіка дозволяє редагувати окремі частини рисунка, не впливаючи на інші;
- векторні зображення, які не містять растрових об'єктів, мають невеликий розмір.

Недоліки:

- векторні зображення виглядають штучно;
- легко масштабувати, але менше відтінків та півтонів ніж у растровій графіці.

Застосовується в комп'ютерній поліграфії, системах комп'ютерного проєктування, комп'ютерному дизайні та рекламі.

Фрактальна графіка, як і векторна, базується на математичних обчисленнях.

Фрактальна графіка, як і векторна - обчислювана, але відрізняється від неї тим, що об'єкти в пам'яті комп'ютера не зберігаються. Зображення будується за рівнянням або системою рівнянь, тому нічого, окрім формули, зберігати не треба.

Змінивши коефіцієнти рівняння, можна отримати зовсім іншу картину.

Таким чином, будують, як найпростіші регулярні структури, так і складні ілюстрації, що імітують природні ландшафти та тривимірні об'єкти (рисунок 3.3).

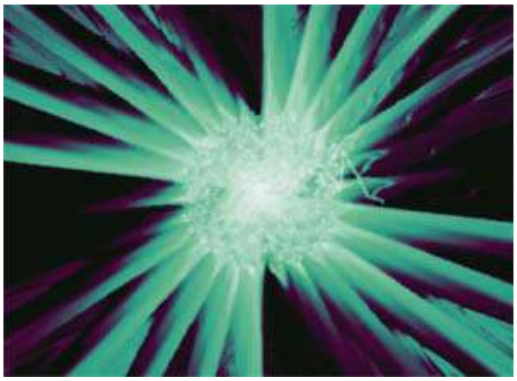


Рисунок 3.3 – Фрактальне зображення

Фрактал - це об'єкт досить складної форми, який можна отримати в результаті виконання простого ітераційного циклу. Рекурсивність і самоподібність - ключові властивості фракталів, де рекурсивний алгоритм створює структуру, в якій частини об’єкта схожі між собою і нагадують цілий об’єкт, в цьому полягає самоподібність. Фрактали будуються через повторення дій чи функцій, і саме ця рекурсивна побудова призводить до появи самоподібності. Завдяки цій властивості можна використовувати фрактал для генерування поверхні місцевості, яка схожа на саму себе, незалежно від масштабу, в якому вона відображена.

Крива Коха – це фрактальна крива, не має дотичних, тобто ніде не диференційовна, при цьому всюди неперервна. Три копії кривої Коха, побудовані на сторонах правильного трикутника, утворюють замкнену криву, так звану сніжинку Коха. Крива Коха стає фракталом в результаті нескінченної кількості ітерацій, в ході яких виконується розподіл кожного відрізка прямої на три частини. Послідовне виконання трьох ітерацій перетворюють лінію на об’єкт, схожий на сніжинку.

Для побудови сніжинки Коха потрібно виконати наступні операції (рисунок 3.4). Розглянемо в якості нульовій ітерації рівносторонній трикутник.

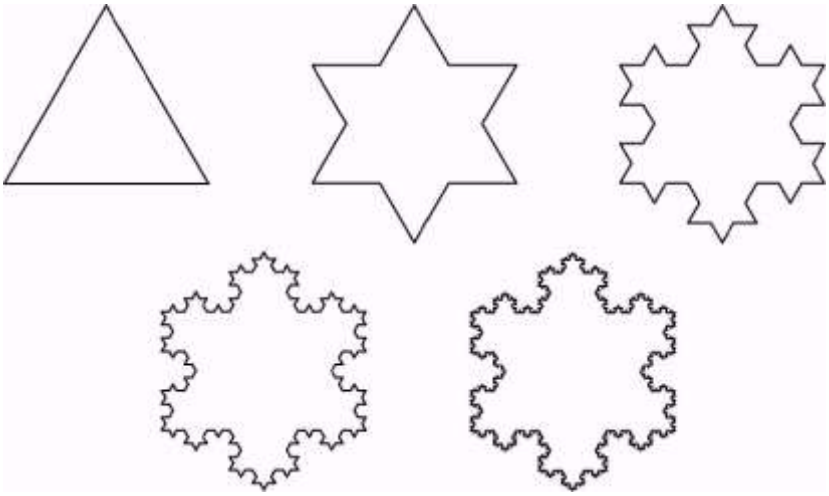
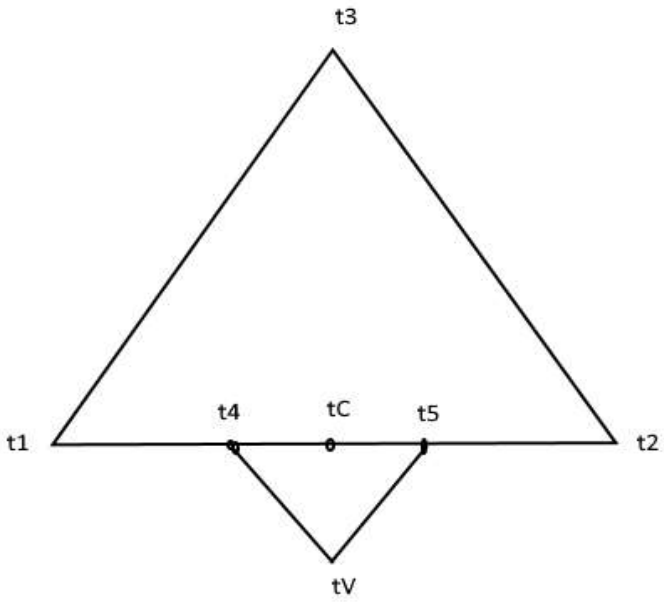


Рисунок 3.4 – Сніжинка Коха

Потім кожну зі сторін цього трикутника розділимо на три рівні частини, приберемо середню частину і в середині добудуємо рівносторонній трикутник (рисунок 3.5).



$$t4(x,y) = (t2(x,y) + 2*t1(x,y)) / 3$$

$$t5(x,y) = (2*t2(x,y) + t1(x,y)) / 3$$

$$tC(x,y) = (t1(x,y) + t2(x,y)) / 2$$

$$tV(x,y) = (4*tC(x,y) - t3) / 3$$

Рисунок 3.5 – Побудова сніжинки Коха

На наступному кроці, такою ж процедурою розподілу на три рівні частини і добудовуванню рівностороннього трикутника, піддається кожна зі сторін нової фігури, і так до нескінченності. В результаті виникає симетрична, схожа на сніжинку, нескінченно зламана крива, яка представляє собою самоподібну фігуру, яка називається сніжинкою Коха.

Алгоритм рішення задачі:

1. Задати координати вихідного рівностороннього трикутника.
2. Поставити глибину вкладеності, тобто скільки вершин матиме сніжинка.
3. Намалювати вихідний трикутник.
4. За формулами (рисунок 3.2) визначити координати вершин наступного опуклого трикутника

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №4 Переміщення зображення по довільній траєкторії. Переміщення зображення за допомогою клавіатури. Переміщення зображення за допомогою маніпулятора миші

Мета: навчитись реалізовувати переміщення об’єкта по таймеру, та його властивості, переміщення зображення за допомогою клавіатури, переміщення зображення за допомогою маніпулятора миші.

Завдання

- намалювати або підключити об'єкт (зображення);
- змінити координати зображення і вивести його в новому місця і так далі;
- відновити екран в місці, де було зображення;
- реалізувати безперервну траєкторію руху зображення;
- розташувати на екрані об'єкти-перешкоди, і забезпечити відбивання від них зображення, яке рухається;
- додати зображення та забезпечити управління переміщенням і перемиканням між зображеннями за допомогою клавіатури;
- додати зображення та забезпечити управління переміщенням і перемиканням між зображеннями за допомогою маніпулятора миші.

Теоретичні відомості

Для малювання зображення використовується метод **DrawImage (Image, Point)** [4], використовуючи його вихідний фактичний розмір, у вказаному місці.

```
public void DrawImage(Image image, Point point)
{
    Image myImage = Image.FromFile("1.jpg"); // Створення зображення
```

```
Point pointMyImage = new Point(100, 100); // Створення точки,
```
верхнього лівого кута зображення
```
e.Graphics.DrawImage(myImage, pointMyImage); // Промальовка
```
зображення
```
}
```

Для того, щоб задати безперервну траєкторію руху зображення можна використати компонент Windows Forms Timer, який викликає подію через певні проміжки часу (Класс System.Windows.Forms.Timer).

Щоб використати Timer, потрібно додати його на форму, вказати значення властивості Interval (в мілісекундах) для таймера. Ця властивість визначає скільки часу пройде до моменту повторного запуску процедури.

Також можна використовувати програмно таймер під час виконання програми:

```
static System.Windows.Forms.Timer time = new
System.Windows.Forms.Timer();
    public Form1()
{
   time.Interval = 70;
   time.Tick += Timer;
   time.Start();
   InitializeComponent();
}
```

Та написати необхідний код у обробці подій Tick. Код, написаний у цій події, виконуватиметься з інтервалом, зазначеним у властивості Interval.

Переміщення

Структури Point та PointF - для роботи з точками у просторі. Ці структури еквівалентні двомірному вектору. Вони містять дві властивості X, Y читання-

запису, які представляють горизонтальне та вертикальне зміщення від певного місця.

Оголошення point та ініціалізація значень x, y:

```
PointF point = new PointF(10, 10);
```

Оголошення point без ініціалізації значень x, y:

```
PointF point = new PointF();
```

Присвоєння значень X та Y для точки point:

```
point.X = 20;
point.Y = 20;
```

Приклад організації безперервного руху зображення:

```
…
Graphics graph;
PointF point;
PointF pointNext;
int x = 4;
int y = 4;
…
    private void Move_Tick(object sender, EventArgs e)
    {
       point = pointNext;
       pointNext = new PointF(point.X + x, point.Y + y);
       if (pointNext.X <= 1 || pointNext.X >= 500)
       {
          x = -x;
       }
       if (pointNext.Y <= 1 || pointNext.Y >= 400)
       {
          y = -y;
```

```
        }
        Refresh(); // метод, який змушує форму перемальовувати себе та всі його дочірні елементи
        graph = panel1.CreateGraphics();
        Image img = Image.FromFile("D:\\111\\111.png");
        graph.DrawImage(img, point);
    }
```

Переміщення зображення за допомогою клавіатури та маніпутора миші

Подія це ситуація, при виникненні якої відбудеться дія або кілька дій. Подія має повідомити клас, інший об'єкт або користувача про цю дію.

З точки зору програми подія являє собою блок коду, який буде виконуватися при здійсненні певної дії. Цей код буде називатися обробкою події. Базовим класом для подій є вбудований клас **EventArgs**.

Створюючи windows-застосунки, ми постійно стикаємося з обробкою подій, пов'язаних з елементами форми.

Наприклад, при додаванні коду, який буде виконуватися після кліку на кнопці Button, автоматично формується подія Click. При цьому з'являвся такий каркас, в який вписується потрібний код - тобто код, який буде виконуватися після натискання кнопки:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Це метод, який відповідає за подію, називається обробником події. У цього методу існує два обов'язкових параметра: **object sender** і **System.EventArgs e**. Ці параметри створюються автоматично. Sender - це посилання на об'єкт, який викликав подію; e - об'єкт, зв'язаний з подією, посилаючись на нього, можна отримати його властивості, наприклад, розташування миші або значення даних. Код, необхідний для підписки на подію, генерується автоматично:

button1.Click += new EventHandler(this.button1_Click);

Події, пов'язані з клавіатурою: KeyDown, KeyPress, KeyUp.

public event KeyEventHandler KeyDown - відбувається при натисканні клавіші, коли фокус на цьому елементі.

public event KeyEventHandler KeyUp - відбувається при відтисканні клавіші, коли фокус на цьому елементі.

public event KeyPressEventHandler KeyPress - відбувається при натисканні клавіші, якщо елемент керування має фокус.

Події, пов'язані з маніпулятором миші: Click, DoubleClick, MouseDown, MouseUp, MouseEnter, MouseMove, MouseLeave.

public event EventHandler Click - відбувається при натисканні елемента управління.

public event EventHandler DoubleClick - відбувається при натисканні елемента управління.

Подвійне клацання визначається параметрами миші в операційній системі користувача. Користувач може встановити час між натисканнями кнопки миші, який слід розглядати як подвійне клацання, а не два кліки.

public event MouseEventHandler MouseDown - відбувається, коли курсор миші знаходиться на елементі управління і кнопка миші натиснута.

public event MouseEventHandler MouseUp - відбувається, коли курсор миші знаходиться на елементі управління і кнопка миші відтиснута.

public event EventHandler MouseEnter - відбувається, коли курсор миші входить в елемент управління.

public event MouseEventHandler MouseMove - відбувається при переміщенні курсору миші на елемент управління.

public event EventHandler MouseLeave - відбувається, коли курсор миші покидає елемент управління.

Графічні об'єкти можна будувати за допомогою методів: DrawLine, DrawRectangle, DrawEllipse, DrawCurve, DrawBezier, FillRectangle, FillPolygon, FillEllipse та ін.

Рогзлянемо структуру Rectangle, простір імен: System.Drawing.

Структура містить набір із чотирьох цілих чисел, що визначають розташування та розмір прямокутника.

Конструктори:

Rectangle(Int32, Int32, Int32, Int32) - ініціалізує новий екземпляр класу Rectangle заданим розташуванням та розміром.

Rectangle(Point, Size) - ініціалізує новий екземпляр класу Rectangle заданим розташуванням та розміром.

e.Graphics.DrawRectangle(new Pen(Color.Red), 10, 10, 200, 100);

Деякі властивості:

Height/Width - повертає або задає висоту/ширину у структурі Rectangle.

Left - повертає координату по осі X лівого краю структури Rectangle.

Right - повертає координату по осі X, що є сумою значень властивостей X та Width даної структури Rectangle.

Location - повертає або задає координати верхнього лівого кута структури Rectangle.

X/Y - повертає або визначає координату по осі X/Y лівого верхнього кута структури Rectangle.

Деякі методи:

DrawRectangle(Rectangle). Прямокутник визначається за його шириною Width і висотою Height від лівого верхнього кута, представленого властивістю Location.

Intersect(Rectangle) - замінює об'єкт Rectangle його перетином із зазначеним прямокутником Rectangle.

Intersect(Rectangle, Rectangle) - повертає третю структуру Rectangle, що є перетином двох інших структур Rectangle. Якщо перетин відсутній, повертається порожня структура Rectangle.

IntersectsWith(Rectangle) - визначає, чи цей прямокутник перетинається з прямокутником rect.

Приклад використання методу IntersectsWith(Rectangle) для визначення перетину двох об’єктів:

```
        …
public partial class FormMove : Form
{
    Rectangle rctMove = new Rectangle(0, 0, 50, 50);
    Rectangle rctMove1 = new Rectangle(570, 0, 50, 50);
    private void FormMove_Paint(object sender, PaintEventArgs e)
      {
        Graphics graph = e.Graphics;
        graph.FillRectangle(Brushes.HotPink, rctMove);
        graph.FillRectangle(Brushes.Pink, rctMove1);
      }
     private void FormMove_KeyDown(object sender, KeyEventArgs e)
     {
        if (e.KeyCode == Keys.Left)
        {
           rctMove.X -= 5;
        }
       …
        if (rctMove.IntersectsWith(rctMove1))
```

```
{
   rctMove.X -= 20;
   rctMove.Y -= 20;
 }
```

Invalidate(); // метод, коли щось потрібно перемалювати. Він визначає область клієнтського вікна як недійсного і тому потребує перемальовки, а потім забезпечує ініціювання події Event.

Для більш складних фігур використовується об'єкт Region, клас GraphicsPath.

Region описує внутрішню частину графічної фігури, що складається з прямокутників та контурів.

```
Region region = new Region(new Rectangle(10, 10, 100, 100));
```

GraphicsPath ініціалізує новий екземпляр класу GraphicsPath із заданими масивами Point та PathPointType, де PathPointType визначає тип точки в об'єкті GraphicsPath.

```
public GraphicsPath(System.Drawing.Point[] pts, byte[] types);
```

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу.

ЛАБОРАТОРНИЙ ПРАКТИКУМ №5 Побудова та переміщення графічних об'єктів на площині

Мета: ознайомитись з матрицями перетворення об’єктів на площині, матрицею перетворення в однорідних координатах. навчитись їх застосовувати для перетворення об’єкту, здійснювати перетворення об’єктів, використовуючи клас Matrix.

Завдання

Побудувати геометричну фігуру та матрицю перетворень будь-якої геометричної фігури, здійснити її переміщення, обертання, віддзеркалення, масштабування на екрані.

Вивести на екран будь-який текст і здійснити його переміщення.

Теоретичні відомості

Можливість перетворення точок та ліній є основою комп'ютерної графіки. При використанні комп'ютерної графіки можна змінювати масштаб зображення, обертати його, зміщувати і трансформувати для поліпшення наочності зображення об'єкта [1].

Необхідно навчитися керувати зображенням на екрані, вносити зміни до його положення, форми, орієнтації, розміру.

Всі маніпуляції з об’єктом (зображенням) можна здійснити, використовуючи математичний апарат. Для цього існують спеціальні геометричні перетворення, які дозволяють змінювати характеристики об'єктів на площині.

Точка представляється на площині двома своїми координатами, які визначаються як елементи матриці [x y]. Точка може задаватися у вигляді

вектора-стовпця або вектора-рядка у двовимірному просторі, які називаються координатним вектором.

Для формування такого вектора використовується матриця-рядок, тобто безліч точок, кожна з яких визначає координатний вектор деякою системою вимірювання.

Ця множина зберігається в комп'ютері у вигляді матриці або масиву чисел. Положенням точок можна керувати шляхом маніпулювання відповідною матрицею. Лінії, що з'єднують точки, формують відрізки, криві та картинки.

Правила матричної алгебри визначають допустимі операції над елементами. Багато фізичних задач зручно виражаються в матричному вигляді.

Для моделей фізичних систем задача зазвичай ставиться так: дано матриці [A] і [B]. Потрібно знайти результуючу матрицю [T], таку, що [A] [T] = [B] . У цьому випадку рішенням є матриця $[T]= [A]^{-1}[B]$, де $[A]^{-1}$ - матриця, обернена до матриці [A].

Водночас матрицю [T] можна інтерпретувати як геометричний оператор. Для виконання геометричного перетворення точок, представлених векторами положень у матриці [A] використовується множення матриць.

Припустимо, що матриці [A] і [T] відомі. Потрібно визначити елементи матриці [B].

Представлення [T] як геометричного оператора є основою математичних перетворень, що використовуються в машинній графіці.

Перетворення точок

Розглянемо результати множення матриці [x y] , яка містить координати точки P на матрицю загального перетворення розміром 2x2:

$$[X][T]=\begin{bmatrix} x & y \end{bmatrix}\begin{bmatrix} a & b \\ c & d \end{bmatrix}=\begin{bmatrix} (ax+cy)(bx+dy) \end{bmatrix}=\begin{bmatrix} x^* & y^* \end{bmatrix}$$

Цей запис означає, що вихідні координати точки x та y перетворюються на x* та y*, $x^* = ax+cy$ $y^* = bx+dy$ де, .

Представляють інтерес значення x*, y* - координати результуючої, перетвореної точки P.

Розглянемо деякі особливі випадки.

При *a=d=1* і *c=b=0* перетворення зведеться до одиничної матриці та координати точки P залишаться незмінними:

$$[X][T]=[x \quad y]\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}=[x \quad y]=[x^* \quad y^*]$$

Як і слід очікувати, у лінійній алгебрі множення на одиничну матрицю еквівалентно множенню на 1 у звичайній алгебрі.

У випадку *d=1* і *c=b=0*

$$[X][T]=[x \quad y]\begin{bmatrix}a & 0\\ 0 & 1\end{bmatrix}=[ax \quad y]=[x^* \quad y^*]$$

де $x^* = ax$ – результат масштабування координати x (рисунок 5.1).

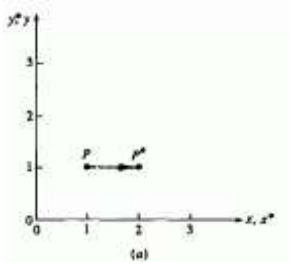


Рисунок 5.1 – Масштабування координати x

У випадку *c=b=0*

$$[X][T]=[x \quad y]\begin{bmatrix}a & 0\\ 0 & d\end{bmatrix}=[ax \quad dy]=[x^* \quad y^*]$$

Дане перетворення призводить до зміни обох координат ***x*** та ***y*** вектора P (рисунок 5.2).

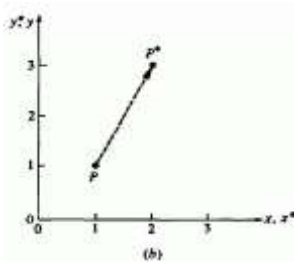


Рисунок 5.2 – Масштабування координат x та ***y***

Якщо $a \neq d$, то координати масштабуються по різному.

Якщо $a = d > 1$ відбувається розтягування вектора P або масштабування координат.

Якщо $0 < a = d < 1$, то має місце стиснення.

Якщо значення a < 0 або d < 0 від'ємне, вектор віддзеркалиться відносно координатних осей або площини.

Візьмемо $b = c = 0 \quad d = 1 \quad a = -1$

$$[X][T] = [x \quad y]\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} = [-x \quad y] = [x^* \quad y^*]$$

і в результаті отримуємо симетричне віддзеркалення відносно осі Y (рисунок 5.3).

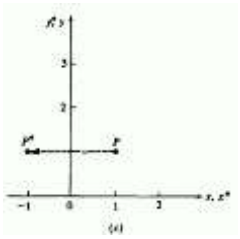


Рисунок 5.3 – Віддзеркалення відносно осі Y

Якщо, $b = c = 0 \quad a = 1 \quad d = -1$

то виконується симетричне віддзеркалення відносно осі X.

Якщо $b = c = 0 \quad a = d < 0$, де $a = -1 \quad d = -1$

відбувається віддзеркалення відносно початку координат (рисунок 5.4).

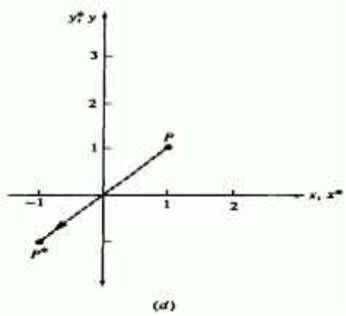


Рисунок 5.4 – Віддзеркалення відносно початку координат

Обидві операції: віддзеркалення і масштабування залежать тільки від діагональних членів матриці перетворення.

Розглянемо тепер випадок з недіагональними членами.

Візьмемо спочатку значення $a = d = 1 \quad c = 0$ тоді:

$$[X][T] = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} x & (bx + y) \end{bmatrix} = \begin{bmatrix} x^* & y^* \end{bmatrix}$$

Бачимо, що координата x точки P залишилася незмінною, тоді як координата y* лінійно залежить від початкових координат. Дане перетворення називається зміщенням (рисунок 5.5).

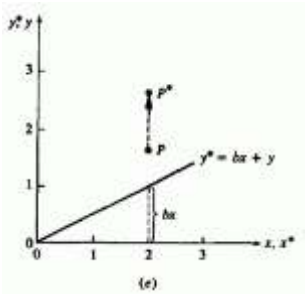


Рисунок 5.5 – Зміщення пропорційно координаті x

Аналогічно у випадку, коли $a = d = 1 \qquad b = 0$ перетворення призведе до зміщення пропорційно координаті **y** (рисунок 5.6).

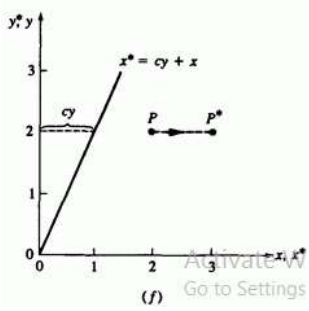


Рисунок 5.6 – Зміщення пропорційно координаті y

Розберемо дію загального перетворення, заданого виразом:

$$\begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} (ax+cy) & (bx+dy) \end{bmatrix} = \begin{bmatrix} x^* & y^* \end{bmatrix}$$

коли початковий вектор лежить у точці початку координат, тобто:

$$\begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 0 & 0 \end{bmatrix} = \begin{bmatrix} x^* & y^* \end{bmatrix}$$

Видно, що початок координат інваріантний відносно перетворення загального виду. Це обмеження не діє з використанням однорідних координат.

Перетворення ліній

Пряму лінію можна визначити за допомогою двох векторів, які задають координати кінцевих точок. Розташування та напрямок лінії, що з'єднує дві точки, може змінюватися в залежності від положень векторів.

Розглянемо лише математичні операції над кінцевими точками лінії.

На рисунку 5.7 зображено пряму лінію, яка проходить між двома точками A і B. Положення векторів точок A і B задається наступним чином:

$$\begin{bmatrix} A \end{bmatrix} = \begin{bmatrix} 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} B \end{bmatrix} = \begin{bmatrix} 2 & 3 \end{bmatrix}$$

Розглянемо матрицю перетворення яка призводить до зміщення зображення:

$$T = \begin{bmatrix} 1 & 2 \\ 3 & 1 \end{bmatrix}$$

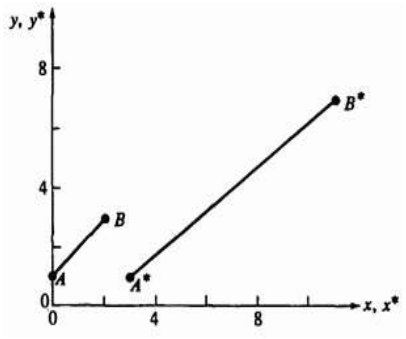


Рисунок 5.7 – Перетворення лінії

Перетворення векторів А і В за допомогою матриці дає нове положення векторів А* і В*:

$$[A][T]=[0 \quad 1]\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}=[3 \quad 1]=\left[A^*\right]$$

$$[B][T]=[2 \quad 3]\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}=[11 \quad 7]=\left[B^*\right]$$

Таким чином, результуючі координати для точки А* - це х*=3, у*=1. Аналогічно, В*- нова точка з координатами х*=11, у*=7.

У більш компактному вигляді відрізок *AB* може бути представлений матрицею розміром (2х2): $L=\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}$

Помножимо цю матрицю на [Т]:

$$[L][T]=\begin{bmatrix}0 & 1\\ 2 & 3\end{bmatrix}\begin{bmatrix}1 & 2\\ 3 & 1\end{bmatrix}=\begin{bmatrix}3 & 1\\ 11 & 7\end{bmatrix}=\left[L^*\right]$$

де компоненти **[**L*] являють собою перетворення координат векторів **[А*]** і **[**В*]**. В результаті перетворення А в А* і В в В*, де х, у - це початкові осі координат, а х*, у*- перетворені осі. З рисунка 5.7 видно, що перетворення зміщення збільшує довжину відрізка та змінює його напрямок.

При перетворенні лінії, шляхом множення на матрицю, гарантується відповідність всіх її точок. Середня точка лінії перетворюється в середня точку.

Застосовуючи ці результати в машинній графіці, будь-яка пряма може бути перетворена в будь-яку іншу пряму шляхом простого перетворення її кінцевих точок і відновлення лінії між ними. Паралельні лінії зберігають паралельність і після перетворення. В лініях, які перетинаються, точка перетину перетвориться в точку перетину перетворених ліній. При перетворенні ліній, які перетинаються і перпендикулярні, перпендикулярність, в загальному випадку, не зберігається. Але при повному оберті (детермінант матриці перетворення = 1) кути між прямими, що перетинаються, зберігаються. Цей результат поширюється також і операцію віддзеркалення, ортогональна матриця якого має визначник -1. Перетворення ліній, які перетинаються, включає в себе обертання, віддзеркалення і масштабування.

Більш складні об’єкти можна представити за допомогою точок і ліній і здійснювати перетворення за допомогою матриць перетворення.

Існує кілька окремих випадків перетворень, використовуючи які досягаються необхідні перетворення на площині. Такими перетвореннями є:

- зміщення;
- обертання;
- віддзеркалення;
- масштабування.

Обертання

Перетворення обертання навколо точки початку координат на довільний кут, задається матрицею:

$$[T] = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Оберти є додатними, якщо вони здійснюються проти годинникової стрілки відносно точки обертання.

Визначник загальної матриці оберту має наступний вигляд:

$$\det[T] = \cos^2\theta + \sin^2\theta = 1$$

У загальному випадку перетворення по матриці з детермінантом, рівним 1, призводять до повного оберту.

Для представлення даних і виконання перетворень за допомогою множення матриць використовуються різні погодження.

Найбільшу увагу потрібно приділяти формулюванню завдань та інтерпретації результатів.

Наприклад, перед виконанням обертання необхідно отримати відповіді на наступні питання:

У правосторонній або лівосторонній системі координат визначаються координатні вектори, які обертаються?

– обертається об'єкт або система координат?

– як визначаються додатній і від’ємний оберт?

– координати записуються у вигляді рядка або стовпця матриці?

– навколо якої лінії або осі здійснюється оберт?

В нашому випадку:

– використовується правостороння система координат;

– об'єкт обертається в нерухомій системі координат;

– додатній оберт проти годинникової стрілки, від’ємний – за годинниковою стрілкою (спостереження з точки початку координат в додатному напрямку координатної осі);

– координатні вектори подаються у вигляді рядка матриці.

Тому наша матриця обертання задає перетворення для додатного оберту навколо початку координат або осі Z.

Так як, вектор задається рядком матриці, то матрицю перетворення слід розмістити після даних або матриці координатних векторів. Це перетворення

задається шляхом множення справа. Операція множення матриць не комутативна, топу важливий порядок матриць.

Віддзеркалення

У той час як повний оберт на площині ху зазвичай здійснюється в двовимірному просторі відносно нормалі до площини, віддзеркалення являє собою той же поворот на кут 180^0 в тривимірному просторі і назад на площину відносно осі, яка лежить в площині ху. На рисунку 5.8 приклади двох відображень на площині трикутника:

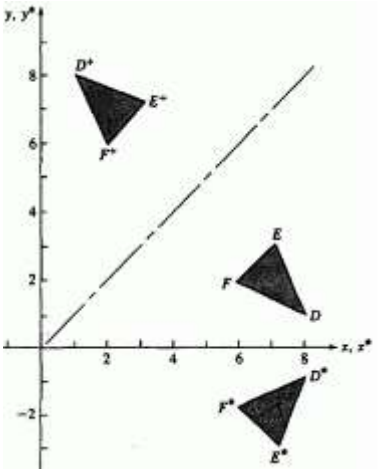


Рисунок 5.8 – Віддзеркалення

Віддзеркалення відносно прямої y=x здійснюється за допомогою матриці:

$$[T]=\begin{bmatrix}0 & 1\\ 1 & 0\end{bmatrix}$$

Віддзеркалення відносно прямої y=-x матиме вигляд:

$$[T]=\begin{bmatrix}0 & -1\\ -1 & 0\end{bmatrix}$$

Віддзеркалення відносно осі X матиме вигляд:

$$[T]=\begin{bmatrix}1 & 0\\ 0 & -1\end{bmatrix}.$$

Віддзеркалення відносно осі Y матиме вигляд:

$$[T]=\begin{bmatrix}-1 & 0\\ 0 & 1\end{bmatrix}.$$

У кожної з цих матриць детермінант дорівнює -1. У загальному випадку, якщо детермінант матриці перетворення дорівнює -1, то перетворення дає повне віддзеркалення.

Якщо обидва повних віддзеркалення здійснюються послідовно відносно прямих, що проходять через початок координат, то результатом буде повний оберт відносно початку координат.

Масштабування

Величина масштабування визначається значенням елементів вихідної діагональної матриці:

$$[T]=\begin{bmatrix}a & 0\\ 0 & d\end{bmatrix}$$

Якщо $a = d \quad b = c = 0$ виконується пропорційне масштабування;

Якщо $a \neq d \quad b = c = 0$ масштабування буде проведено непропорційно.

В першому випадку для $a = d > 1$ відбувається розширення.

Якщо $a = d < 1$ відбувається рівномірне стиснення.

Непропорційне розширення і стиснення виникають в залежності від значень a та d, які можуть бути менше або більше 1, незалежно один від одного.

Якщо матриця:

$$[T]=\begin{bmatrix}2 & 0\\ 0 & 2\end{bmatrix}$$

то має місце «двократне» розширення або рівномірне масштабування відносно точки початку координат.

Якщо матриця:

$$[T]=\begin{bmatrix} 3 & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$$

по осі X буде «трикратне» розширення, по осі Y буде «двократне» стиснення.

За допомогою матричних операцій над координатними векторами, що визначають вершини фігур, можна управляти формою і положенням поверхні. Однак для отримання бажаної орієнтації може знадобитися більше одного перетворення. Так як операція множення матриці не комутативна, то важливий порядок застосування перетворень:

$$[T_2][T_1] \neq [T_1][T_2]$$

Ми розглянули ряд перетворень, що здійснюються за допомогою (2*2) - матриці загального перетворення.

Серед них обертання, віддзеркалення, масштабування, зміщення.

Зазначалося, що вихідна система координат інваріантна стосовно всіх перетворень.

Однак виникає необхідність змінювати положення початку координат, тобто перетворювати кожну точку на площині.

Цього можна досягти шляхом переміщення точки початку координат або будь-якої іншої точки на площині:

$$x^* = ax + cy + m$$

$$y^* = bx + dy + n$$

константи переміщення ***m*** та ***n*** не можна ввести в (2*2) - матрицю перетворення, так як це не простір. Це можна подолати, використовуючи однорідні координати. Однорідні координати неоднорідного координатного вектора [x,y] представляють собою трійку $[x' \quad y' \quad h]$, де, $x = x'/h \quad y = y'/h$ а h - деяке дійсне число. Випадок $h=0$ є особливим.

Завжди існує один набір однорідних координат вигляду [x y 1]. Ми вибрали цю форму, щоб подати координатний вектор [x,y] на фізичній площині xy. Всі інші однорідні координати подаються у вигляді [*hx hy h*]

Дані координати не зберігають однозначності, наприклад, всі наступні координати представляють фізичну точку (3, 2): [6 4 2] [12 8 4] [3 2 1].

Матриця перетворення для однорідних координат має розмір (3*3):

$$[T]=\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ m & n & 1 \end{bmatrix}$$

де дія елементів ***a, b, c, d*** верхньої частини (2*2) - матриці точно відповідає діям, розглянутим раніше. елементи ***m***, ***n*** є коефіцієнтами переміщення в напрямках ***x*** та ***y*** відповідно.

Повна матриця перетворення має вигляд:

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix}=\begin{bmatrix} x & y & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}=\begin{bmatrix} x+m & y+n & 1 \end{bmatrix}$$

Таким чином, кожна точка площини і навіть початок координат ***x=y=0*** тепер можуть бути перетворені. Таким чином, кожна точка площини і навіть початок координат x=y=0 тепер можуть бути перетворені.

<u>Обертання</u> вектора [x y 1] навколо точки ***m, n*** на довільний кут можна здійснити в такий спосіб:

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix}=\begin{bmatrix} x & y & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -m & -n & 1 \end{bmatrix}\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ m & n & 1 \end{bmatrix}$$

Віддзеркалення відносно довільної прямої

можна зробити, скориставшись процедурою, аналогічною обертанню навколо довільної точки.

Виконуються наступні дії:

– переміщення лінії і об'єкта таким чином, щоб лінія пройшла через початок координат;

– обертання лінії і об’єкту навколо точки початку координат до збігу з однією з координатних осей;

– віддзеркалення відносно координатної осі;

– зворотній оберт навколо початку координат;

– переміщення в початкове положення.

У матричному вигляді дане перетворення має вигляд

$$[T] = [T'][R][R'][R]^{-1}[T']^{-1}$$

де *T’* - матриця переміщення, *R* - матриця обертання навколо початку координат, *R’* - матриця віддзеркалення.

Переміщення, оберти і віддзеркалення також застосовуються для перетворення довільних фігур.

Матрицю перетворення розміром 3*3 для двовимірних однорідних координат можна розбити на чотири частини:

$$[T] = \begin{bmatrix} a & b & \vdots & p \\ c & d & \vdots & q \\ \cdots & \cdots & \cdots & \cdots \\ m & n & \vdots & s \end{bmatrix}$$

Коефіцієнти **a, b, c, d** - коефіцієнти масштабування, обертання, віддзеркалення і зсуву відповідно.

Елементи **m** та **n** задають переміщення.

Ми розглядали випадки, при яких коефіцієнти мали значення **p=q=0, s=1**.

При p=q=0 та s=1 однорідні координати перетворених векторів завжди рівні h=1. Геометрично цей результат інтерпретується як обмеження перетворення фізичною площиною h=1.

Для ілюстрації ефекту перетворення при p та q, відмінних від нуля, розглянемо такий вираз:

$$\begin{bmatrix} X & Y & h \end{bmatrix} = \begin{bmatrix} hx & hy & h \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & p \\ 0 & 1 & q \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & (px+qy+1) \end{bmatrix}$$

В даному виразі $X=hx$, $Y=hy$, $h=px+qy+1$. Перетворений координатний вектор, виражений в однорідних координатах, лежить тепер в тривимірному просторі, визначеному як $h=px+qy+1$. Перетворення показано на рисунку 5.9, де відрізок AB, що належить фізичній площині $h=1$, перетворюється в CD зі значенням $h \neq 1$, тобто $h=px+qy-h+1=0$

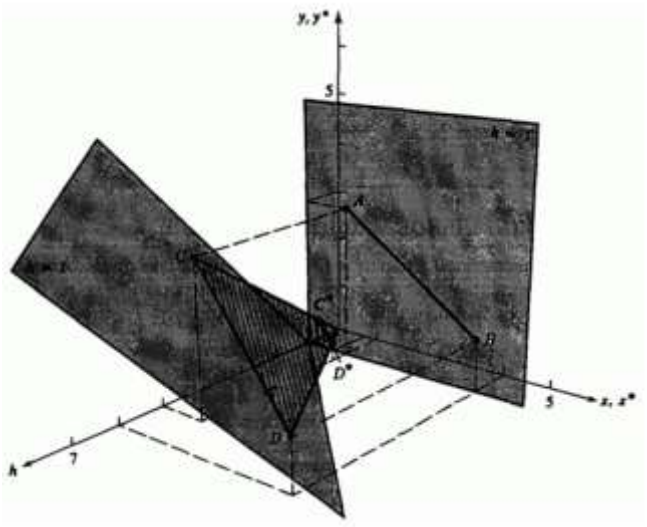


Рисунок 5.9 – Перетворення відрізка з фізичної площини $h=1$ на площину h не дорівнює 1 і проєкціювання назад на фізичну площину

Але, становлять інтерес результати, що належать фізичній площині $h=1$, які можна отримати шляхом геометричного проєкціювання прямої CD з $h \neq 1$ площини назад на площину $h=1$ *з* використанням для цього променів проєкціювання, що проходять через початок координат. Відбувається

зменшення прямої C*D*. Використовуючи правило подібності трикутників, отримаємо:

$$x^* = \frac{X}{h} \qquad y^* = \frac{Y}{h}$$

Або в однорідних координатах:

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} \frac{X}{h} & \frac{Y}{h} & 1 \end{bmatrix}$$

Після цього, нормалізуємо вираз [*x y (px+qy+1)*] розподілом однорідних координат на величину ***h***, знаходимо:

$$\begin{bmatrix} x^* & y^* & 1 \end{bmatrix} = \begin{bmatrix} \frac{X}{h} & \frac{Y}{h} & 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{px+qy+1} & \frac{y}{px+qy+1} & 1 \end{bmatrix}$$

Або

$$x^* = \frac{X}{h} = \frac{x}{px+qy+1} \qquad y^* = \frac{Y}{h} = \frac{y}{px+qy+1}$$

Залишається елемент ***s*** матриці перетворення (3*3) – він відповідає пропорційному масштабуванню, при якому всі компоненти вектора змінюються пропорційно. Покажемо це, розглянувши наступне перетворення:

$$\begin{bmatrix} X & Y & h \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{bmatrix} = \begin{bmatrix} x & y & s \end{bmatrix}$$

де *X=x, Y=y n, h=s*. Після нормалізації отримаємо *X*=x/s, Y*=y/s*. Таким чином, перетворення *[x y 1][T]=[x/s y/s 1]* є рівномірним масштабуванням координатного вектора. Якщо s < 1, то відбувається розтягнення, а якщо s > 1, то маємо стиснення.

Перетворення здійснюється також в площині *h=1*. тут *h = s = const*, і тому площина $h \neq 1$ паралельна площині *h=1*. Геометрична інтерпретація даного ефекту показана на рисунку 5.10:

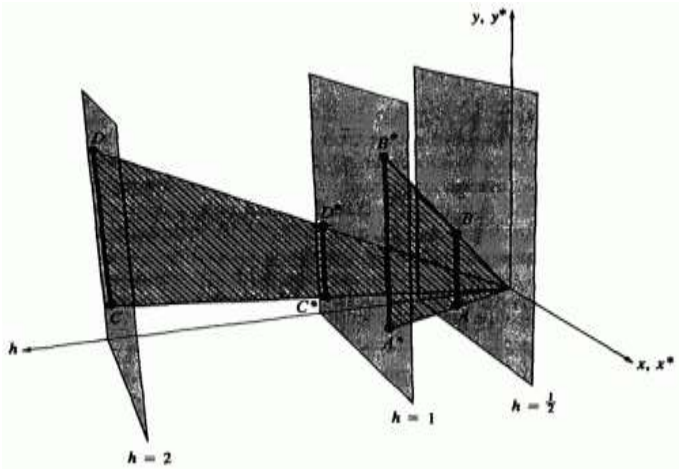


Рисунок 5.10 – Геометрична інтерпретація пропорційного масштабування

Однорідні координати надають зручний і ефективний спосіб перенесення точок з однієї системи координат в відповідні точки альтернативної координатної системи.

Нескінченна область в одній системі координат часто перетворюється в скінченну область в альтернативній системі.

Вектор з однорідною компонентою *h=0* представляє точку нескінченності і може бути також інтерпретований як рух до ліміту (рисунок 5.11):

h	x^*	y^*	X	Y
1	4	3	4	3
1/2	8	6	4	3
1/3	12	9	4	3
⋮				
1/10	40	30	4	3
⋮				
1/100	400	300	4	3
⋮				

Рисунок 5.11 – Випадок *h=0*

В .NET є клас System.Drawing.Drawing2D.Matrix для перетворення координат за допомогою матриці, яка задає афінне перетворення з матричним перетворенням 3 на 3 Афінні перетворення – це лінійні перетворення, які

супроводжуються перенесенням об’єктів: масштабування, обертання, віддзеркалення, переміщення. Для афінного перетворення останній стовпець в узагальненій матриці перетворення розміром 3×3 повинен бути (0 0 1).

В C# для використання класу Matrix, необхідно підключити простори імен:

using System.Drawing;

using System.Drawing.Drawing2D;

У GDI+ можна зберегти афінне перетворення в об'єкті Matrix. Оскільки третій стовпець матриці, що представляє афінне перетворення, завжди має значення (0, 0, 1), при створенні об'єкта у перших двох стовпчиках вказується лише шість чисел Matrix.

Оператор Matrix myMatrix = new Matrix(0, 1, -1, 0, 3, 4) створює матрицю (рисунок 5.12):

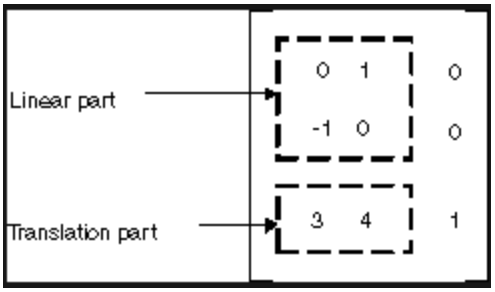


Рисунок 5.12 – Матриця перетворень

Комбіноване перетворення - це послідовність декількох перетворень. Розглянемо матриці перетворення:

– матриця A - обертання на 90 градусів
– матриця B - масштабування з коефіцієнтом 2 в напрямку осі x
– матриця C – переміщення на 3 одиниці по осі y
– матриця D – результат добутку матриць ABC.

На рисунку 5.13 показане комбіноване перетворення:

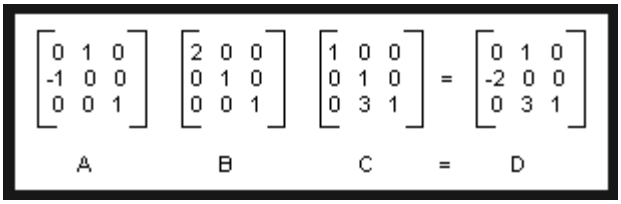


Рисунок 5.12 – Комбіноване перетворення

Матриця комбінованого перетворення може бути сформована шляхом множення окремих матриць перетворення, і це означає, що будь-яка послідовність афінних перетворень може зберігатися в одному об'єкті Matrix.

Порядок множення матриць перетворення має значення та валиває на результат перетворення.

MatrixКлас надає кілька методів для створення складних перетвореннь: Multiply, Rotate, RotateAt Scale Shear і Translate.

Приклад: матриця складного перетворення, яка спочатку повертає на 30 градусів, потім масштабується за коефіцієнтом 3 у напрямку осі y, а потім переміщює на 10 одиниць у напрямку осі x:

```
Matrix myMatrix = new Matrix();
myMatrix.Rotate(45);
myMatrix.Scale(2, 2, MatrixOrder.Append);
myMatrix.Translate(5, 5, MatrixOrder.Append);
```

Конструктори:

Matrix() - ініціалізує новий екземпляр класу Matrix у вигляді одиничної матриці.

Matrix(Matrix3x2) – конструктор об’єкту, Matrix використовує зазначений об'єкт matrix .

Matrix(Rectangle, Point[]),Matrix(RectangleF, PointF[]) - ініціалізує новий екземпляр класу Matrix для геометричного перетворення, що визначається зазначеним прямокутником та масивом точок.

Matrix(Single, Single, Single, Single, Single, Single) - ініціалізує новий екземпляр класу Matrix вказаними елементами.

Деякі властивості:

Elements - повертає масив значень з плаваючою комою, що є елементами цього об'єкта Matrix.

IsInvertible - повертає значення, яке вказує, чи є матриця Matrix оберненою.

MatrixElements - повертає або задаєелементи для матриці.

Деякі методи:

Clone() - створює точну копію об'єкта Matrix.

Dispose() - звільняє всі ресурси, що використовуються цим об'єктом Matrix.

Multiply(Matrix) - перемножує цю матрицю Matrix на матрицю, вказану в параметрі matrix, шляхом додавання перед матрицею

Matrix.Multiply(Matrix, MatrixOrder) - перемножує цей об'єкт Matrix на матрицю, вказану у параметрі matrix, у порядку, що задається у параметрі order.

Reset() - скидає об'єкт Matrix, щоб отримати елементи одиничної матриці.

Rotate(Single) - додає до цього об'єкту Matrix поворот за годинниковою стрілкою навколо початку координат на вказаний кут.

Rotate(Single, MatrixOrder) - застосовує обертання за годинниковою стрілкою навколо початку координат (нульові координати X та Y) на величину, вказану в параметрі angle, до цього об'єкта Matrix.

RotateAt(Single, PointF) - застосовує обертання за годинниковою стрілкою до цього об'єкту Matrix; поворот проводиться навколо точки, вказаної в параметрі point, та шляхом додавання повороту на початок.

RotateAt(Single, PointF, MatrixOrder) - застосовує обертання за годинниковою стрілкою навколо зазначеної точки до об'єкта Matrix у зазначеному порядку.

Scale(Single, Single) - застосовує зазначений вектор масштабування до об'єкта Matrix, додаючи вектор масштабування на початок.

Scale(Single, Single, MatrixOrder) - застосовує зазначений вектор масштабування (scaleX та scaleY) до цього об'єкта Matrix у зазначеному порядку.

Shear(Single, Single) - застосовує зазначений вектор зсуву до цього об'єкту Matrix, додаючи перетворення зсуву на початок.

Shear(Single, Single, MatrixOrder) - застосовує зазначений вектор зсуву до об'єкта Matrix у зазначеному порядку.

ToString() - повертає рядок, що становить поточний об'єкт.

TransformPoints(Point[]) - застосовує геометричне перетворення, яке представляє цей об'єкт Matrix, до зазначеного масиву точок.

TransformPoints(PointF[]) - застосовує геометричне перетворення, яке представляє цей об'єкт Matrix, до зазначеного масиву точок.

TransformVectors(Point[]) - застосовує лише компоненти масштабування та повороту цього об'єкта Matrix до вказаного масиву точок.

TransformVectors(PointF[]) - перемножує кожен вектор масиву на матрицю. Елементи зміщення даної матриці (третій рядок) ігноруються.

Translate(Single, Single) - застосовує зазначений вектор зсуву (offsetX і offsetY) до цього об'єкта Matrix, додаючи вектор зсуву на початок.

Translate(Single, Single, MatrixOrder) - застосовує зазначений вектор зміщення до об'єкту Matrix у зазначеному порядку.

VectorTransformPoints(Point[]) - перемножує кожен вектор масиву на матрицю. Елементи зміщення даної матриці (третій рядок) ігноруються.

MultiplyTransform(Matrix) - перемножує світове перетворення даного об'єкта Graphics на вказаний об'єкт Matrix.

MultiplyTransform(Matrix, MatrixOrder) - перемножує світове перетворення даного об'єкта Graphics на вказаний об'єкт Matrix у заданому порядку.

RotateTransform(Single) - застосовує заданий поворот до матриці перетворення об'єкта Graphics.

RotateTransform(Single, MatrixOrder) - застосовує заданий поворот до матриці перетворення об'єкта Graphics у вказаному порядку.

ScaleTransform(Single, Single) - застосовує зазначену операцію масштабування до матриці перетворення об'єкта Graphics шляхом її додавання перед матрицею перетворення об'єкта.

ScaleTransform(Single, Single, MatrixOrder) - застосовує задану операцію масштабування до матриці перетворення об'єкта Graphics у зазначеному порядку.

ToString() - повертає рядок, що становить поточний об'єкт.

TranslateTransform(Single, Single) - змінює початок системи координат шляхом додавання заданого зсуву до матриці перетворення об'єкта Graphics.

TranslateTransform(Single, Single, MatrixOrder) - змінює початок системи координат шляхом застосування заданого зсуву до матриці перетворення об'єкта Graphics в зазначеному порядку.

Приклад обертання навколо точки:

```
g.TranslateTransform(dx, dy);   // переміщення в необхідну точку
g.RotateTransform(angle);     // обертання на кут
g.TranslateTransform(-dx, -dy); // повернення в початкову точку
```

де angle – кут, на який потрібно повернути, (x, y) – точка, навколо якої здійснюється оберт.

Приклад масштабування фігури, обертання тексту

```
…
 public MyForm()
 {
    InitializeComponent();
    private Point[] points = new Point[] { new Point(50, 160), new Point(180, 110), new Point(250, 210), new Point(50, 160) };
 int leftText = 200;
```

```
        int topText = 200;
        fontFamily = new FontFamily("Arial");
        font = new Font(fontFamily, 25, FontStyle.Bold);
        text = "Maryna";
        }
    private void btnScale_Click(object sender, EventArgs e)
    {
        Graphics g = CreateGraphics();
        g.Clear(Color.White);
        g.FillPolygon(Brushes.Black, points);
        Matrix myMatrix = new Matrix();
        double xScale = Convert.ToDouble(txtXScale.Text);
        double yScale = Convert.ToDouble(txtYScale.Text);
        myMatrix.Scale((float)xScale, (float)yScale, MatrixOrder.Append);
        g.Transform = myMatrix;
        g.FillPolygon(Brushes.Red, points);
    }
    private void btnTextRotate_Click(object sender, EventArgs e)
    {
        Graphics g = CreateGraphics();
        g.Clear(Color.White);
        g.DrawString(text, font, Brushes.Black, leftText, topText);
        int angle = Convert.ToInt32(txtTextRotate.Text);
        GraphicsPath path = new GraphicsPath(FillMode.Winding); // створення
графічного об’єкту
        path.AddString(text, fontFamily, (int)FontStyle.Bold, 50, new Point(leftText +
100, topText + 100), StringFormat.GenericDefault); // додавання тексту
        Matrix matrix = new Matrix();
```

```
            matrix.RotateAt(angle, new PointF(leftText + 100, topText + 100));
//обертання
            path.Transform(matrix); // застосування матриці перетворень до об’єкту
path
            g.FillPath(Brushes.Red, path);
        }
```

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №6 Побудова та переміщення графічних об'єктів в просторі з використанням OpenGL

Мета: ознайомитись загальною матрицею перетворення об’єктів у просторі, яка задає наступні перетворення об’єктів: масштабування, обертання, віддзеркалення, зсув, переміщення, перспективні перетворення та проєкцієвання. Познайомитись з OpenGL (Open Graphics Library) [6], навчитись будувати об’єкти та керувати формою та положенням об’єкта, використовуючи бібліотеки OpenGL.

Завдання

Побудувати модель будь-якого Платонового тіла, здійснити її переміщення, обертання, масштабування, віддзеркалення на екрані з використанням OpenGL.

Теоретичні відомості

Здатність візуалізувати просторовий об'єкт є основою для розуміння форми цього об'єкта.

у багатьох випадках для цього важлива здатність обертати, переносити і будувати види проєкцій об'єкта. це легко демонструється на прикладі знайомства з відносно складним незнайомим об'єктом. Щоб зрозуміти його форму, ми починаємо обертати об'єкт, переміщати на відстань, пересувати вгору і вниз, вперед і назад і т. д.

Щоб зробити те ж саме за допомогою комп'ютера, ми повинні поширити наш попередній двовимірний аналіз на три виміри.

Грунтуючись на отриманому досвіді, ми вводимо однорідні координати. Таким чином, точка в тривимірному просторі [x y z] представляється чотиривимірним вектором [1]:

$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} T \end{bmatrix}$$

, де [T] є матрицею певного перетворення.

Як і раніше, перетворення з однорідних координат в звичайні задається формулою:

$$\begin{bmatrix} x^* & y^* & z^* & 1 \end{bmatrix} = \begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} \frac{x'}{h} & \frac{y'}{h} & \frac{z'}{h} & 1 \end{bmatrix}$$

Узагальнену матрицю перетворення розмірності 4*4 для тривимірних однорідних координат можна представити в наступному вигляді:

$$\begin{bmatrix} T \end{bmatrix} = \begin{bmatrix} a & b & c & p \\ d & e & f & q \\ g & i & j & r \\ l & m & n & s \end{bmatrix}$$

Матрицю перетворення 4*4 можна розділити на чотири окремі частини:

$$\left[\begin{array}{ccc:c} & & & 3 \\ & 3\times 3 & & \times \\ & & & 1 \\ \hdashline \cdots & \cdots & \cdots & \cdots \\ & 1\times 3 & & 1\times 1 \end{array}\right]$$

– **верхня ліва** (3*3) - підматриця задає лінійне перетворення в формі масштабування, зміщення, віддзеркалення і обертання.

– **нижня ліва** (1*3) - підматриця задає переміщення,

– **верхня права** (3*1)- підматриця задає перспективне перетворення.

– **нижня права** (1*1) - підматриця задає загальне масштабування.

Загальне перетворення, отримане після застосування цієї (4*4) - матриці до однорідного вектору і обчислення звичайних координат, називається білінійним перетворенням.

У загальному випадку дане перетворення здійснює комбінацію зсуву, локального масштабування, обертання, віддзеркалення, переміщення, перспективного перетворення і загального масштабування.

Масштабування

Діагональні елементи (4*4) - матриці узагальненого перетворення задають локальне і загальне масштабування.

Наступне перетворення, показує дію локального масштабування:

$$[X][T]=[x \quad y \quad z \quad 1]\begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}=[ax \quad ey \quad jz \quad 1]=[x^* \quad y^* \quad z^* \quad 1]$$

Загальне масштабування можна здійснити, скориставшись четвертим діагональним елементом:

$$[X][T]=[x \quad y \quad z \quad 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}=[x' \quad y' \quad z' \quad s]$$

Звичайні або фізичні координати мають вигляд:

$$[x^* \quad y^* \quad z^* \quad 1]=\left[\frac{x'}{s} \quad \frac{y'}{s} \quad \frac{z'}{s} \quad 1\right]$$

Як і в випадку двовимірного загального масштабування, однорідний координатний множник НЕ дорівнює одиниці. Це означає перетворення з фізичного об’єму ***h=1*** в інший об’єм в 4-вимірному просторі. Перетворені

фізичні координати виходять проєкціюванням через центр 4-вимірної координатної системи назад в фізичний об'єм ***h=1***.

Як і раніше, якщо ***s<1***, відбувається однорідне розширення. якщо ***s>1***, відбувається однорідне стиснення координатного вектора.

Такий результат можна отримати, використовуючи однакові коефіцієнти локальних масштабувань. В цьому випадку матриця перетворення має вигляд

$$[T]=\begin{bmatrix} 1/s & 0 & 0 & 0 \\ 0 & 1/s & 0 & 0 \\ 0 & 0 & 1/s & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Тут однорідний координатний множник дорівнює одиниці, тобто ***h=1***. Таким чином, всі перетворення відбувається у фізичному об'ємі ***h=1***.

Зміщення

Недіагональні елементи у верхній лівій (3*3) - підматриці узагальненої матриці перетворення розміром (4*4) задають зміщення в трьох вимірах, тобто:

$$[X][T]=[x \quad y \quad z \quad 1]\begin{bmatrix} 1 & b & c & 0 \\ d & 1 & d & 0 \\ g & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}=[x+yd+gz \quad bx+y+iz \quad cx+fy+z \quad 1]$$

Обертання

Перш ніж переходити до тривимірного обертання навколо довільної осі, розглянемо обертання навколо кожної з координатних осей.

При обертанні навколо осі ***x*** залишаються незмінними ***x***- координати координатного вектора.

Фактично обертання відбувається в площинах, перпендикулярних осі ***x***.

Аналогічним чином обертання навколо осей ***y*** та ***z*** відбувається в площинах, перпендикулярних осям ***y*** та ***z*** відповідно.

Перетворення координатного вектора в кожній з цих площин задається матрицею двовимірного обертання:

$$[T]=\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

Ця матриця і незмінність координати ***x*** при обертанні навколо осі ***x*** дозволяють записати (4*4) - перетворення однорідних координат при оберті на кут θ у вигляді:

$$[T]=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Аналогічно, матриця перетворення для обертання навколо осі ***z*** на кут ψ має вигляд:

$$[T]=\begin{bmatrix} \cos\psi & \sin\psi & 0 & 0 \\ -\sin\psi & \cos\psi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

При обертанні на кут ϕ навколо осі ***y*** перетворення має вигляд:

$$[T]=\begin{bmatrix} \cos\phi & 0 & -\sin\phi & 0 \\ 0 & 1 & 0 & 0 \\ \sin\phi & 0 & \cos\phi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

В матриці обертання навколо осі ***y,*** знаки у синусів протилежні знакам цих членів в перших двох рівностях. Це потрібно для того, щоб виконувалось погодження про додатній напрямок за правилом правої руки.

Віддзеркалення

При віддзеркаленні відносно площини **xy** змінюються тільки значення **z**-координати координатного вектора об'єкта. насправді, вони змінюють знак. Таким чином, матриця перетворення для відображення відносно площини **xy** дорівнює:

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

При віддзеркаленні відносно площини **yz**:

$$[T] = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

При віддзеркаленні відносно площини **xz:**

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Перенесення

Матриця просторового перенесення має вигляд:

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix}$$

Переміщені однорідні координати отримуємо за допомогою перетворення:

$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l & m & n & 1 \end{bmatrix}$$

Виконавши множення, отримаємо:

$$\begin{bmatrix} x' & y' & z' & h \end{bmatrix} = \begin{bmatrix} (x+l) & (y+m) & (z+n) & 1 \end{bmatrix}$$

З цього випливає, що перетворені фізичні координати:

$$x^* = x + l$$
$$y^* = y + m$$
$$z^* = z + n$$

Композиція перетворень

Послідовні перетворення можуть бути скомбіновані або об'єднані в одне (4*4)- перетворення, що дає той самий результат.

Так як перемноження матриць є некомутативною операцією, то важливий порядок їх виконання (в загальному випадку $[A][B] \neq [B][A]$).

Правильний порядок визначається положенням конкретної матриці перетворення відносно матриці координатного вектора. Матриця, найближча до матриці координатного вектора, встановлює перше перетворення, а остання - Останнє перетворення.

Математично це можна записати:

$$[X][T] = [X][T_1][T_2][T_3][T_4]\ldots$$

де

$$[T] = [T_1][T_2][T_3][T_4]\ldots$$

і $[T_1]$ є довільною комбінацією матриць масштабування, зміщення, обертання, віддзеркалення, перенесення, перспективного перетворення і проєкціювання.

Так як перспективні перетворення спотворюють геометричні об'єкти, а перетворення проєкціювання призводять до втрати інформації, то в разі наявності цих матриць вони повинні бути розташовані відповідно передостанньою і останньою по порядку.

Обертання навколо осі, яка паралельна координатній осі:

– перемістити тіло так, щоб локальна вісь співпала з координатною;

– здійснити обертання навколо зазначеної осі;

– перемістити перетворене тіло у вихідне положення.

Математично це можна записати:

$$[X^*] = [X][Tr][R_x][Tr]^{-1}$$

де

$[X^*]$ - перетворене тіло,

$[X]$ - початкове тіло,

$[Tr]$ - матриця переміщення,

$[R_x]$ - відповідна матриця обертання,

$[Tr]^{-1}$ - матриця, обернена до матриці переміщення.

Для того, щоб зробити кілька обертів в локальній системі осей, паралельних осям глобальної системи координат, потрібно сумістити початки локальної та глобальної систем.

Таким чином, обертання можуть бути виконані за допомогою такої процедури:

– перемістити локальну систему осей таким чином, щоб початки локальної та глобальної систем збіглися;

– виконати необхідні обертання;

– перемістити локальну систему осей назад у вихідне положення.

Обертання навколо довільної осі у просторі

Узагальнений випадок обертання навколо довільної осі в просторі зустрічається часто (в робототехніці, моделюванні).

Обертання навколо довільної осі в просторі виконується за допомогою перенесення і простих обертань навколо координатних осей. метод обертання навколо координатної осі відомий, то основна ідея полягає в тому, щоб сумістити довільну вісь обертання з однією з координатних осей.

Припустимо, що довільна вісь в просторі проходить через точку з (x_0, y_0, z_0) направляючим вектором (c_x, c_y, c_z) .

Обертання навколо довільної осі деякий кут δ виконується за наступним правилом:

– виконати перенесення таким чином, щоб точка (x_0, y_0, z_0) перебувала в початку системи координат;

– виконати відповідні оберти таким чином, щоб вісь обертання збіглася з віссю ***z***;

– виконати оберт на кут δ навколо осі ***z***;

– виконати перетворення, протилежні тим, які дозволили сумістити вісь обертання з віссю ***z***;

– виконати зворотне перенесення.

У загальному випадку для того, щоб довільна вісь, яка проходить через початок координат, збіглася з однією з координатних осей, необхідно зробити два послідовних обертання навколо двох інших координатних осей.

Для суміщення довільної осі обертання з віссю ***z***, спочатку виконаємо обертання навколо осі ***x***, а потім навколо осі ***y***.

Віддзеркалення відносно довільної площини

Часто виникає необхідність віддзеркалити об'єкт відносно довільної площини.

І знову це можна зробити за допомогою процедури, яка об'єднує раніше визначені прості перетворення.

Один з можливих методів полягає в наступному:

– перенести точку ***P***, яка лежить у площині віддзеркалення, в початок системи координат;

– повернути вектор нормалі до площини віддзеркалення в початку координат до суміщення з віссю ***z***. Далі площина віддзеркалення буде збігатися з координатною площиною ***z=0***;

– застосовуючи вже відомі перетворення, віддзеркалити об'єкт відносно координатної площини ***z=0;***

– щоб отримати результати, необхідно виконати перетворення, зворотні до описаних в перших двох пунктах.

Тоді загальне перетворення описується матрицею:

$$[M]=[T][R_x][R_y][Rflt_z][R_y]^{-1}[R_x]^{-1}[T]^{-1}$$

де матриці $[T]$ $[R_x]$ $[R_y]$ задаються відповідними рівняннями, $Rflt_z$ - матриця віддзеркалення відносно площини ***z=0***, $(x_0, y_0, z_0)=(P_x, P_y, P_z)$ - координати точки ***P*** на площині віддзеркалення; а (c_x, c_y, c_z) - вектор нормалі до площини віддзеркалення.

Перспективне перетворення

На відміну від паралельних перетворень, у разі перспективного перетворення, паралельні прямі сходяться, розмір об'єкта зменшується зі збільшенням відстані до центру проєкції, і відбувається неоднорідне спотворення ліній об'єкта, яке залежить від орієнтації та відстані від об'єкта до центру проєкції.

Одноточкове перспективне перетворення задається рівністю:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & rz+1 \end{bmatrix}$$

де $h = rz + 1 \neq 1$ і звичайні координати отримуємо діленням на ***h*** :

$$\begin{bmatrix} x^* & y^* & z^* & 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{rz+1} & \frac{y}{rz+1} & \frac{z}{rz+1} & 1 \end{bmatrix}$$

і має центр проєкції $\begin{bmatrix} -1/p & 0 & 0 & 1 \end{bmatrix}$

а точку сходження , розташовану на осі ***x*** в $\begin{bmatrix} 1/p & 0 & 0 & 1 \end{bmatrix}$

Якщо в четвертому стовпці (4*4)-матриці перетворення два елементи з перших трьох не дорівнюють нулю, то таке перетворення називається двоточковим перспективним перетворенням:

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & (px+qy+1) \end{bmatrix}$$

зі звичайними координатами:

$$\begin{bmatrix} x^* & y^* & z^* & 1 \end{bmatrix} = \begin{bmatrix} \frac{x}{px+qy+1} & \frac{y}{px+qy+1} & \frac{z}{px+qy+1} & 1 \end{bmatrix}$$

І має два центри проєкції:

перший на осі ***x*** в точці $\begin{bmatrix} -1/p & 0 & 0 & 1 \end{bmatrix}$

і другий на осі ***y*** в точці $\begin{bmatrix} 0 & -1/q & 0 & 1 \end{bmatrix}$

і дві точки сходження:

на осі ***x*** в точці $[1/p \quad 0 \quad 0 \quad 1]$

на осі ***y*** в точці $[0 \quad 1/q \quad 0 \quad 1]$.

Триточкову перспективу отримуємо, якщо не дорівнюють нулю три перші елементи четвертого стовпця (4*4)-матриці перетворення:

$$[x \quad y \quad z \quad 1] = \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} = [x \quad y \quad z \quad (px+qy+rz+1)]$$

зі звичайними координатами:

$$[x^* \quad y^* \quad z^* \quad 1] = \left[\frac{x}{px+qy+rz+1} \quad \frac{y}{px+qy+rz+1} \quad \frac{z}{px+qy+rz+1} \quad 1\right]$$

І має три центри проєкції:

перший на осі ***x*** в точці $[-1/p \quad 0 \quad 0 \quad 1]$

другий на осі ***y*** в точці $[0 \quad -1/q \quad 0 \quad 1]$

тратій на осі ***z*** в точці $[0 \quad 0 \quad -1/r \quad 1]$

і дві точки сходження:

на осі ***x*** в точці $[1/p \quad 0 \quad 0 \quad 1]$

на осі ***y*** в точці $[0 \quad 1/q \quad 0 \quad 1]$

на осі z в точці $[0 \quad 0 \quad 1/r \quad 1]$

Перспективне проєкціювання

Перспективне проєкціювання на деяку двовимірну видову площину можна отримати, поєднавши ортографічну проєкцію (промені проєкціювання перпендикулярні до площини проєкції) з перспективним перетворенням.

Ортографічна проєкція – це проєкція на одну з координатних площин **x=0**, **y=0**, **z=0**. Матриця проєкції на площину **x=0** має вигляд:

$$[P_x]=\begin{bmatrix}0&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix}.$$

Перспективне проєкціювання на площину **z=0** виконується за допомогою перетворень:

$$[T]=[P_r][P_z]=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&r\\0&0&0&1\end{bmatrix}\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&0\\0&0&0&1\end{bmatrix}=\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&r\\0&0&0&1\end{bmatrix}$$

$$[x\quad y\quad z\quad 1]\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&0&r\\0&0&0&1\end{bmatrix}=[x\quad y\quad 0\quad rz+1]$$

звичайні координати:

$$[x^*\quad y^*\quad z^*\quad 1]=\left[\frac{x}{rz+1}\quad \frac{y}{rz+1}\quad 0\quad 1\right]$$

Одно-, дво- або триточкове перспективне перетворення можна зробити за допомогою обертань і переносів навколо і вздовж головних осей з наступним одноточковим перспективним перетворенням з центром проєкції, який знаходиться на одній з головних осей.

Ці результати також справедливі для обертання навколо довільної осі у просторі.

Правильними багатогранниками (Платонова тілами) називаються такі опуклі багатогранники, всі грані яких, по суті, правильні багатокутники і всі багатогранні кути при вершинах рівні між собою.

Існує рівно 5 правильних багатогранників (це довів Евклід): правильний тетраедр, гексаедр (куб), октаедр, додекаедр і ікосаедр. Їх основні характеристики приведені в наступній таблиці 6.1.

Таблиця 6.1 – Багатогранники

Багатогранник	*№ граней - Г*	*№ ребер - Р*	*№ вершин - В*
Тетраедр	4	6	4
Гексаедр	6	12	8
Октаедр	8	12	6
Додекаедр	12	30	20
Ікосаедр	20	30	12

В кожному з п'яти випадків числа Г, Р і В пов'язані рівністю Ейлера:

$$Г + В = Р + 2.$$

Правильні багатогранники володіють багатьма цікавими властивостями. Ми розглянемо лише ті властивості, які можна застосувати для побудови цих багатогранників.

Для повного опису правильного багатогранника внаслідок його опуклості досить вказати спосіб пошуку всіх його вершин.

Операції побудови перших трьох Платонових тіл є особливо простими.

Куб (гексаедр) будується зовсім нескладно.

Використовуючи куб, можна побудувати тетраедр і октаедр. Для побудови тетраедра достатньо провести діагоналі протилежних граней куба, які схрещуються. Таким чином, вершинами тетраедра є будь-які 4 вершини куба, попарно не суміжні з жодним з його ребер.

Для побудови октаедра скористаємося наступним властивістю подвійності: вершини октаедра по суті центри граней куба.

Координати вершин октаедра легко обчислюються за координатами вершин куба (кожна координата вершини октаедра є середнім арифметичним однойменних координат чотирьох вершин грані куба, який містить її).

Додекаедр і ікосаедр також можна побудувати за допомогою куба.

Але з використанням OpenGL (Open Graphics Library), побудова тіл значно спрощується [7].

OpenGL розглядається як API (Application Programming Interface - Інтерфейс прикладного програмування), що надає великий набір функцій, які можна використовувати для управління графікою та зображеннями. Тобто OpenGL є специфікацією, розробленою та підтримуваною Khronos Group.

Специфікація OpenGL визначає, яким має бути результат/виведення кожної функції, і як вона має виконуватися. А ось реалізація цієї специфікації залежить від конкретних розробників. Оскільки специфікація OpenGL не надає нам подробиць реалізації, то фактично розроблені версії OpenGL можуть мати різні реалізації доти, поки їх результати відповідають специфікації (і, отже, є однаковими для користувача).

Розробники бібліотек OpenGL, зазвичай є виробниками відеокарт. Кожна відеокарта підтримує певні версії OpenGL, розроблені спеціально під цю лінійку відеокарт.

OpenGL є кросплатформним, незалежним від мови програмування API для роботи з графікою. OpenGL - низькорівневий API.

Основні можливості OpenGL, надані розробникам:

- геометричні примітиви (точки, лінії та багатокутники);
- растрові примітиви (бітові масиви пікселів);
- робота з кольором у RGBA та індексному режимах;
- видові, модельні та текстурні перетворення;

– видалення невидимих ліній та поверхонь (z-буфер);
– робота з прозорістю поверхні багатокутників;
– використання B-сплайнів;
– робота з текстурами;
– застосування освітлення;
– змішування кольорів, усунення ступінчастості (anti-aliasing).

Іменування функцій в OpenGL починаються з приставки gl. Функції, що задають певний параметр, що характеризується набором чисел (наприклад, координати або колір), мають суфікс виду [число параметрів + тип параметрів + представлення параметрів].

Кількість параметрів - вказує кількість параметрів, що приймаються. Приймає значення: 1, 2, 3, 4.

Тип параметрів - вказує тип параметрів. Можливі такі значення: b, s, i, f, d, ub, us, ui. Тобто byte (8-бітне ціле число), short (16-бітне ціле число), int (32-бітне ціле число), float (число з плаваючою комою), double (число з плаваючою комою подвійної точності), unsigned byte, unsigned short, unsigned int (останні три - беззнакові цілі числа).

Представлення параметрів - вказує в якому вигляді передаються параметри, якщо кожне число окремо, то нічого не пишеться, якщо параметри передаються у вигляді масиву, то до назви функції дописується буква v.

Приклад: glVertex2fv визначає координату вершини, що складається з двох чисел з плаваючою комою, що передаються у вигляді покажчика на масив.

Всі графічні об'єкти в OpenGL це набір точок, ліній і багатокутників. Існує 10 різних примітивів, з яких будуються всі об'єкти: двомірні, тривимірні. Всі примітиви, у свою чергу, задаються точками - вершинами.

GL_POINTS - кожна вершина задає точку

GL_LINES - кожна окрема пара вершин задає лінію

GL_LINE_STRIP - кожна пара вершин задає лінію (тобто кінець попередньої лінії є початком наступної)

GL_LINE_LOOP - аналогічно GL_LINE_STRIP за винятком того, що остання вершина з'єднується з першою і виходить замкнута фігура

GL_TRIANGLES - кожна окрема трійка вершин задає трикутник

GL_TRIANGLE_STRIP - кожна наступна вершина задає трикутник разом із двома попередніми

GL_TRIANGLE_FAN - кожен трикутник задається першою вершиною та наступними парами (тобто трикутники будуються навколо першої вершини)

GL_QUADS - кожні чотири вершини утворюють чотирикутник

GL_QUAD_STRIP - кожна наступна пара вершин утворює чотирикутник разом із парою попередніх

GL_POLYGON - задає багатокутник з кількістю кутів, що дорівнює кількості заданих вершин.

Для того, щоб задати примітив, використовується конструкція glBegin (тип_примітиву) ... glEnd (). Вершини задаються glVertex*. Вершини задаються проти годинникової стрілки. Координати задаються від верхнього лівого кута вікна. Колір вершини визначається командою glColor*. Колір задається як RGB чи RGBA. Команда glColor* діє на всі вершини, що йдуть після, доки не зустрінеться інша команда glColor* або на всі вершини, якщо інших команд glColor* немає.

Ось код, що будує трикутник з вершинами синього кольору:

```
1. glBegin(GL_ TRIANGLES);
2. glColor3f(0.0, 0.0, 1.0);
3. glVertex2i(50,  0);
4. glVertex2i(0, 50);
5. glVertex2i(100, 50);
6. glEnd();
```

Існують реалізації OpenGL під різні мови програмування. Дані реалізації можуть відрізнятись від стандартного API, тобто можуть бути внесені певні зміни в бібліотеки. .NET та мова C# не є виключенням і під неї існує множина реалізацій OpenGL: SharpGL, Tao Framework, OpenTK та інші.

Більш детально розглянемо бібліотеку SharpGL, підтримувані версії OpenGL в даній бібліотеці з 1.x по 4.x версію.

Для використання SharpGL в Visual Studio потрібно скачати та підключити в ваш проєкт бібліотеки SharpGL (рисунок 6.1).

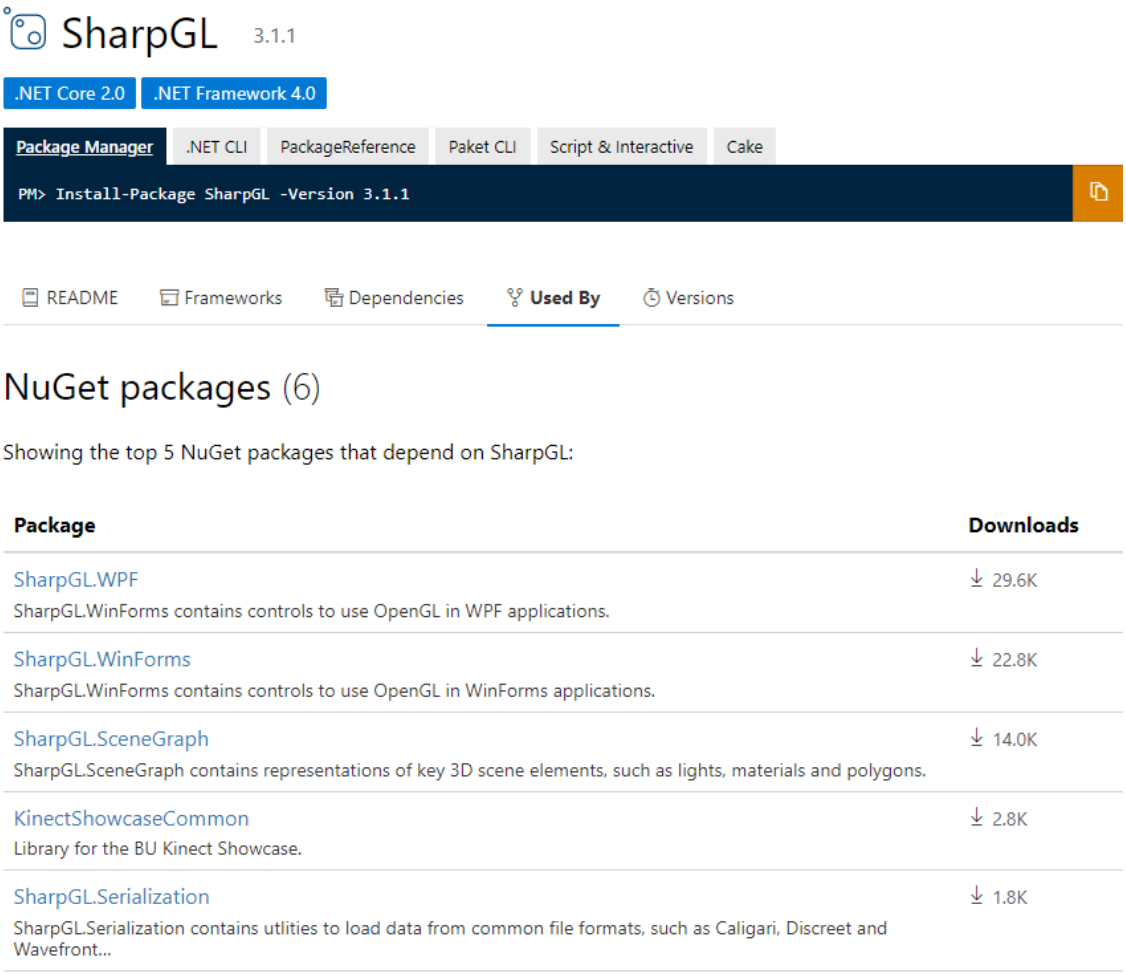


Рисунок 6.1 – Пошук бібліотеки SharpGL

Для підключення, в VS потрібно в пункті меню *Projects* перейти до пункту *NuGet Package Manager -> Package Manager Console ->* вибрати в якості *Package Source "nuget.org" ->* ввести *Install-Package SharpGL ->* ввести *Install-Package SharpGL.WinForms (для WinForms застосунків), Install-Package SharpGL.WPF (для WPF застосунків)* (рисунок 6.2)

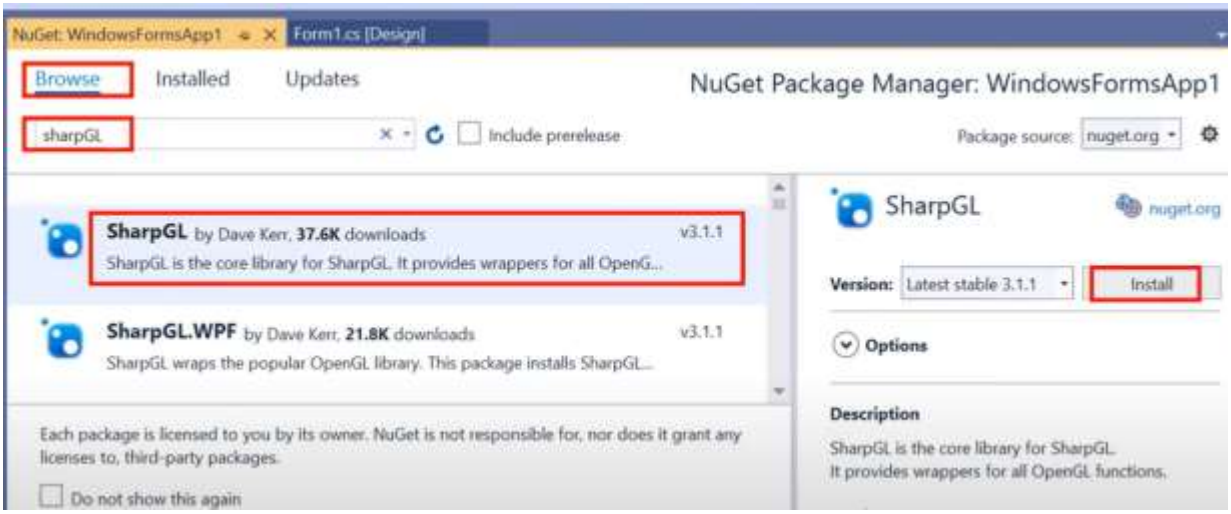


Рисунок 6.2 – Підключення бібліотеки SharpGL

або в *Solution Explorer -> References -> Add reference*: відкриється вікно Управління референсами, потрібно знайти в своїй файловій системі папку *SharpGL.WinForms для WinForms застосунків)*, в якій обрати бібліотеки *SharpGl.dll, SharpGl.SceneGraph.dll, SharpGl.WinForms.dll*.

Для промальовки фігур на формі, потрібно на форму додати елемент OpenGLControl, який ініціалізує OpenGL.

У вкладці Toolbox на будь-якому із елементів викликати контекстне меню та обрати пункт Choose elements… У новому обрати вкладку «Components .NET Framework». Потрібно знайти та додати файл SharpGL.WinForms.dll. Після додавання мають з’явитись елементи: Pointer, GLColorPicker, OpenGLControl, SceneControl, VertexControl.

На форму потрібно перетягнути елемент OpenGLControl – в ньому і буде завантажеватись OpenGL застосунок. Зходимо у властивості нового об'єкта, знаходимо в категорії SharpGL такі атрибути:

Dock – позиція нового контролу на формі. Обираємо Fill, щоб наш контрол зайняв всю площу.

FrameRate – кількість кадрів, які будуть відображені за секунду. За замовчуванням там 20, можна поставити більше. Однак цей параметр залежить від того, наскільки «важка» програма.

OpenGLVersion – версія технології, яка реалізується.

RenderTrigger – один із контекстів рендерингу, можна вибрати TimerBased – тоді кожен кадр буде малюватись на основі часу, тобто з певною частотою в секундах.

Заходимо у події Events нового об'єкта, клікаємо по назаповненому ролю атрибуту OpenGLDraw, переходимо до коду форми, з'явилася функція – обробник подій, яка й малює «контрол»:

```
private void openGLControl1_OpenGLDraw(object sender, RenderEventArgs args)
{
}
```

Вона викликатиметься щоразу при малюванні нашого кадру – як у циклі.

Для того, щоб не використовувати повні імена типів, спочатку пропишемо вгорі простору імен:

```
using ShaprGL;
```

Там знаходяться всі ті функції, які нам потрібні для роботи. В коді бачимо Ініціалізатор форми **InitializeComponent();** – її конструктор, створений автоматично.

В обробнику подій **openGLControl1_OpenGLDraw** потрібно прописати наступні команди:

OpenGL gl = this.openGLControl1.OpenGL; // ми реалізуємо ООП, то спочатку створюємо екземпляр вікна, в якому малюватимемо, щоб було зручно звертатися до нашого «контролу».

gl.Clear(OpenGL.GL_COLOR_BUFFER_BIT | OpenGL.GL_DEPTH_BUFFER_BIT); // викликаємо функцію очищення буфера і очищаємо і колірний буфер – буфер, в якому зберігаються кольори об'єктів, і буфер глибини за допомогою операції «АБО» (|). Без цієї операції зображення та цифри просто накладатимуться одне на одного, і будемо бачити зображення з помилкою *System.NullReferenceException*.

gl.LoadIdentity(); // повертаємо центр координат у початкову точку та скидаємо модельно-видову матрицю, яка реалізовує перетворення (переміщення, обертання та подібні операції над об'єктами). Всі об’єкти, які промальовуються в OpenGL, знаходяться в видовій матриці (матриця з однорідною системою координат (рисунок 6.2), систему координат можна переміщувати та обертати.

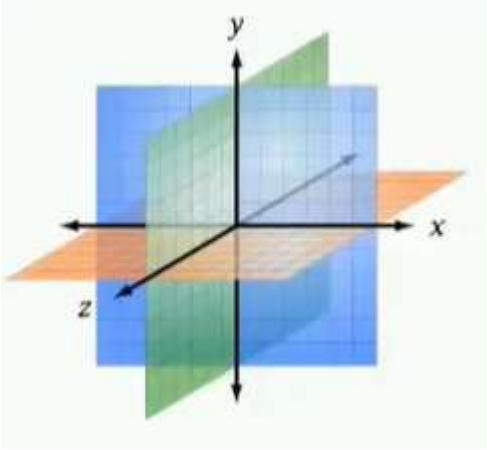


Рисунок 6.3 – Однорідна система координат

gl.Translate(-1.0f, 0.0f, -6.0f); // зсув пера, яким малюватимемо об'єкти вліво та вглиб екрану (gl.Translate(X.Xf, Y.Yf, Z.Zf))

gl.Rotate(angle, 0.0f, 1.0f, 0.0f); // оберт на кут angle навколо осі Y (gl.Rotate(angle, X.Xf, Y.Yf, Z.Zf)). Змінна **angle** ініціалізується до використання (**float angle = 0;**)

І потім малюємо фігуру.

gl.Begin(OpenGL.GL_TRIANGLES);

gl.Color(1.0f, 0.0f, 0.0f); // колір точки (червоний)

gl.Vertex(0.0f, 1.0f, 0.0f); //координати вершини

gl.Vertex(-1.0f, -1.0f, 0.0f);

gl.Vertex(1.0f, -1.0f, 1.0f);

gl.End();

gl.Flush(); // викликає негайне малювання раніше переданих команд

angle += 3.0f // збільшуємо кут обертання (кут змінюється з кожним кадром).

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №7 Створення колажа в Gimp. Пакетна обрабка даних в Gimp. Анімація в Gimp

Мета: познайомитись з графічними редакторами, навчись працювати в них, створювати колажі, анімації, виконувати пакетну обробку зображень.

Завдання

Створити колаж в Gimp. Обробити набір зображень за допомогою механізму пакетної обробки в Gimp. Створити анімацію в Gimp.

Теоретичні відомості

Растрова графіка це - зображення в вигляді масиву точок(пікселей), кожна з який має свій колірний тон і прозорість.

Тіло не має певного кольору. Саме поняття кольору тісно пов'язане з тим, як людина сприймає світло. Можна сказати, що колір зароджується в оці.

Колір - це електромагнітна хвиля. У кожного кольору свій діапазон (довжину кольорової хвилі вимірюють в наноментрах). Найдовші хвилі, що сприймає око людини - червоні, найкоротші - фіолетові. Довші за червоні – інфрачервоні та коротші за фіолетові - ультрафіолетові хвилі, людське око не сприймає.

Колір предмету залежить від самого предмета, від джерела світла, що освічує предмет і від системи людського бачення. Також, одні предмети відбивають світло (дошка, папір), інші його пропускають (скло, вода). Якщо поверхня, яка відбиває тільки синє світло, освітлюється червоним світлом, вона здаватиметься чорною. Аналогічно, якщо джерело зеленого світла розглядати через скло, що пропускає тільки червоне світло, воно здаватиметься чорним.

Найпростішим є ахроматичний колір (не розкладає променів світла на складові кольори), колір, який ми бачимо на екрані чорно-білого приладу. При цьому білими виглядають об'єкти, які ахроматично відбивають понад 80% світла білого джерела, а чорними - менше 3%. Єдиним атрибутом такого кольору є інтенсивність (кількість).

Хроматичний колір - це кольори спектру. Спектр - це діапазон довжин хвиль, які випромінюються джерелом світла. В описі такого кольору зазвичай використовують три величини: колірний тон, насиченість та яскравість.

Колірний тон дозволяє розрізняти кольори, такі як червоний, зелений, жовтий тощо.

Насиченість характеризує чистоту, тобто, ступінь ослаблення (розведення, освітлення) даного кольору білим світлом, і дозволяє відрізняти рожевий колір від червоного, смарагдовий від яскраво-зеленого тощо.

Яскравість відображає представлення про інтенсивність. Наприклад, при зменшенні яскравості синій колір поступово наближається до чорного.

Колір в комп'ютерній графіці потрібен для того, що:

– він несе у собі певну інформацію про об'єкти;

– колір необхідний, щоб розрізняти об'єкти;

– за його допомогою можна вивести одні частини зображення на перший план, інші ж відвести у фон;

– без збільшення розміру за допомогою кольору можна виділити деякі деталі зображення.

Будь-яке комп'ютерне зображення характеризується максимальною кількістю кольорів, які можуть бути використані в ньому - це називається глибиною кольору.

Окрім повноколірних, існують типи зображень з різною глибиною кольору: чорно-білі штрихові, у відтінках сірого, з індексованим кольором. Деякі

типи зображень мають однакову глибину кольору, але різняться за колірною моделлю.

Колірні моделі (математичні моделі) - це спосіб опису кольору за допомогою кількісних характеристик. Для опису кольору, який випромінюється та відбивається використовуються різні колірні моделі.

Колірні моделі можуть бути апаратно-залежними (їх поки що більшість, RGB і CMYK серед них) та апаратно-незалежними (Модель Lab).

Основні колірні моделі:

– RGB (червоний, зелений, синій);
– CMY (блакитний, пурпуровий, жовтий);
– CMYK (блакитний, пурпуровий, жовтий, чорний);
– HSB (Колір, насиченість, яскравість);
– HSV (відтінок, насиченість, значення);
– HLS (відтінок, яскравість, насиченість);
– Lab (Яскравість, колір, колір) та інші.

За принципом дії перелічені колірні моделі можна умовно поділити на три класи:

– адитивні (RGB), базуються на додаванні кольорів;
– субтрактивні (CMY, CMYK), основу яких становить операція віднімання кольорів (субтрактивний синтез);
– перцепційні (HSB,HLS,LAB,YCC), що базуються на сприйнятті.

Колірні моделі представляють засоби для концептуального та кількісного опису кольору. Колірний режим це спосіб реалізації певної колірної моделі у межах конкретної графічної програми.

Колірна модель RGB це одна з найпоширеніших моделей, яка часто використовуються. Вона застосовується в приладах, що випромінюють світло, наприклад монітори, проєктори, фільтри та інші подібні пристрої.

Ця колірна модель базується на трьох основних кольорах: Red, Green і Blue. Кожна з перерахованих вище складових може змінюватись в межах від 0 до 255, утворюючи різні кольори. Повна кількість 256 * 256 * 256 = 16777 216.

При збільшенні яскравості окремих складових збільшуватиметься і яскравість результуючого кольору: якщо змішати всі три кольори з максимальною інтенсивністю, то результатом буде білий колір; навпаки, за відсутності всіх кольорів виходить чорний (рисунок 7.1).

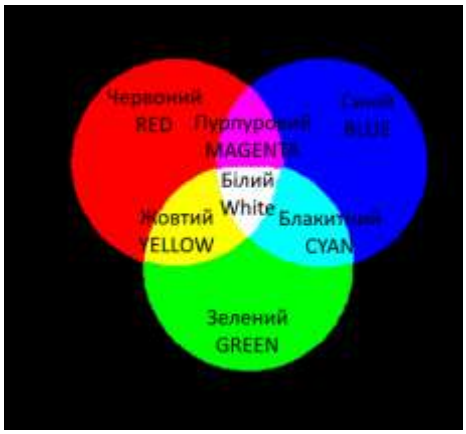


Рисунок 7.1 – Модель RGB

Модель CMY. У цій моделі основні кольори утворюються шляхом віднімання з білого кольору основних адитивних кольорів моделі RGB (рисунок 7.2).

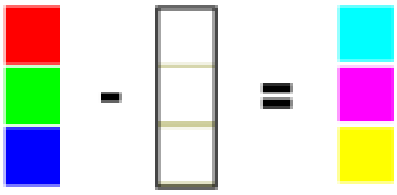


Рисунок 7.2 – Субтрактивна модель CMY

При змішуванні двох субтрактивних кольорів результат затемнюється (у моделі RGB було навпаки) (рисунок 7.3). При нульовому значенні всіх

компонентів утворюється білий колір. Ця модель представляє світло, яке відбивається, Ця є апаратно-залежною, як і модель RGB.

Рисунок 7.3 – Модель CMY

Колірна модель HSB - літери визначають Hue тон (колір), Saturation насиченість і Brightness яскравість (рисунок 7.4).

Всі кольори розташовуються по колу, кожному відповідає свій градус, тобто всього налічується 360.

H визначає частоту світла і набуває значення від 0 до 360 градусів (червоний – 0, жовтий – 60, зелений – 120 градусів тощо). Тобто, будь-який колір у ній визначається своїм кольором (тоном), насиченістю та яскравістю.

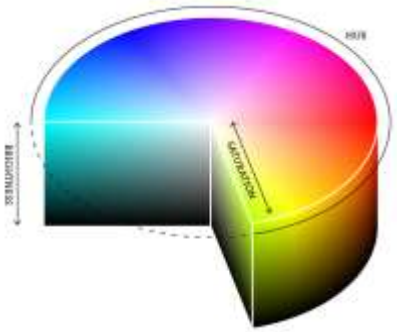


Рисунок 7.4 – Модель HSB

Колірна модель Lab апаратно-незалежна модель і визначає кольори, незважаючи на особливості пристрою (сканера, монітора, принтера). Назву вона

отримала від своїх базових компонентів L, a і b. Компонента L несе інформацію про яскравість зображення, а компоненти a і b - про його кольори. Компонента a змінюється від зеленого до червоного, компонента b від синього до жовтого. Яскравість у цій моделі відокремлена від кольору, що зручно для регулювання контрастності, чіткості тощо.

Растрові зображення використовуються для обробки фотозображень, художньо графіки, реставраційних робіт. При масштабуванні растрових зображень виникають характерні спотворення - "сходинки". Якщо пікселі досить малі, то межі між ними непомітні і око сприймає все, як одне ціле зображення.

Інструментальні засоби растрових редакторів

До фундаментальних інструментів растрової графіки відносяться такі інструменти обробки зображень, як:

- інструменти виділення;
- канали і маски;
- інструменти ретушування;
- гістограми;
- криві;
- інструменти для колірної (колірний баланс) і тонової корекції (рівні);
- фільтри (спецефекти);
- шари.

Крім перерахованих інструментальних засобів до складу растрових редакторів входить велика кількість інструментів, назви яких асоціюються із застосуванням: Пензел, Олівець, Текст, Перо, Лінія, Заливка, Піпетка, Трансформація, Масштаб, Рука, Рамка і т.д. Аналоги цих інструментів також є в більшості векторних редакторів.

Інструменти виділення. Маски

Растрове зображення на відміну від векторного не містить об'єктів, які можна легко розділити для індивідуального редагування. Тому для створення, наприклад, колажів (фотомонтаж) з окремих фрагментів декількох зображень, кожен з них попередньо необхідно виділити. Даний процес нагадує процедуру вирізання окремих фрагментів з цілісного зображення і називається процесом виділення зображень.

Виділення (Selection) - це область, обмежена замкнутою рамкою виділення в вигляді рухомої пунктирної лінії (контуру), яка виділяє частину зображення, доступну для копіювання, редагування та виконання різних типів перетворень.

Маски - це один з базових інструментів професійних растрових редакторів.

Концепції *маски* і *виділення* тісно пов'язані, але не дивлячись на це поняття маски ширше. Будь-яка маска включає в себе два типи областей: непрозорі/білі і прозорі/чорні. Перші використовуються для захисту частин зображень або об'єктів від небажаних змін. Вони і виконують функцію *маскування*. Прозорі області можна розглядати, як отвори в масці. Їх використовують для виділення фрагментів зображення або об'єкта, які збираються редагувати. Ці області називаються *виділеною областю*.

Таким чином, маска не є щось протилежним виділенню. Протилежними властивостями володіють частини маски, а саме захищені і вибрані області. Співвідношення між цими частинами не є постійним. В процесі роботи над зображенням воно може змінюватися за рахунок збільшення частки однієї з них і відповідно зменшення частки іншої. Для цієї мети в растрових редакторах є спеціальний набір інструментів виділення.

Маска і поняття альфа-каналу

Кількість колірних каналів визначається кількістю базових кольорів в використовуваній колірній моделі. Зображення у форматі Grayscale має один канал, в колірних моделях RGB і Lab - три канали, а в моделі CMYK - чотири канали. У растрових редакторах колірні канали генеруються автоматично при

створенні або відкритті зображення. У Gimp [8] доступ до них реалізується за допомогою вікна Канали (рисунок 7.5), для відображення якого необхідно відкрити меню Вікно > Діалоги з підтримкою прикріплення > Канали. Подібні засоби, але під іншими назвами використовуються і в інших растрових редакторах.

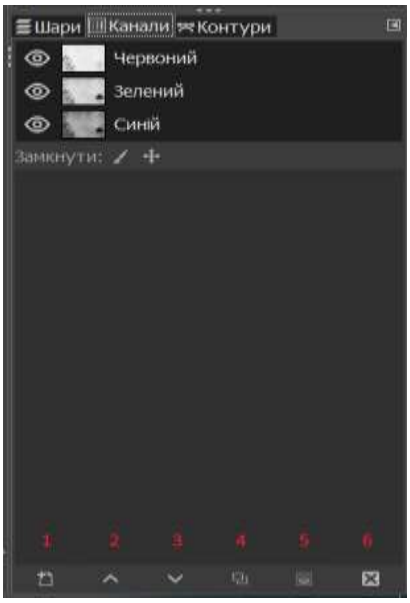


Рисунок 7.5 – Канали

Поряд з колірними каналами, число яких жорстко визначено типом використовуваної колірної моделі, в растрових редакторах можливе використання додаткових каналів (альфа-каналів), кількість яких обмежена тільки можливостями комп'ютера. Цей різновид каналів широко використовується для ретушування, компонування і локальної корекції зображень.

Призначення альфа-каналів тісно пов'язане з поняттям маски. Більш того, фактично кожен такий канал є маскою. Тому створення маски призводить до одночасного створення альфа-каналу, в який поміщається "сіре" зображення маски.

Зовні маска нагадує трафарет. Маска сама є зображенням. Це зображення створюється в моделі Grayscale (Градації сірого) і поміщається поверх іншого зображення, над фрагментами якого потрібно виконати певні операції. Для будь-якого пікселя маски значення відтінку сірого кольору можна змінювати в межах 256 градацій сірого (від 0 до 255). Область маски зі значенням кольору пікселів, рівного 0 (чорний), повністю захищає зображення від змін. Область, пікселі якої мають значення 255 (білий), повністю відкрита для змін.

Як і колірні, альфа-канали зберігаються у вкладці Канали. На рисунку 7.5 відображені три монохромних колірних канали: червоний, зелений і синій, і один альфа-канал, який використовується для зберігання відповідної йому маски. Для роботи з альфа-каналами передбачений ряд інструментів, доступ до яких здійснюється за допомогою набору кнопок, розміщених в нижній частині вкладки Канали.

Таблиця 7. 1 – Канали

Кнопка	**Найменування**	**Призначення**
1	Створення нового каналу	Створення шару-маски
2	Підняти канал на один рівень	Доступно лише для масок виділення: переміщення каналу вгору по списку
3	Опустити канал на один рівень	Переміщення каналу вниз по списку
4	Створити копію каналу та додати його до зображення	Створення копії активного каналу

5	Створити з каналу позначену ділянку	Трансформувати канал в виділення області
6	Вилучити цей канал	Доступно лише для виділення масок: видалення активного каналу

Інструменти і методи ретушування

Застосування інструментів ретушування зображень полягає в ретуші фотографій і відновленні пошкоджених зображень. *Ретуш* - корекція зображення з метою усунення дрібних дефектів, виправлення тонального і колірного балансів.

Найбільш часто використовуваними засобами ретушування є: *інструменти клонування, інструменти розмиття, інструменти Палець і Гумка, інструменти Освітлювач і Випалювання*.

Для ретушування зображень можна використовувати деякі з фільтрів, незважаючи на те, що більшість з них призначений для застосування до зображення спеціальних ефектів. У більшості випадків для отримання потрібного ефекту. їх слід застосовувати в сукупності з масками і виділеннями. При роботі з зображеннями найчастіше доводиться стикатися з дефектами локального характеру, такими, як подряпини, плями і інші подібні спотворення. В даному випадку процес ретушування можна здійснити без застосування виділень і масок, використовуючи лише інструментальні засоби локального поліпшення.

Інструменти клонування (Cloning Tools) призначені для копіювання деталей з одного місця зображення (неушкодженого) в інше (пошкоджене).

Інструменти розмиття (Blur) і підвищення різкості (Sharpen) дозволяють відповідно локально знижувати або підсилювати контраст між пікселями зображення.

Інструменти Палець (Smudge) і Гумка (Sponge) згладжують відмінності між сусідніми відтінками в тих місцях, де проходить кисть.

Інструменти Освітлювач (Dodge) і *Випалювання* (Burn) роблять об'єкти більш світлими або тьмяними.

Гістограма. Оцінка зображення

Інструмент Гістограма (Histogram) дозволяє оцінити різницю між мінімальною і максимальною яскравістю зображення (динамічний діапазон). З його допомогою стає можливим отримати уявлення про розподіл всіх тонів в зображенні. Тому гістограма є одним з основних засобів, що використовується для контролю за тональними і колірними налаштуваннями зображення. Гістограмою називається графік, що відображає розподіл пікселів зображення по яскравості. При побудові цього графіка по осі X відкладаються значення яскравостей в діапазоні від 0 (чорний) до 255 (білий), а по осі Y - кількість пікселів, що мають відповідне значення яскравості. Аналіз гістограми дозволяє зрозуміти, які тонові області зображення потребують корекції. Також за допомогою гістограми можна визначити, які тонові області домінують: тіні (темні області), світла (світлі області) або середні тони.

Терміни *тіні* (shadows), *середні тони* (midtones) і *світлі* (highlights) використовуються в графічних редакторах для позначення відповідно темних, середніх і світлих тонів зображення.

Тонова корекція зображення

Сенс тонової корекції полягає в додаванні зображенню максимального динамічного діапазону. Це безпосередньо пов'язано з налаштуванням яскравості зображення.

Тон - рівень (градація, відтінок) сірого кольору. Тоновое зображення має безперервну шкалу градацій сірого від білого до чорного. Для одного каналу число таких градацій одно 256.

Для оцінки і корекції яскравості і контрастності зображення, тобто для його тонової корекції, в професійних растрових редакторах є широкий набір засобів. До них відносяться потужні універсальні інструменти Рівні (Levels) і Криві (Curves), а також більш прості, наприклад Яскравість / Контраст (Brightness / Contrast), призначені для усунення найбільш грубих дефектів типу недостатній яскравості або підвищеної контрастності.

Колірна корекція і колірний баланс

Колірна корекція - зміна колірних параметрів пікселів (яскравості, контрастності, колірного тону, насиченості) з метою досягнення оптимальних результатів. Найбільш поширеними засобами, які використовуються для підвищення якості кольорових зображень, є такі команди, як Баланс кольорів (Color balance) і Відтінок / Насичення (Hue / Saturation).

Колірний баланс - співвідношення кольорів в зображенні. Регулювання колірного балансу дозволяє посилити або послабити один колір за рахунок іншого додаткового (комплементарного йому).

Фільтри (Plug-ins) і спецефекти (Effects)

В основному фільтри призначені для створення спеціальних ефектів, таких, як імітація мозаїки або будь-якого живописного стилю. За допомогою тривимірних спецефектів двомірні графічні програми здатні перетворити плоске двомірне зображення в об'ємне.

Фільтри і спецефекти є невеликими програми, виконують заздалегідь встановлену послідовність команд. Вони автоматично обчислюють значення і характеристики кожного пікселя зображення і потім модифікують їх відповідно до нових значень.

Робота з шарами

Шар (layer) - додатковий рівень (полотно) для малювання, метафора прозорої кальки. Кожен шар повторює всі параметри основного зображення - розміри, роздільну здатність, колірну модель, число каналів. При збільшенні

кількості шарів зростає розмір зображення. Шари можна міняти місцями, робити невидимими, а також можна малювати тільки на одному шарі, не зачіпаючи інші.

Природно, що якщо зафарбувати будь-який шар суцільним малюнком або суцільним кольором, то на нижніх шарах не буде видно того, що там намальовано. Однак комп'ютерий живопис дозволяє зробити шар напівпрозорим, що відкриває нові можливості в редагуванні зображень.

Опис графічного редактора Gimp

Безкоштовний редактор зображень з відкритим кодом Gimp [8] представляє собою програму для редагування зображень і фото, створення і обробки ілюстрацій. Скачати можна на офіційному сайті https://www.gimp.org/.

Редактор Gimp використовує наступні колірні режими: RGB, Grayscale, Індексовані зображення, CMYK, HSV/HSB.

Головне вікно Gimp (рисунок 7.6):

1 – меню;

2 – відкриті зображення;

3 – панель інструментів;

4 – вибір кольорів;

5 – параметри інструмента;

6 – робоче поле;

7 – діалогові вікна;

8 – вкладки;

9 – рядок стану.

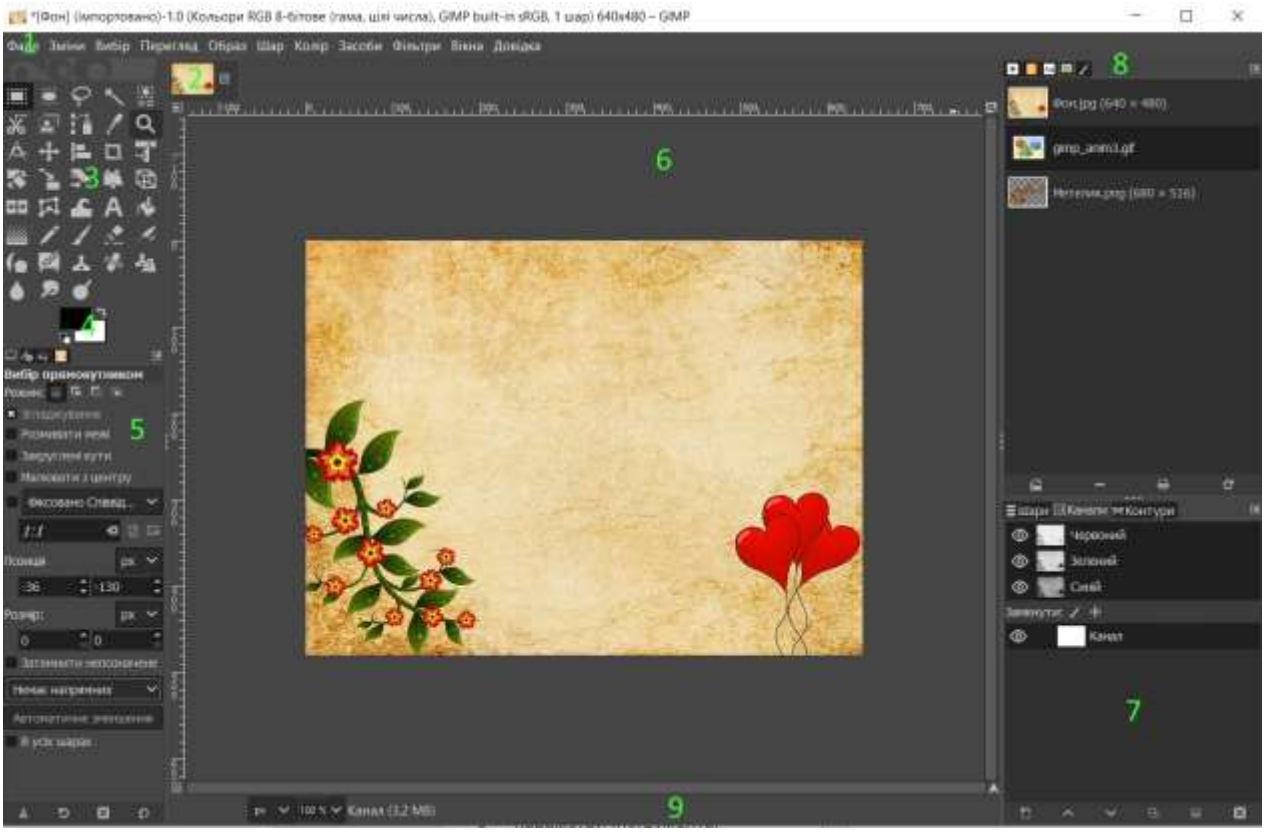


Рисунок 7.6 – Головне вікно Gimp

Панель інструментів (tools bar) є основною робочою панеллю Gimp (рисунок 7.7).

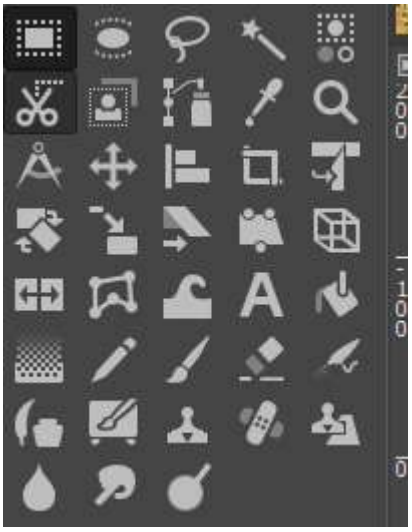


Рисунок 7.7 – Інструменти Gimp

Панель параметрів **(**рисунок 7.8). Дана панель змінює вигляд залежно від обраного інструменту.

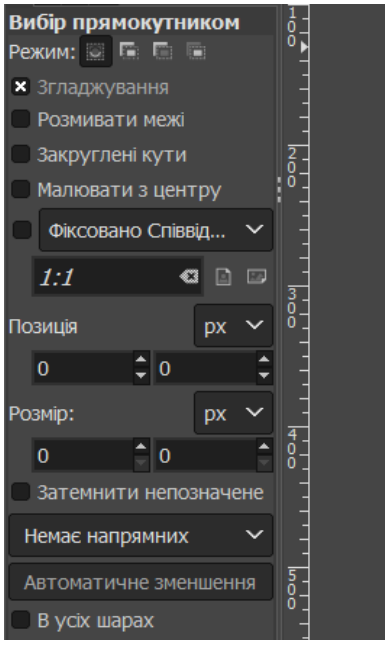


Рисунок 7.8 – Панель параметрів

В таблиці 7.2 описані основні інструменти графічного редактора Gimp.

Таблиця 7.2 – Інструменти редактора Gimp

Інструменти виділення	
	Виділення прямокутником: створення прямокутних виділень
	Виділення еліпсом: створення овальних виділень
	Вільне виділення (Lasso): створення виділень довільної форми
	Вибір пов’язаної ділянки: виділення ділянки за кольором
	Виділення за кольором (Select by Color): виділення ділянки із заповненням схожим кольором
	Розумні ножиці (Intelligent Scissors): виділення фігур за допомогою розпізнавання країв

	Виділення переднього плану (Foreground Select): виділення об'єкта на передньому плані
Інструменти малювання, ретуші	
	Піпетка (Eyedropper): отримання кольору із зображення
	Заливка (Bucket Fill): заповнення ділянки кольором або візерунком
	Градієнт (Gradient): створення плавних переходів між кольорами
	Олівець (Pencil): малювання різкими штрихами
	Пензель: малювання плавних штрихів
	Гумка (Eraser): стирання пензля, стирання до прозорості
	Аерограф (Airbrush): малювання пензлем зі змінним натиском
	Перо: каліграфічне малювання
	Пензель MyPaint: використання в Gimp
	Штамп: вибіркове копіювання із зображення або текстури за допомогою пензля
	Лікувальний пензель: виправлення дефектів у зображенні
	Штамп з перспективою: використання інструменту «Штамп» з урахуванням перспективи зображення
	Різкість/ Розмиття (Sharpen/Blur): зменшення або збільшення різкості пензлем / вибіркове розмиття
	Палець: вибіркове розминня пальцем

	Затемнення/Освітлення (Dodge/Burn): вибіркове затемнення або освітлення
Інструменти трансформації та переміщення	
	Збільшувальне скло (Zoom): збільшення/зменшення зображення
	Вимірювач (Measure): вимірювання відстаней і кутів
	Переміщення (Move): переміщення шарів
	Вирівнювання: вирівнювання чи розстановка шарів, ділянок
	Кадрування: вилучення зображень з межі зображення чи шару
	Об’єднане перетворення: перетворення шарів, позначених, як контур
	Обертання (Rotate): обертання шарів, виділених ділянок, контурів
	Масштаб (Scale): масштабування шару, виділеної ділянки, контуру
	Нахил (Shear): нахиляння шару, виділеної ділянки, контуру
	Перетворення за точками: деформація шару, виділеної ділянки, контуру на основі опорних точок
	Перспектива (Perspective): зміна перспективи шару
	Дзеркало (Reflect): горизонтальне, вертикальне віддзеркалення шару, виділеної ділянки, контуру
	Перетворення виділення: деформування позначеного разом з виділенням
	Викривлення: деформація різними інструментами

Інструменти введення тексту, створення контурів і фігур	
	Текст (Text): додавання та редагування текстового шару
	Контури (Paths): створення та редагування векторних контурів

Колаж

Для побудови колажу нам знадобляться 2 зображення: фон (рисунок 7.9), метелики на фоні (рисунок 7.10).

Рисунок 7.9 – Фон

Рисунок 7.10 – Метелики на фоні

Відкриваємо зображення: Файл > Відкрити

З зображення «Метелики на фоні» виріжемо метелика, для цього використаємо інструмент «Ласо» (рисунок 7.11).

Рисунок 7.11 – Інструмент Ласо

Скопіюємо виділене і вставимо на зображення «Фон», як новий шар (рисунок 7.12): Зміни > Вставити як > Новий шар.

Рисунок 7.12 – Новий шар

Використовуючи інструменти «Масштаб», «Переміщення», «Обертання» - змінимо розмір і положення метелика (рисунок 7.13).

Рисунок 7.13 – «Масштаб», «Переміщення», «Обертання»

Потрібно прибрати фон метелика, для цього створимо «Маску шару». Правою кнопкою миші клікаємо на шарі з метеликом > Додати маску шару. У властивостях маски обираємо «Білий колір – повна непрозорість» (рисунок 7.14).

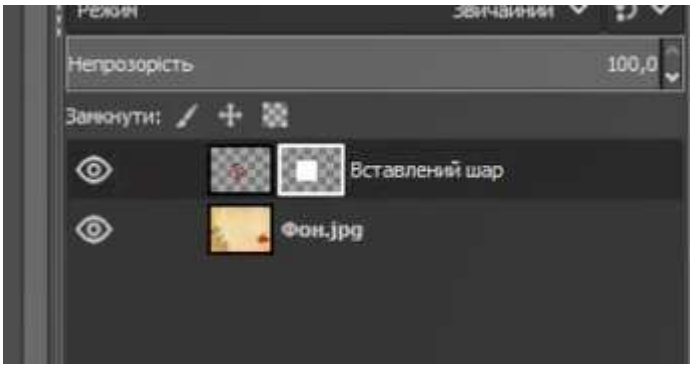


Рисунок 7.14 – Маска шару

Далі виділяємо маску, обираємо інструмент «Пензель» і прибираємо фон метелика (рисунок 7.15).

Додаємо текст, використовуючи інструмент «Тест» і накладемо градієнт на текст. Для цього виділимо шар з текстом, клікаємо на шарі правою кнопкою миші і додаємо «Альфа канал». Всі літери виділяться пунктиром (рисунок 7.16).

Рисунок 7.15 – Інструмент «Пензель»

Рисунок 7.16 – Альфа канал

Створюємо новий шар, тип шару – обраємо прозорий. Далі в тексту виділим контур і вміст: Вибір > Зменшити > Скоротити позначення на, і вказуємо 1 px. Скористаємось інструментом «Градієнт», застосовуємо градієнт до тексту (рисунок 7.17).

Рисунок 7.17 – Градієнт

Далі експортуємо зображення в файл: Файл > Експортувати як - 1.jpg (рисунок 7.18). Якщо ви хочете працювати з проєктом, його потрібно зберегти: Файл > Зберегти як (в цьому випадку будуть доступні всі шари). Результат зображено на рисунку 7.19.

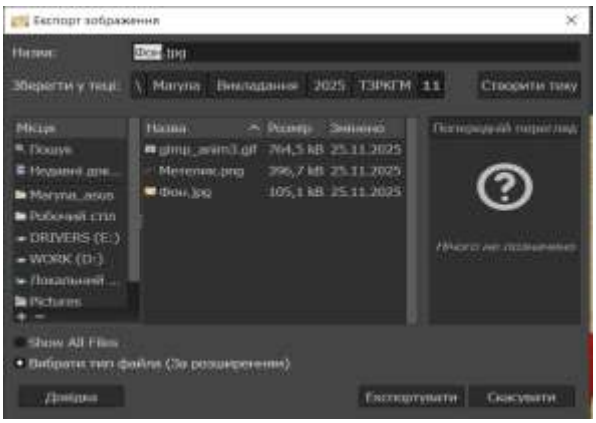


Рисунок 7.18 – Експорт зображення

Рисунок 7.19 – Результат

Пакетна обробка

Засоби автоматизації дій дозволяють значно скоротити час, витрачений на виконання однотипних операцій. Одним з таких засобів є пакетна обробка зображень.

Сенс пакетної обробки полягає додавання різних ефектів і застосування їх до пакету зображень,

Пакетну обробку має сенс застосовувати в тих випадках, коли необхідно, наприклад, змінити розмір зображень, змінити освітленість, зробити однакову корекцію.

В Gimp такої функції нема, але можна скачати Batch Image Manipulation Plugin (BIMP) (для 2-ої версії) і встановити його, після чого він уде вбудований в програму: Файл > Batch Image Manipulation Plugin.

Відкриваємо BIMP (рисунок 7.20):

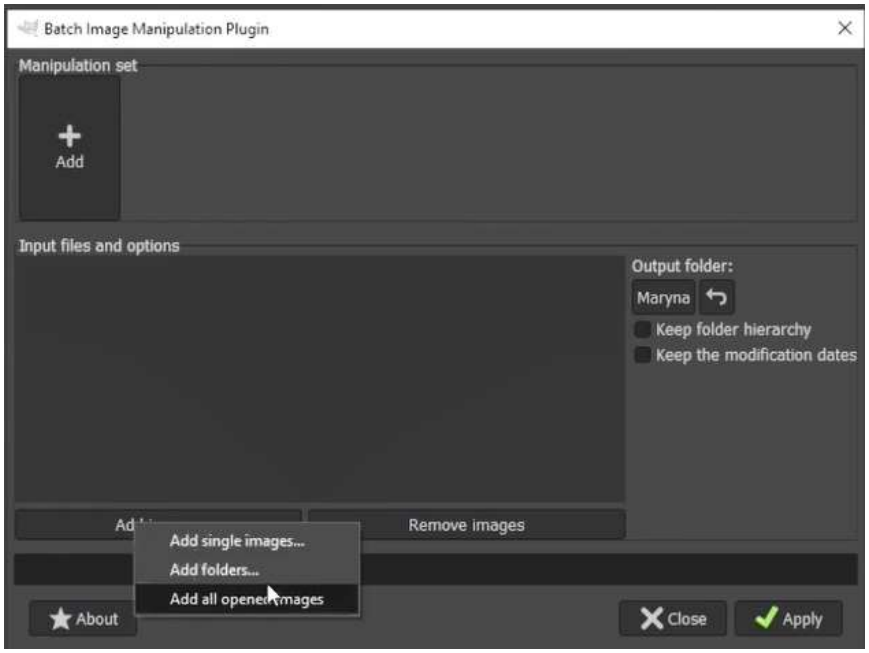


Рисунок 7.20 – BIMP

Обираємо зображення для обробки, каталог, в якій будуть зберігатись оброблені зображення і додаємо ефекти з переліку або мо;на скористатись пошуком інших ефектів (рисунок 7.21) і натискаємо «Застосувати/Apply»

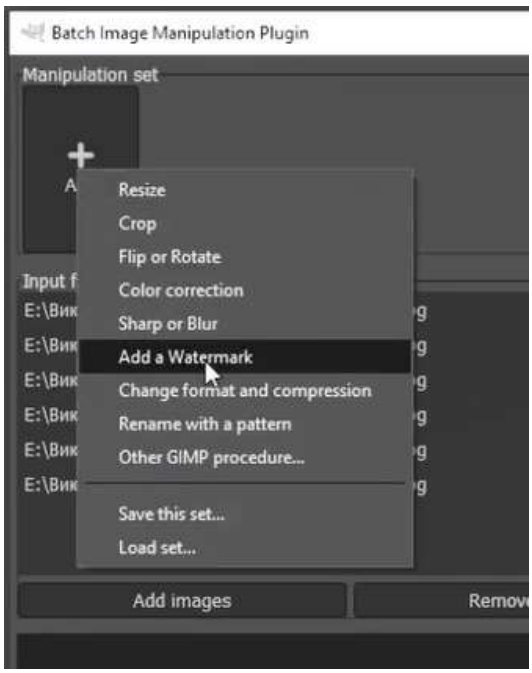


Рисунок 7.21 – Додавання ефектів

Додали водяний знак і розмиття (рисунок 7.22):

Рисунок 7.22 – Результат

Анімація

1-ий спосіб

Розглянемо, як перетворити серію зображень в циклічну анімацію. Щоб створити анімацію нам знадобиться пакет зображень. Даний метод добре працює для створення покадрової анімації, також його можна використовувати для створення GIF анімації з короткого відео.

Відкриємо зображення (рисунок 7.23):

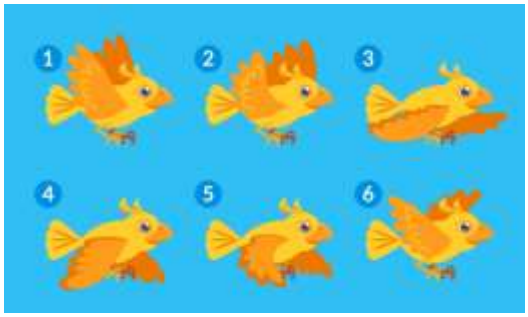


Рисунок 7.23 – Вихідне зображення

Розріжемо це зображення на 6 частин за допомогою інструменту «Напрямні за відсотком»: Образ > Напрямні > Напрямні за відсотком ы збережемо всі зображення окремими файлами.

Відкриємо всі зображення як шари (рисунок 7.24).

Рисунок 7.24 – Всі зображення, як шари

Далі: Фільтри > Анімація > Відтворення і переглянути, що у нас вийшло. Можна зробити оптимізацію для gif: Фільтри > Анімація > Оптимізація (для gif), це зменшить розмір анімації. Далі можна Зберегти проєкт або експортувати в .gif, при цьому в налаштуваннях експорту обов'язково обрати пункт «Анімація». Результат (рисунок 7.25):

Рисунок 7.25 – Результат

2-ий спосіб

Відкриваємо 2-а зображення: Фон.jpg (рисунок 7.26) , Метелики.png (рисунок 7.27) як шари:

Рисунок 7.26 – Фон

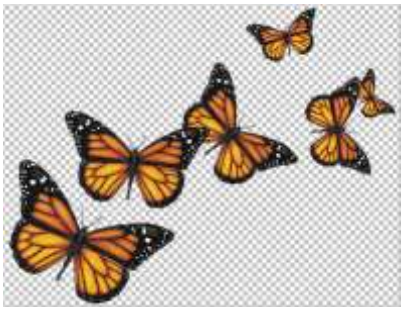


Рисунок 7.27 – Метелики

Використовуючи інструменти «Масштаб», «Переміщення», «Обертання» - змінимо розмір і положення метеликів. Отримуємо перший кадр з двох шарів (рисунок 7.28):

Рисунок 7.28 – Перший кадр

Для наступних кадрів будемо використовувати ці два шари, зробимо кілька копій цих шарів (рисунок 2. 29):

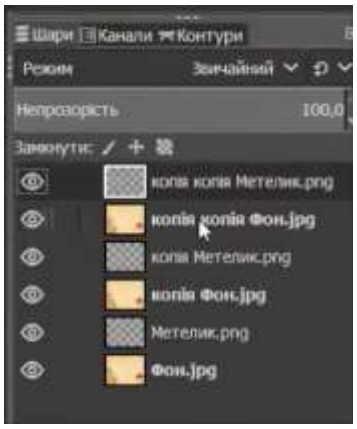


Рисунок 7.29 – Копії шарів для наступних кадрів

Відключаємо шари з фоном і будемо формувати наступні кадри, працюючи з шарами-копіями метелика. Скористаємось інструментами «Переміщення», «Обертання» «Об’єднане перетворення» і змінимо положення метеликів для кожної копії метелика.

Далі: Фільтри > Анімація > Оптимізація (для gif) та експортувати в .gif. Результат (рисунок 7.30):

Рисунок 7.30 – Результат

Відео виконання завдання можна переглянути за посиланням.

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу

ЛАБОРАТОРНИЙ ПРАКТИКУМ №8 Монтаж відео в CapCut

Мета: познайомитись з програмами, які призначені для монтажу відео, навчитись створювати відеокліпа, використовуючи різні види інформації: відео, зображення, звук, текст.

Завдання

Створити відеокліп з відеофайлів та аудіофайлів, зображень, використовуючи CapCut.

Теоретичні відомості

В розділі коротко описано процес створення створити відеокліпа з зображень, відео, аудіо з використанням програми CapCut (рисунок 8.1) [9]. Скачати можна на офіційному сайті https://www.capcut.com/uk-ua/tools/video-editor-download.

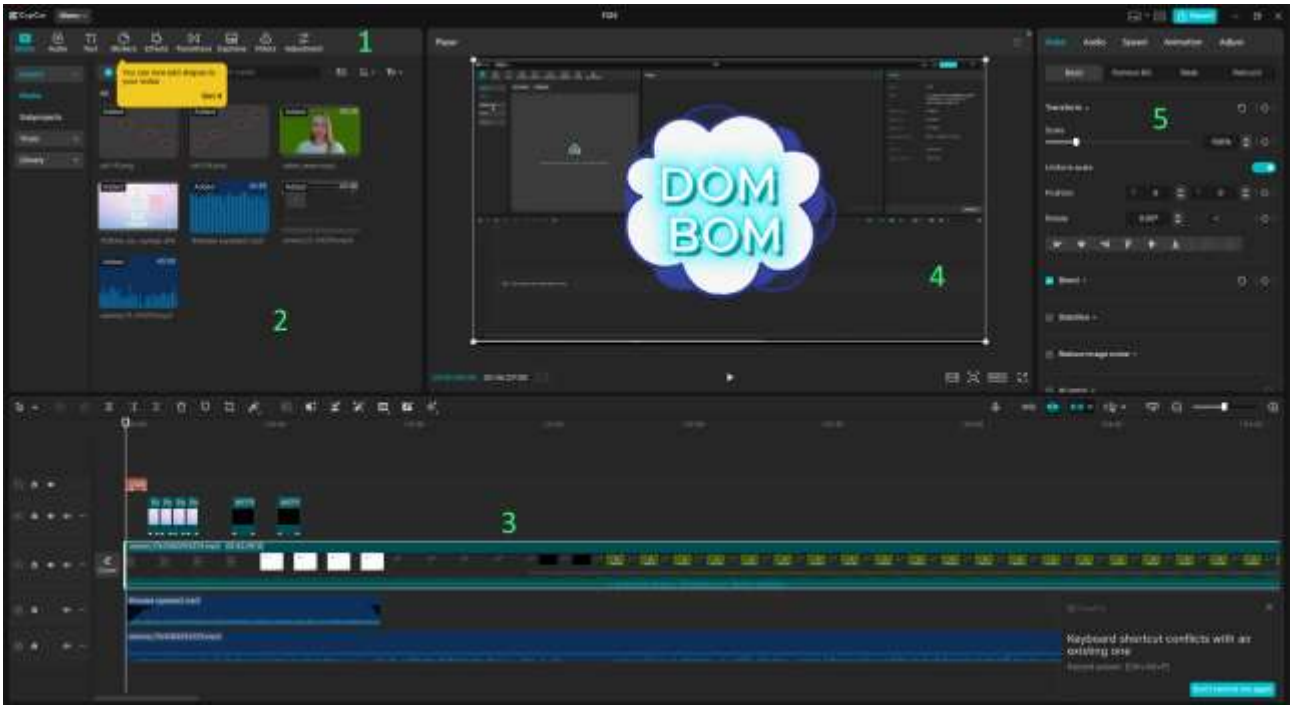


Рисунок 8.1 – Програма CapCut

1 – рядок меню;

2 – вікно з завантаженими для роботи файлами;

3 – панель кадрування (доріжки);

4 – вікно перегляду;

5 – вікно налаштувань.

Для створення проєкту: Media > Import потрібно завантажити файли: зображення, аудіофайл, відеофайл. Після цього перетягнути файли на панель кадрування. Основна доріжка буде відеофайл, фонову музику ставимо останньою доріжкою, зображення додаємо на доріжку, вище основної, якщо доріжку з зображеннями помістити після доріжки з відео, то в результаті, зображень не буде видно в кліпі.

Можна редагувати доріжки (розрізати, вирізати, склеювати), використовуючи панель інструментів (рисунок 8.2):

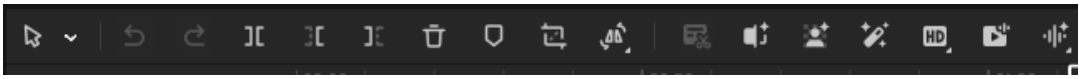


Рисунок 8.2 – Програма CapCut

Можна додавати переходи між частинами відео (рисунок 8.3):

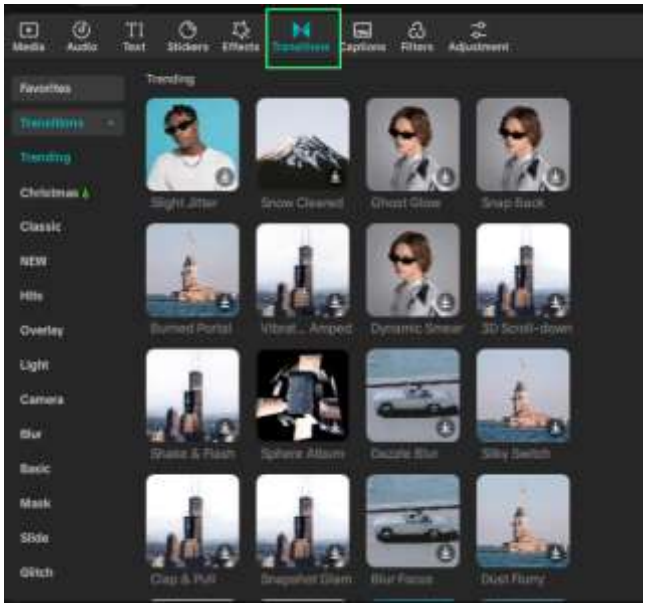


Рисунок 8.3 – Налаштування переходів

Можна додати текст (першою доріжкою). Можна налаштувати час, скільки він буде відображатись, його зовнішній вигляд (рисунок 8.4):

Рисунок 8.4 – Налаштування зовнішнього вигляду тексту

Можна додати ефекти, коли текст з’являється і зникає (рисунок 8.5):

Рисунок 8.5 – Налаштування ефектів для тексту

Доріжка з зображеннями представляється, як відеоряд і з ними можна працювати як з відео. Можна розділити на частини і налаштувати окремо кожну

частику, також додати ефекти, коли зображення з’являється і зникає (рисунок 8.6):

Рисунок 8.6 – Налаштування ефектів для тексту

Аудіодоріжку теж можна налаштувати: рівень звуку, посилення, на початку, затухання в кінці (рисунок 8.7):

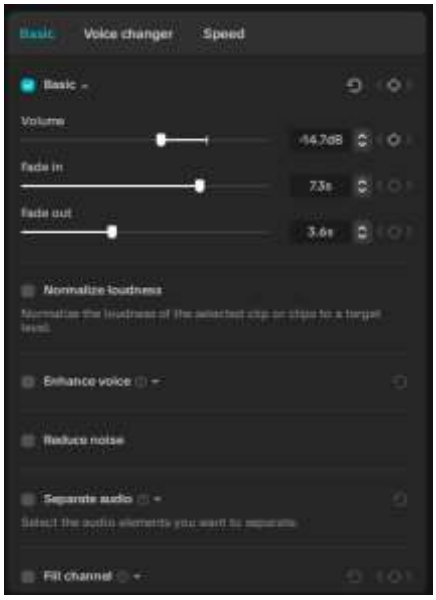


Рисунок 8.7 – Налаштування ефектів для аудіо

Готовий кліп потрібно експортувати Menu > File > Export, даємо ім’я файлу, шлях, куди експортувати (рисунок 8.8):

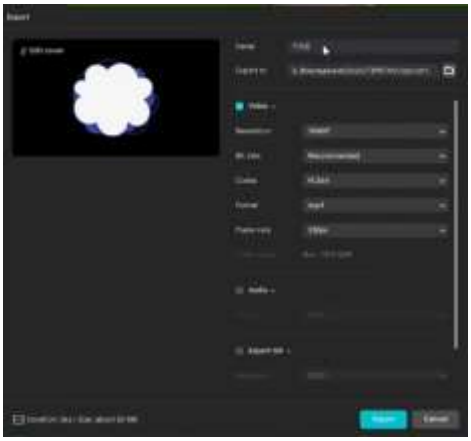


Рисунок 8.8 – Експорт

В програмі нема можливості зберегти проєкт окремим файлом, але CapCut автоматно зберігає проєкт при закритті у власній структурі і він буде доступний при наступному запуску програми (рисунок 8.9):

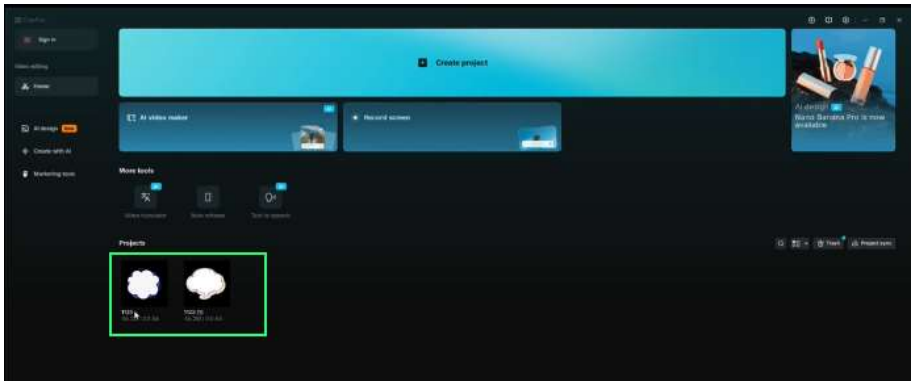


Рисунок 8.9 – Проєкти

Відео виконання завдання можна переглянути за посиланням.

Порядок виконання лабораторної роботи

1) ознайомитися з теоретичними відомостями;
2) виконати завдання;
3) оформити звіт;
4) продемонструвати результат на комп'ютері і захистити роботу

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Пічугін М.Ф., Канкін І.О., Воротніков В.В. Комп’ютерна графіка – Центр учбової літератури, 2019 – 346 с
2. Visua Studio Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/uk-ua/visualstudio/windows/?view=visualstudio
3. Mike McGrath. C# Programming in Easy Steps 3rd Edition – in Easy Steps, 2022 – 192 p
4. Chris Sells. Windows Forms Programming in C# – Addison-Wesley Professional, 2004 - 681 p.
5. GDI Microsoft Learn. [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/api/system.drawing.graphics?view=windowsdesktop-10.0
6. Donald D. Hearn, Pauline Baker . Computer Graphics with Open GL: Pearson New International Edition, 4th Edition – Longman, 2013 – 824 p
7. Open GL Open Tutorial. [Електронний ресурс] – Режим доступу: https://learnopengl.com/
8. GIMP User Manual. [Електронний ресурс] – Режим доступу: https://docs.gimp.org/3.0/en/
9. CapCut. [Електронний ресурс] – Режим доступу: https://www.capcut.com/uk-ua/resource/how-to-use-capcut