ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»

КОМП’ЮТЕРНА ГРАФІКА ТА 3D-МОДЕЛЮВАННЯ:

методичні рекомендації до виконання індивідуальних завдань

Запоріжжя 2025

metinvest polytechnic

УДК 004.92:004.42(072)
К63

Рекомендовано Науково-методичною радою
ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«МЕТІНВЕСТ ПОЛІТЕХНІКА»
(протокол № 8 від 27.06.2025 р.)

Укладач
Міхєєнко Д. Ю., канд. техн. наук, доцент

К63 Комп’ютерна графіка та 3D-моделювання: методичні рекомендації до виконання індивідуальних завдань (для студентів спеціальності 122 Комп’ютерні науки усіх форм навчання першого (бакалаврського) рівня вищої освіти) / уклад. Д. Ю. Міхєєнко. Запоріжжя : ТОВ «ТЕХНІЧНИЙ УНІВЕРСИТЕТ «МЕТІНВЕСТ ПОЛІТЕХНІКА». 2025. 54 с.

Методичні вказівки містять основні теоретичні відомості, приклади, похідні данні для виконання індивідуальних завдань за темою «Створення графічних програм у системі windows використанням OpenTK та OpenGL» та «Моделювання тривимірних об'єктів» при вивченні студентами дисципліни «Комп’ютерна графіка та 3D-моделювання». Матеріал навчального посібника має на меті підвищити якість виконання завдань, ознайомити студентів з основними методами створення графічних програм та моделювання, а також допомогти в освоєнні сучасних технологій тривимірної графіки. Крім того, посібник сприяє формуванню практичних навичок використання отриманих знань у розробці та візуалізації графічних об'єктів.

Рекомендовано для студентів спеціальності 122 Комп’ютерні науки усіх форм навчання першого (бакалаврського) рівня освіти.

УДК 004.92:004.42(072)

ЗМІСТ

ВСТУП

Сучасні технології комп’ютерної графіки та 3D-моделювання є невід’ємною частиною різних галузей науки, техніки, мистецтва та виробництва. Вони широко застосовуються в анімації, геймдеві, архітектурному проєктуванні, промисловому дизайні, інженерних розрахунках і науковій візуалізації. Використання графічних API, таких як OpenGL та OpenTK, дозволяє створювати високоякісні візуалізації, які значно спрощують аналіз і розуміння складних процесів.

Дисципліна «Комп’ютерна графіка та 3D-моделювання» спрямована на вивчення основних методів і алгоритмів, що використовуються для створення графічних об’єктів, а також освоєння інструментів для їх реалізації. У процесі навчання студенти набувають навичок програмування, роботи з графічними бібліотеками та моделювання тривимірних сцен. Опанування цих технологій дозволяє їм застосовувати отримані знання в різних сферах діяльності.

Ці методичні вказівки допоможуть студентам виконати індивідуальні завдання з тривимірного моделювання та програмування графіки в середовищі Windows за допомогою OpenTK і OpenGL. Посібник містить необхідні теоретичні відомості, практичні рекомендації та приклади реалізації графічних програм, що сприятиме ефективному засвоєнню матеріалу та розвитку практичних навичок.

1 СТВОРЕННЯ ГРАФІЧНИХ ПРОГРАМ У СИСТЕМІ WINDOWS ВИКОРИСТАННЯМ OPENTK ТА OPENGL

1.1 Короткі теоретичні відомості

OpenGL (Open Graphics Library) — це кросплатформний графічний API, який забезпечує програмістів засобами для створення двовимірної та тривимірної графіки. Він використовується у багатьох сферах, зокрема в комп'ютерних іграх, науковій візуалізації, інженерному моделюванні та анімації. OpenGL є низькорівневим API, що дозволяє безпосередньо взаємодіяти з графічним процесором (GPU), надаючи програмістам високий рівень контролю над рендерингом сцени.

OpenGL, розроблений в 90-х роках XX століття, є потужним і популярним стандартом для програмування комп'ютерної графіки. Його різні версії, від 1.0 до 4.x, постійно вдосконалюються, але для початкового вивчення графіки в курсах комп'ютерної графіки зазвичай використовують версію 3.0. Це пов'язано з тим, що версії 4.x і Vulkan потребують глибших знань у 3D-графіці, системному програмуванні, а також навичок роботи з багатопоточністю і асинхронним програмуванням.

Однією з головних переваг OpenGL є зворотна сумісність. Нові версії стандарту забезпечують підтримку старих програм, що дозволяє користувачам оновлювати свої застосунки без необхідності переписувати код. Це гарантує стабільну роботу програм, навіть якщо застосовуються нові функції та можливості стандарту.

OpenGL є кросплатформним стандартом, що забезпечує однаковий графічний результат на різних операційних системах, таких як Windows, Linux і macOS. Це дозволяє розробникам створювати програми, які працюватимуть на різних пристроях та системах, без потреби в додатковій оптимізації для кожної платформи.

OpenGL дозволяє реалізовувати сучасні графічні можливості відеокарт, що забезпечує створення високоякісних, реалістичних зображень. Оскільки стандарт спроектований для роботи з апаратними засобами відеокарт, він дозволяє максимально ефективно використовувати їх можливості для досягнення високої продуктивності в графічних застосунках.

OpenGL має чітко структурований та зручний інтерфейс, що робить його простим для використання навіть для початківців. Це дозволяє створювати графічні застосунки за меншу кількість рядків коду порівняно з іншими графічними бібліотеками, що полегшує розробку та прискорює процес навчання.

OpenGL — це потужний графічний стандарт у галузі комп'ютерної графіки, який надає більше 300 функцій для малювання складних тривимірних зображень, що складаються з простих геометричних примітивів, таких як точки, лінії, полігони та інші базові елементи. Завдяки своїй гнучкості та широким можливостям, OpenGL є незамінним інструментом у ряді сфер, пов'язаних із створенням графіки та візуалізацій.

Завдяки своїм можливостям рендерингу 3D-графіки, OpenGL активно використовується для створення відеоігор, де важливим є реалістичне відображення графічних елементів у реальному часі. OpenGL застосовується у програмних продуктах для проектування, де необхідно здійснювати візуалізацію складних 3D-моделей і графічних конструкцій, що допомагає інженерам та дизайнерам точніше оцінювати проекти. Віртуальна реальність потребує високої швидкості та точності відображення 3D-середовищ. OpenGL використовується для рендерингу складних 3D-сцен у реальному часі, що є критичним для ефективної роботи VR-систем. OpenGL допомагає у створенні візуальних представлень складних наукових, медичних або інженерних даних, що дозволяє користувачам краще аналізувати інформацію через інтуїтивно зрозумілу графіку.

OpenTK (Open Toolkit) — це бібліотека для роботи з OpenGL у середовищі .NET (C#). Вона спрощує використання OpenGL у Windows-додатках, забезпечуючи зручний інтерфейс для взаємодії з графічною підсистемою. OpenTK містить функції для керування вікном програми, обробки подій введення (клавіатура, миша) та реалізації анімації. Завдяки OpenTK програмісти можуть створювати продуктивні графічні додатки, використовуючи можливості OpenGL у поєднанні з мовою програмування C#.

Розробка графічних програм у Windows з використанням OpenGL та OpenTK включає кілька основних етапів: ініціалізація графічного контексту, налаштування параметрів рендерингу, створення геометричних об'єктів, застосування шейдерів і виконання процесу рендерингу. Також важливою частиною є оптимізація продуктивності програми шляхом ефективного управління ресурсами GPU, зокрема буферами вершин і текстур.

Однією з ключових особливостей OpenGL є його розширюваність через розширення (extensions), які дозволяють використовувати нові можливості графічних карт без очікування офіційних оновлень API. Завдяки цьому OpenGL залишається актуальним для різних платформ, включаючи Windows, macOS, Linux та мобільні пристрої.

OpenGL побудований за принципом **графічного конвеєра (Graphics Pipeline)**, який послідовно обробляє графічні дані перед їхнім відображенням на екрані. Конвеєр складається з декількох етапів: обробка вершин, складання примітивів, растеризація, обробка фрагментів і вивід пікселів (рис. 1.1). Така структура дозволяє ефективно використовувати можливості GPU, оптимізуючи рендеринг сцени.

Одним із ключових принципів OpenGL є використання станової машини (State Machine). Це означає, що всі параметри, встановлені під час роботи API, зберігаються у внутрішньому стані OpenGL і впливають на подальші операції. Наприклад, якщо змінити колір перед рендерингом примітиву, цей колір буде застосований до всіх подальших об'єктів, поки не

буде встановлене нове значення.

OpenGL працює через виклики функцій (API calls), які передають команди до GPU. У сучасних версіях OpenGL (починаючи з 3.0) активно використовуються шейдери (Shaders), що дозволяють програмістам керувати обробкою графічних даних на рівні GPU. Шейдери пишуться мовою GLSL (OpenGL Shading Language) і можуть значно розширювати можливості рендерингу, забезпечуючи високий рівень деталізації та складні ефекти.

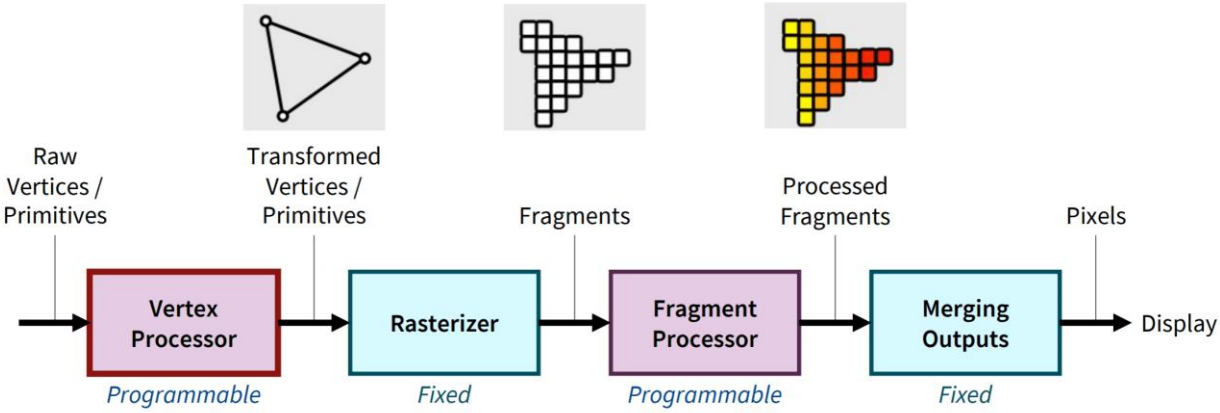


Рисунок 1.1 – Графічний конвеєр OpenGL

OpenGL працює у буфері кадру, а для відображення вмісту буфера кадра у вікно екрану використовуються різні бібліотеки, серед яких найбільш популярні — freeglut, CPW, GLOW, GLUI, GLFW та інші. Серед них GLFW є сучасним і найбільш використовуваним варіантом. Це багатоплатформна бібліотека з відкритим вихідним кодом, яка забезпечує простий API для створення вікон, контекстів, поверхонь, отримання вхідних даних та інших операцій. GLFW підтримує операційні системи Windows, macOS, X11 і Wayland і ліцензується за ліцензією zlib/libpng.

Для розв'язання математичних задач, пов'язаних з графікою, OpenGL часто використовує спеціалізовані бібліотеки, такі як GLM

(OpenGL Mathematics). GLM надає класів і функцій для роботи з векторами, матрицями та кватерніонами, що є основними елементами для роботи з графічними поняттями. Бібліотека містить також допоміжні класи для створення таких важливих 3D-структур, як матриці перспективи та огляду, що значно спрощує процес розробки графічних застосунків.

Ще однією важливою функцією OpenGL є можливість додавання текстур до об'єктів у графічних сценах. Це можна здійснити за допомогою різних бібліотек для роботи з текстурами, таких як FreeImage, DevIL, GLI (OpenGL Image), Glraw і SOIL (Simple OpenGL Image Loader). Вони надають зручні інтерфейси для завантаження, збереження та обробки зображень, які використовуються як текстури в графічних сценах.

Програмування на OpenGL тісно пов'язане з апаратним і програмним забезпеченням. Для роботи з OpenGL необхідна графічна карта (GPU), оскільки стандарт використовує багатоступінчастий графічний конвеєр, який частково програмується за допомогою GLSL (OpenGL Shading Language). API OpenGL написано на C і є сумісним з мовами C і C++, а також доступне для інших мов програмування, таких як Java, Python, Perl, Ruby, Visual Basic, Delphi тощо, що робить OpenGL універсальним інструментом для розробників різних платформ.

1.2 Приклад виконання індивідуального завдання

Open Toolkit Library (OpenTK) — це набір бібліотек для мови C#, який дозволяє легко працювати з OpenGL, OpenCL і OpenAL. OpenTK використовують для створення ігор, наукових програм та інших проєктів, де потрібна тривимірна графіка.

Бібліотека OpenTK підтримує нові версії OpenGL і має зручний спосіб виклику функцій, що допомагає уникнути багатьох помилок. OpenTK можна використовувати як у середовищі Microsoft .NET Framework (починаючи з версії 2.0), так і у Mono Framework — це кросплатформена версія

.NET з відкритим кодом.

Щоб встановити OpenTK, найзручніше скористатися системою керування пакетами NuGet, яка є у середовищі розробки Microsoft Visual Studio.

Спочатку потрібно створити новий проєкт типу Windows Forms Application у Visual Studio (рис. 1.2). Після цього запустіть диспетчер пакетів NuGet. Для цього можна відкрити меню «Інструменти» → «Диспетчер пакетів NuGet» → «Консоль диспетчера пакетів» або клацнути правою кнопкою миші на проєкті у рішенні та вибрати «Керувати пакетами NuGet». У консолі диспетчера пакетів потрібно написати команду: Install-Package OpenTK. Це завантажить і додасть бібліотеку OpenTK до вашого проєкту.

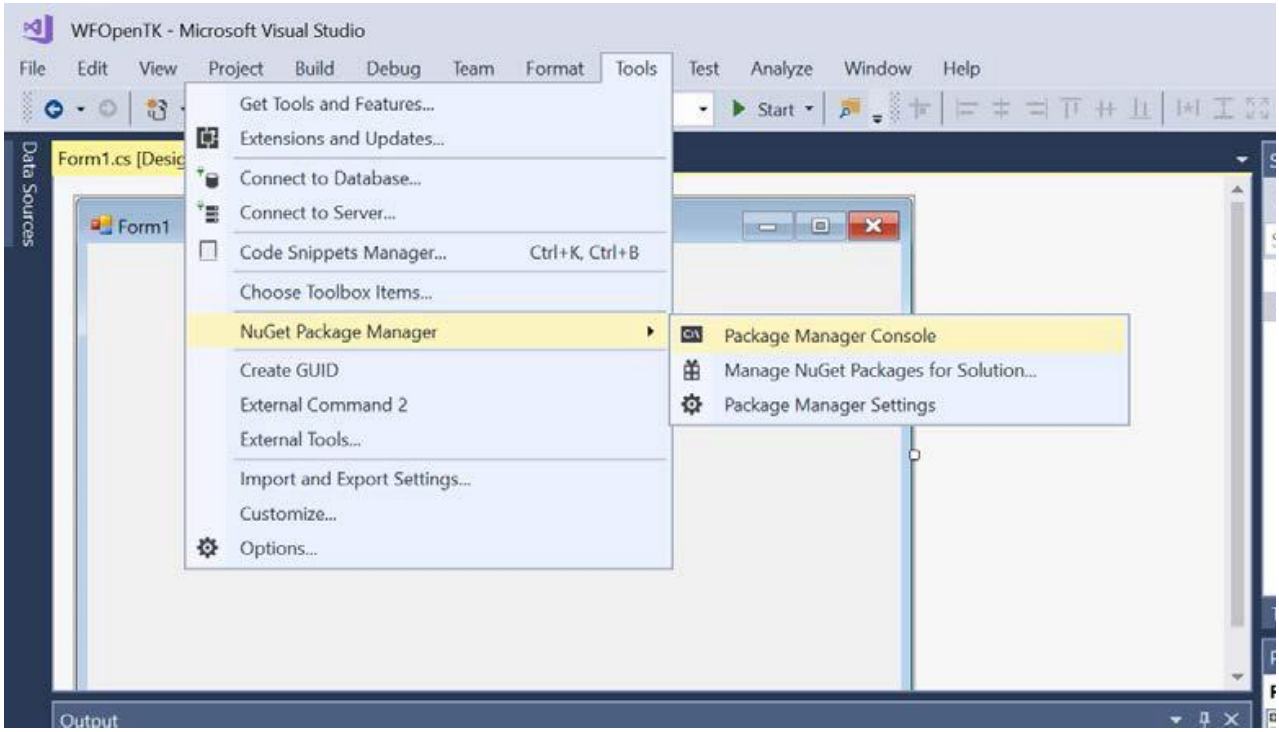


Рисунок 1.2 – Створення проєкту типу Windows Forms Application у Visual Studio

Для коректної роботи з OpenTK у коді необхідно підключити потрібні простори імен. Для цього на початку файлу програми додайте такі рядки:

using OpenTK;

using OpenTK.Graphics;

using OpenTK.Graphics.OpenGL;

У створеному проєкті у вікні Toolbox (панель елементів) відкрийте вкладку OpenTK, знайдіть елемент керування GLControl (рис. 1.3), перетягніть його на форму та за потреби розтягніть до потрібного розміру . Це дозволить створити область для відображення графіки OpenGL у вашій програмі.

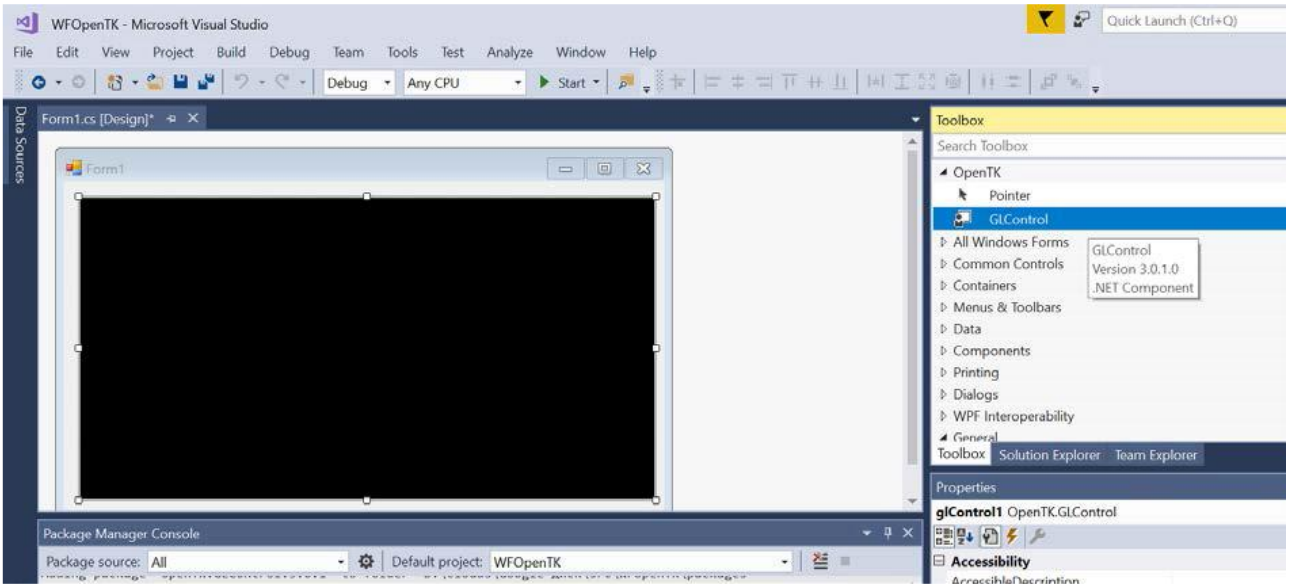


Рисунок 1.3 – Вибір елементу керування GLControl

Тепер елемент GLControl стане доступним для розміщення на формі програми. Перетягніть цей елемент і помістіть його на форму.

Цей елемент відповідає за параметри та властивості області для малювання графіки.

Далі відкрийте властивості елемента (Properties) і змініть параметр Name з glControl1 на AnT — це полегшить роботу з цим контролом у коді (рис. 1.4).

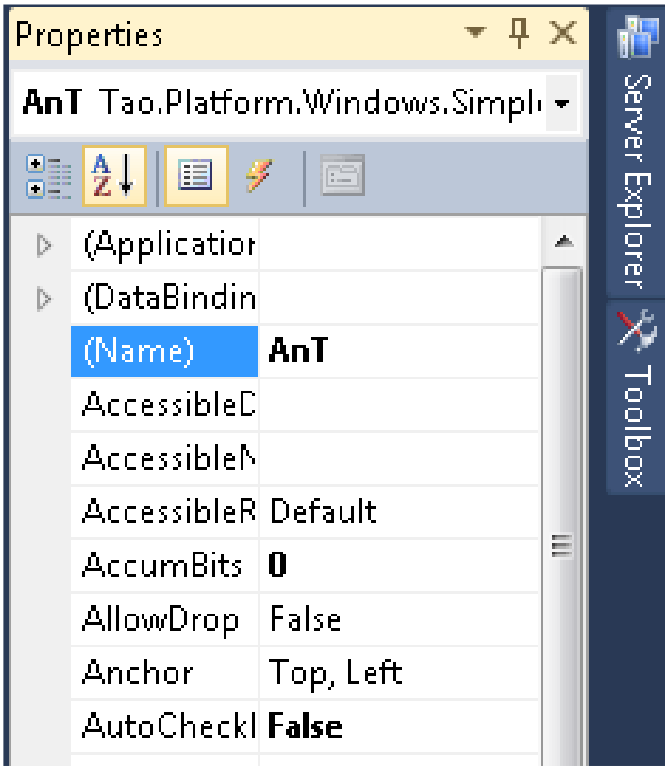


Рисунок 1.4 – Заміна параметру name на “AnT”

Розглянемо приклад створення програми, яка відображає графік заданої функції та анімовано показує зміну її значень.

У цій програмі буде реалізовано наступне: по лінії графіка рухатиметься червона точка. Вона демонструватиме, як змінюються значення функції y у при зміні x х у межах видимої області графіка (рис. 1.5). Це дозволить наочно спостерігати за поведінкою функції у реальному часі.

Для створення програми використовуйте елемент GLControl, який вже розміщений на формі (з ім’ям AnT).

Щоб забезпечити коректну анімацію руху точки по графіку, потрібно додати до форми об'єкт Timer. Він буде відраховувати проміжки часу та запускати оновлення зображення приблизно 30 разів на секунду, що дозволить створити плавний рух.

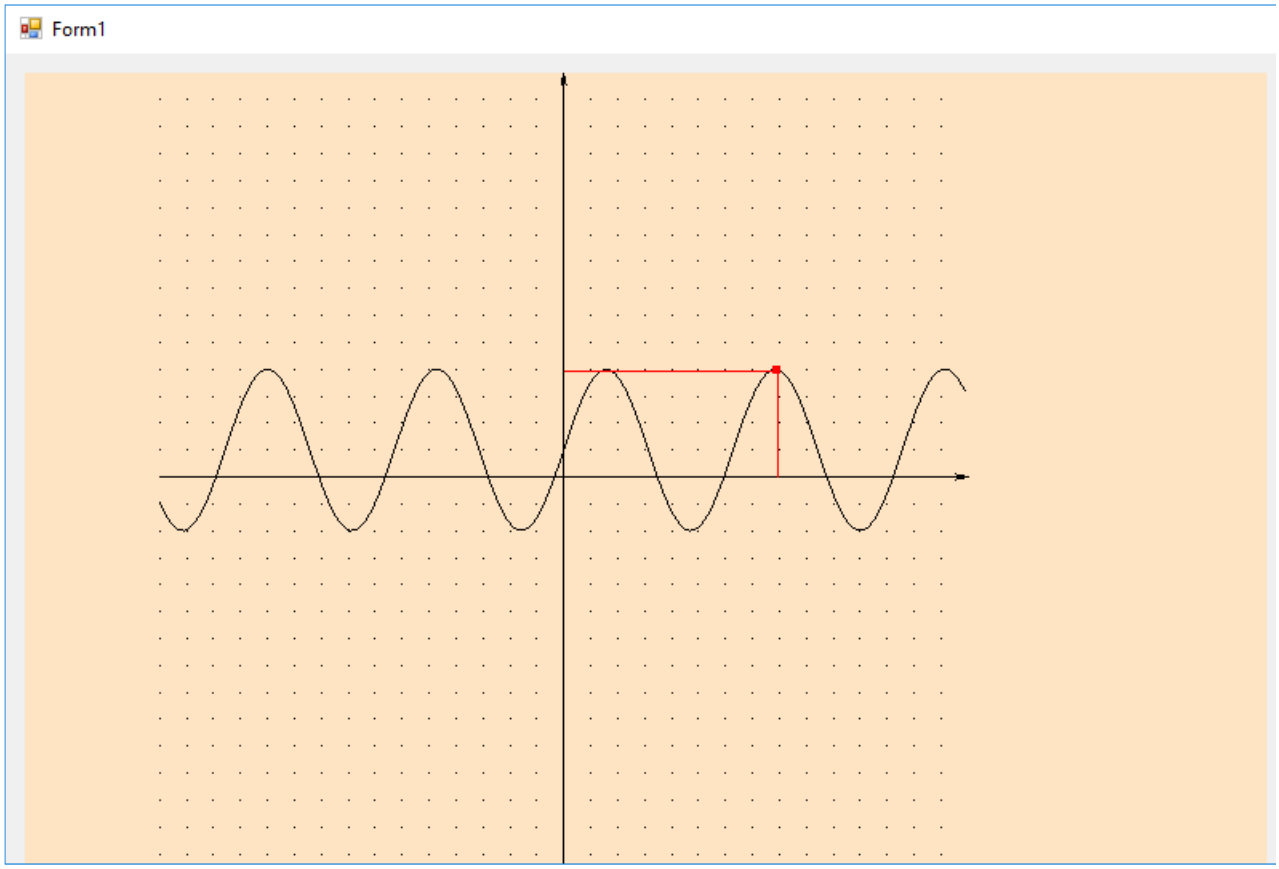


Рисунок 1.5 – Графік функції

Для цього відкрийте вікно ToolBox, знайдіть розділ Components, оберіть елемент Timer і перетягніть його на форму. Після цього у властивостях таймера встановіть інтервал, наприклад, 33 мілісекунди, щоб забезпечити потрібну частоту оновлення (рис. 1.6).

Місце, куди ви перетягнете елемент Timer, не має значення — цей компонент не займає простір на формі, а відобразиться в панелі об’єктів під формою як «невізуальний елемент».

Клацніть по таймеру, перейдіть до його властивостей і змініть такі параметри:

– Name — встановіть значення PointInGrap

– Interval — встановіть значення 30 (це означає, що кожні 30 мілісекунд буде викликатись подія OnTimer).

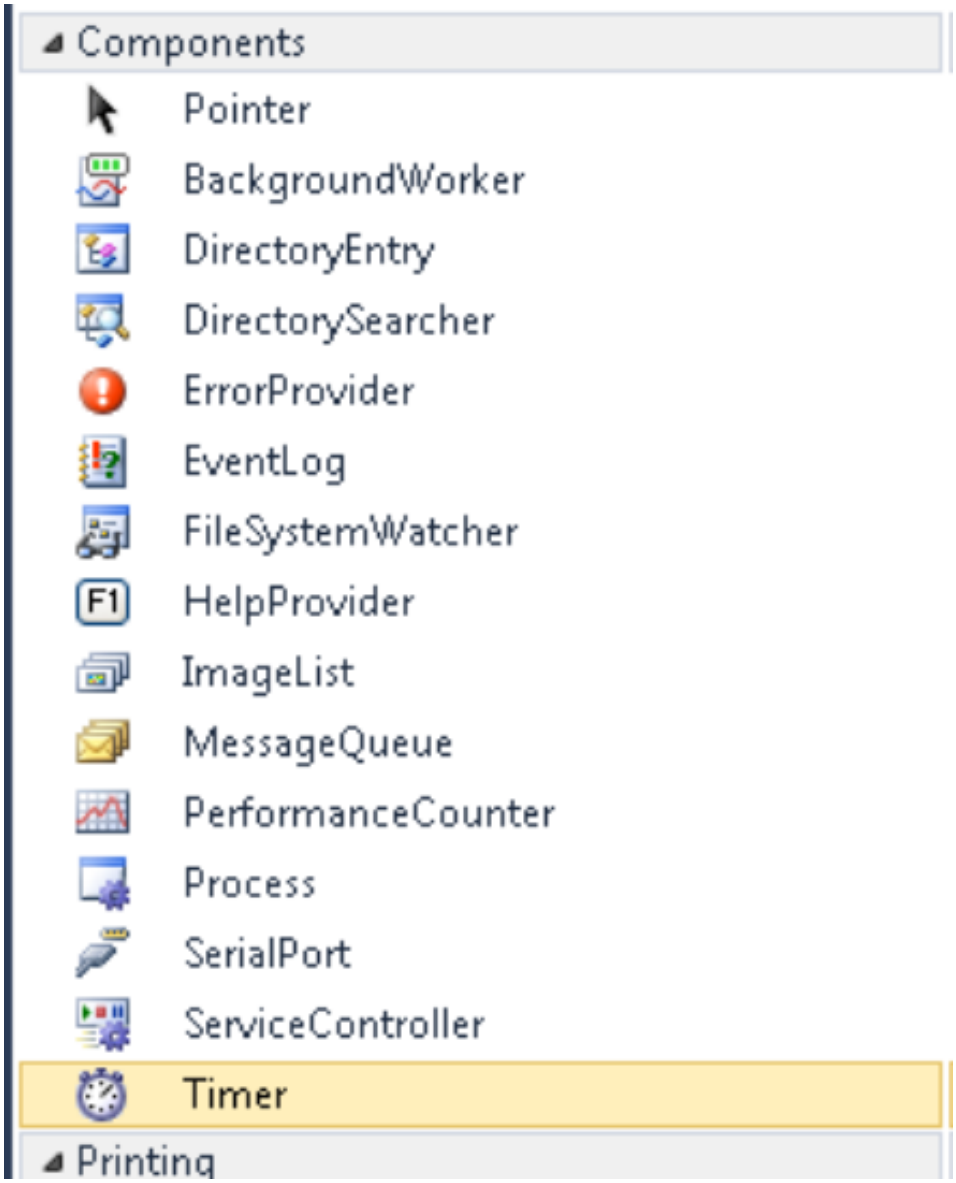


Рисунок 1.6 – Встановлення властивостей таймера

Після цього двічі клацніть по таймеру, щоб автоматично створити функцію PointInGrap_Tick, яка буде відповідати за оновлення стану анімації при кожному тіку таймера.

Тепер, перед кодом конструктора класу форми, додайте ініціалізацію наступних змінних — вони будуть потрібні для побудови графіка та анімації (рис. 1.7)

```
// Розміри вікна візуалізації
double ScreenW, ScreenH;

// Відношення сторін вікна для коректного перетворення координат
private float devX;
private float devY;

// Масив для зберігання координат точок графіка
private float[,] GrapValuesArray;

// Кількість елементів у масиві
private int elements_count = 0;

// Прапор, що означає, що масив ще не заповнений
private bool not_calculate = true;

// Номер елемента масиву для поточної позиції червоної точки
private int pointPosition = 0;

// Допоміжні змінні для побудови ліній від курсору миші до координатних осей
float lineX, lineY;

// Поточні координати курсору миші
float Mcoord_X = 0, Mcoord_Y = 0;
```

Рисунок 1.7 –Код ініціалізації наступних змінних

Перейдемо до конструктора і зробимо подвійне клацання лівою клавішею миші на формі - створиться функція обробник події завантаження форми. У ній ми помістимо код ініціалізації OpenGl (рис. 1.8).

```
private void Form1_Load(object sender, EventArgs e)
{
    // Визначення параметрів налаштування проекції залежно від співвідношення сторін елемента AnT
    if ((float)AnT.Width <= (float)AnT.Height)
    {
        ScreenW = 30.0;
        ScreenH = 30.0 * (float)AnT.Height / (float)AnT.Width;
        GL.Ortho(0.0, ScreenW, 0.0, ScreenH, -1, 1);
    }
    else
    {
        ScreenH = 30.0;
        ScreenW = 30.0 * (float)AnT.Width / (float)AnT.Height;
        GL.Ortho(0.0, ScreenW, 0.0, ScreenH, -1, 1);
    }

    // Збереження коефіцієнтів для переведення координат вказівника миші у координати OpenGL сцен
    devX = (float)ScreenH / (float)AnT.Width;
    devY = (float)ScreenW / (float)AnT.Height;

    // Встановлення об'єктно-відкової матриці (Modelview)
    GL.MatrixMode(MatrixMode.Modelview);

    // Запуск таймера, що викликатиме функцію оновлення анімації та візуалізації
    PointInGrap.Start();
}
```

Рисунок 1.8 – Код ініціалізації OpenGl

Ініціалізація OpenGL у програмі починається з налаштування області виводу, яка називається viewport. За допомогою функції GL.Viewport визначається, у якій частині елемента керування GLControl буде відображатися графіка. У нашому випадку область виводу займає весь простір елемента AnT, тобто відображення відбувається у всій площі вікна.

Далі відбувається налаштування матриці проекції, яка визначає, як тривимірні об'єкти проєктуються на двовимірний екран. Спершу встановлюється режим роботи з матрицею проекції за допомогою GL.MatrixMode. Потім поточна матриця скидається до одиничної за допомогою

GL.LoadIdentity, що означає її очищення. Після цього задається ортогональна проекція функцією GL.Ortho, яка встановлює межі видимого обсягу з урахуванням пропорцій вікна.

Наступним кроком є встановлення режиму роботи з матрицею моделі та виду (Modelview). Ця матриця використовується для позиціонування об’єктів у просторі та визначення положення камери. Так само, як і раніше, поточна матриця очищується, щоб почати з початкового стану.

Під час створення сцени спочатку позиціонується камера в просторі, визначаючи обсяг видимості. Потім розміщується модель (об’єкт), яка буде відображатися. Після цього встановлюється проекція — форма видимого обсягу. Нарешті, визначається область виводу, куди потрапить зображення сцени.

Таким чином, у процесі ініціалізації OpenGL готується сцена для коректного відображення графіки у вікні програми. Всі ці кроки забезпечують правильне позиціонування камери, налаштування видимої області та проекції, що є основою для подальшої візуалізації.

Функція PointInGrap_Tick виконується з інтервалом 30 мілісекунд і служить для керування анімацією руху червоної точки на графіку. Вона визначає, з якого елемента масиву координат зараз потрібно взяти значення для відображення точки.

Якщо позиція точки досягає останнього елемента масиву, функція повертає її індекс до початку, щоб анімація повторювалася циклічно. Далі викликається функція Draw, яка відповідає за малювання сцени — зокрема, червоної точки на графіку (рис. 1.9).

Після візуалізації позиція точки збільшується на одиницю, що забезпечує поступовий рух точки по масиву координат. Таким чином, програма плавно відображає зміну значень функції по всій видимій області графіку.

Цей механізм таймера та оновлення позиції точки дозволяє створити ефект анімації, що демонструє, як змінюються значення функції в реальному часі.

```
// Функція обробника події таймера
private void PointInGrap_Tick(object sender, EventArgs e)
{
    // Якщо досягли останнього елемента масиву, повертаємось до початку
    if (pointPosition == elements_count - 1)
        pointPosition = 0; // Починаємо анімацію заново

    // Виклик функції, яка малює сцену з червоною точкою
    Draw();

    // Переходимо до наступного елемента масиву для наступного кадру
    pointPosition++;
}
```

Рисунок 1.9 – Код функції PointInGrap_Tick

Тепер перед тим, як перейти до функцій, що відповідають за візуалізацію, ми розглянемо кілька невеликих допоміжних функцій.

Почнемо з функції AnT_MouseMove. Для обробки руху миші над елементом GLControl (AnT) створюється подія MouseMove. Це робиться через властивості елемента AnT у вкладці подій (Event), де додається обробник події MouseMove. У цій функції зберігаються поточні координати миші, які пізніше використовуються для візуалізації графіка. Крім того, обчислюються параметри ліній, які з'єднують поточне положення курсору з координатними осями.

Код функції виглядає так (рис. 1.10).

Наступна функція відповідає за обчислення координат для побудови графіка. У цій функції відбувається ініціалізація масиву координат та обчислення всіх точок графіка залежно від заданого діапазону значень x і кроку збільшення цих значень.

```
// Обробка руху миші над елементом AnT
private void AnT_MouseMove(object sender, MouseEventArgs e)
{
    // Зберігаємо координати миші
    Mcoord_X = e.X;
    Mcoord_Y = e.Y;

    // Обчислюємо параметри для домальовування ліній від покажчика миші до координатних осей
    lineX = devX * e.X;
    lineY = (float)(ScreenH - devY * e.Y);
}
```

Рисунок 1.10 – Код функції AnT_MouseMove

Важливо пам’ятати, що при ініціалізації масиву для зберігання координат потрібно вказати достатню кількість елементів. Якщо масив буде меншим за необхідний розмір, під час виконання програми відбудеться помилка через спробу доступу до пам’яті поза межами масиву.

Код функції обчислення координат наведено на рисунку 1.11.

Функція, яка виконує візуалізацію графіка, виділена в окрему підфункцію, щоб не перевантажувати основну функцію візуалізації сцени. Спочатку ця функція перевіряє прапорець not_calculate, який сигналізує, чи були вже обчислені координати графіка. Якщо координати ще не обчислені, викликається функція Calculation(), яка заповнює масив координат.

Після цього відбувається проходження циклом по масиву координат точок, і точки послідовно з'єднуються лініями, створюючи графік. В кінці функції малюється червона точка, яка показує поточне положення на графіку — координати для неї беруться з масиву за індексом pointPosition (рис. 1.12).

```
// Приватна функція для обчислення координат графіка
private void Calculation()
{
    // Локальні змінні для обчислень
    float x = 0, y = 0;

    // Ініціалізація масиву для зберігання 300 точок графіка (300 рядків, 2 стовпці: x і y)
    GrapValuesArray = new float[300, 2];

    // Лічильник кількості елементів масиву
    elements_count = 0;

    // Обчислення значень y для x у діапазоні від -15 до 15 з кроком 0.1
    for (x = -15; x < 15; x += 0.1f)
    {
        // Формула функції y = f(x), наприклад: y = x * x * (x * x) * (x - 2);
        y = x * x * x * x * (x - 2); // Приклад функції, можна замінити на іншу

        // Запис координат x та y у масив
        GrapValuesArray[elements_count, 0] = x;
        GrapValuesArray[elements_count, 1] = y;

        // Збільшення лічильника елементів
        elements_count++;
    }

    // Прапор, що сигналізує про те, що обчислення завершено
    not_calculate = false;
}
```

Рисунок 1.11 – Код функції обчислення координат

```
private void DrawGraph()
{
    // Перевірка, чи обчислені координати графіка
    if (not_calculate)
    {
        // Якщо ні, викликаємо функцію обчислення координат
        Calculation();
    }

    // Починаємо режим малювання лінії (з'єднання точок)
    GL.Begin(PrimitiveType.LineStrip);

    // Малюємо початкову точку графіка
    GL.Vertex2(GrapValuesArray[0, 0], GrapValuesArray[0, 1]);

    // Проходимо по масиву координат і малюємо лінію, з'єднуючи вершини
    for (int i = 1; i < elements_count; i++)
    {
        GL.Vertex2(GrapValuesArray[i, 0], GrapValuesArray[i, 1]);
    }

    // Завершуємо режим малювання лінії
    GL.End();

    // Встановлюємо розмір точки в 5 пікселів для червоної точки
    GL.PointSize(5);

    // Встановлюємо поточний колір - червоний
    GL.Color3(Color.Red);

    // Починаємо режим малювання точок
    GL.Begin(PrimitiveType.Points);

    // Малюємо червону точку за координатами поточного елемента масиву
    GL.Vertex2(GrapValuesArray[pointPosition, 0], GrapValuesArray[pointPosition, 1]);

    // Завершуємо режим малювання точок
    GL.End();

    // Повертаємо розмір точки до 1 пікселя (за замовчуванням)
    GL.PointSize(1);
}
```

Рисунок 1.12 – Код функції візуалізації графіка

Функція Draw відповідає за візуалізацію координатної сітки під графіком, координатних осей із стрілками, підписів осей, а також за виклик функції малювання самого графіка. Крім того, у цій функції виводяться координати миші з лініями, що з'єднують покажчик миші з координатними осями. Код функції наведено на рисунках 1.13 та 1.14.

```
private void Draw()
{
    // Встановлюємо колір заливки вікна виведення
    GL.ClearColor(Color.Bisque);

    // Очищення буфера кольору і буфера глибини
    GL.Clear(ClearBufferMask.ColorBufferBit | ClearBufferMask.DepthBufferBit);

    // Скидаємо поточну матрицю перетворень
    GL.LoadIdentity();

    // Встановлюємо чорний колір для малювання сітки та осей
    GL.Color3(0, 0, 0);

    // Зберігаємо поточний стан матриці
    GL.PushMatrix();

    // Здійснюємо переміщення по осях X і Y, щоб змістити початок координат
    GL.Translate(20, 15, 0);

    // Малюємо координатну сітку точками
    GL.Begin(PrimitiveType.Points);
    for (float ax = -15; ax < 15; ax += 1f)
    {
        for (float bx = -15; bx < 15; bx += 1f)
        {
            GL.Vertex2(ax, bx);
        }
    }
    GL.End();

    // Малюємо координатні осі з стрілками
    GL.Begin(PrimitiveType.Lines);

    // Вісь Y
    GL.Vertex2(0, -15);
    GL.Vertex2(0, 15);
```

Рисунок 1.13 – Код початку функції Draw

```
    // Вісь X
    GL.Vertex2(-15, 0);
    GL.Vertex2(15, 0);

    // Стрілка вгору (вісь Y)
    GL.Vertex2(0, 15);
    GL.Vertex2(0.1, 14.5);

    GL.Vertex2(0, 15);
    GL.Vertex2(-0.1, 14.5);

    // Стрілка вправо (вісь X)
    GL.Vertex2(15, 0);
    GL.Vertex2(14.5, 0.1);

    GL.Vertex2(15, 0);
    GL.Vertex2(14.5, -0.1);

    GL.End();

    // Викликаємо функцію малювання графіка
    DrawGraph();

    // Відновлюємо попередній стан матриці
    GL.PopMatrix();

    // Встановлюємо червоний колір для ліній, що з'єднують курсор миші з осями
    GL.Color3(Color.Red);

    // Малюємо лінії від покажчика миші до координатних осей
    GL.Begin(PrimitiveType.Lines);
    GL.Vertex2(lineX, 15);
    GL.Vertex2(lineX, lineY);

    GL.Vertex2(20, lineY);
    GL.Vertex2(lineX, lineY);
    GL.End();

    // Оновлюємо буфери виводу на елементі GLControl (AnT)
    AnT.SwapBuffers();
}
```

Рисунок 1.14 – Код завершення функції Draw

2 МОДЕЛЮВАННЯ ТРИВИМІРНИХ ОБ'ЄКТІВ

2.1 Короткі теоретичні відомості

Моделювання тривимірних об’єктів є важливою складовою сучасного комп’ютерного проектування та дизайну. Завдяки 3D-моделюванню користувачі можуть створювати реалістичні віртуальні об’єкти, які точно відображають форму, розміри та конструкційні особливості реальних виробів.

Графічні редактори є основними інструментами для створення та редагування технічної та художньої графіки. Вони забезпечують користувачам широкий набір засобів для побудови дво- та тривимірних об’єктів, що використовується в багатьох сферах — від дизайну й архітектури до машинобудування та виробництва.

Графічні редактори, такі як CorelDRAW, Adobe Illustrator або Photoshop, зазвичай орієнтовані на обробку растрової або векторної графіки та використовуються переважно для створення ілюстрацій, макетів і дизайнерських елементів. На відміну від них, CAD-системи — наприклад, AutoCAD, SolidWorks, CATIA чи Fusion 360 — призначені для точного інженерного проектування й моделювання об’єктів у двох і трьох вимірах.

Моделювання тривимірних об’єктів у CAD-системах ґрунтується на використанні просторових координат, різних типів геометричних примітивів, операцій редагування та спеціалізованих інструментів побудови. У середовищі AutoCAD доступні кілька підходів до 3D-моделювання: твердотільне, поверхневе та каркасне. Розуміння принципів побудови та обробки моделей у CAD-системах дозволяє ефективно виконувати проєктні завдання та створювати точну технічну документацію для виробництва.

3D-моделювання — це процес створення тривимірних моделей об’єктів у віртуальному просторі з використанням комп’ютерних програм. Такі моделі дозволяють більш наочно відтворювати реальні або уявні

об’єкти, спрощуючи їх розробку, аналіз та візуалізацію.

В AutoCAD моделювання у трьох вимірах базується на використанні просторових координатних осей: X, Y та Z. Це дозволяє точно визначати положення точок, ліній, площин та об’ємів у просторі, що є основою для побудови складних геометричних форм.

AutoCAD підтримує кілька підходів до створення 3D-об’єктів, серед яких найбільш поширеним є **твердотільне моделювання (Solid Modeling)**. Такий метод дозволяє створювати об’єкти, що мають фізичний об’єм і можуть відображати реальні властивості матеріалів (рис. 2.1).

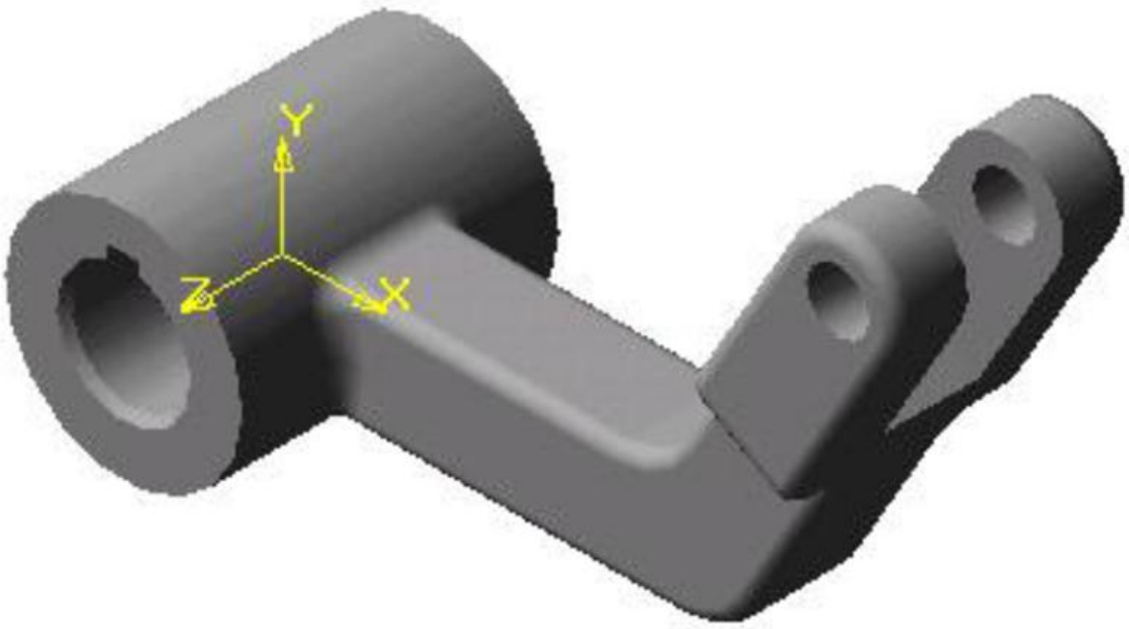


Рисунок 2.1 – Твердотільне моделювання (Solid Modeling)

Ще одним підходом є **поверхневе моделювання (Surface Modeling)**. У цьому випадку моделі описуються як набір поверхонь, що формують зовнішню оболонку об’єкта, проте не мають внутрішньої структури. Такий метод часто використовується для створення складних обтічних форм, які важко описати твердотільним методом (рис. 2.2).

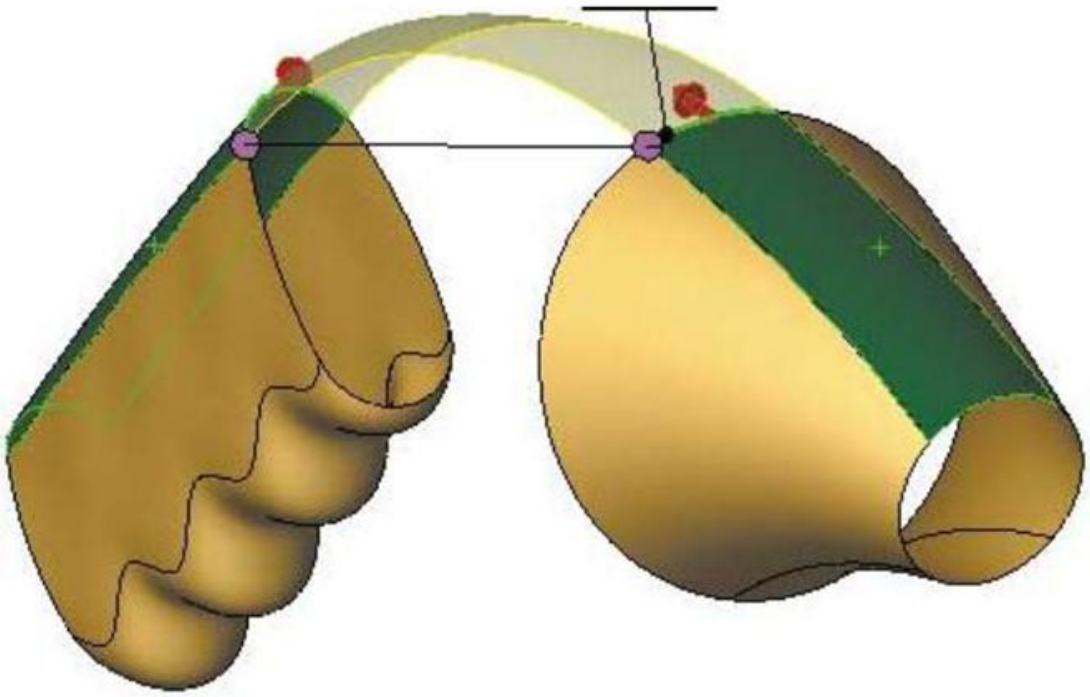


Рисунок 2.2 – Поверхневе моделювання (Surface Modeling)

Каркасне моделювання (Wireframe Modeling) є найпростішим способом 3D-моделювання. Об’єкти створюються шляхом з’єднання ліній та вершин у просторі, утворюючи каркас без поверхонь і об’єму (рис. 2.3). Цей метод зручний для швидкого конструювання контурів і геометричних схем.

Вибір методу моделювання залежить від завдань проектування, вимог до деталізації та подальшого використання моделей. Застосування 3D-моделювання в AutoCAD дозволяє створювати точні та наочні проекти, що значно полегшує роботу інженерів, архітекторів та дизайнерів.

Рисунок 2.3 – Каркасне моделювання (Wireframe Modeling)

Світова координатна система (WCS) є базовою глобальною системою координат в AutoCAD. Вона задається під час створення креслення і визначає загальну орієнтацію об’єктів у просторі за допомогою трьох взаємно перпендикулярних осей: X, Y та Z. Усі побудови та розміри спочатку орієнтовані саме відносно WCS.

Для зручності моделювання користувач може створювати власні координатні системи. **Користувацька координатна система (UCS)** дозволяє змінювати положення й орієнтацію координатних площин залежно від потреб проекту. Це спрощує побудову об’єктів під різними кутами та на складних поверхнях.

Завдяки UCS можна швидко переключатися між різними площинами креслення або моделювання, точно визначати нові опорні точки й орієнтувати геометрію відповідно до складної форми моделі. Гнучке використання координатних систем робить процес 3D-моделювання більш ефективним і зручним для користувача.

В AutoCAD для створення тривимірних моделей використовуються стандартні геометричні фігури, які називаються примітивами.

До основних примітивів належать куб, сфера, циліндр, конус, тор, піраміда та клин (рис. 2.4). Ці базові об’єкти слугують відправною точкою для створення більш складних тривимірних моделей, оскільки з них можна комбінувати будь-які геометричні форми.

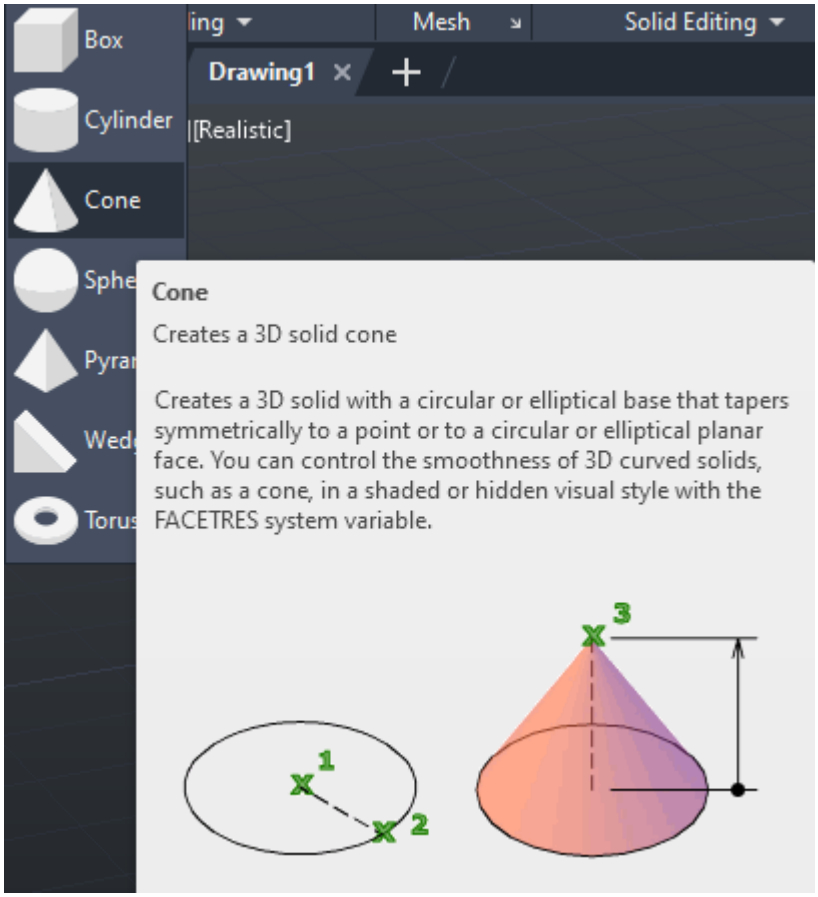


Рисунок 2.4 – Основні примітиви

Куб є універсальним примітивом для моделювання паралелепіпедів і прямокутних деталей. Сфера використовується для побудови округлих елементів, куль, частин механізмів або декоративних деталей. Циліндр часто застосовують для моделювання труб, валів та інших об’єктів з круглим перерізом.

Конус дозволяє створювати загострені або конусоподібні форми, які потрібні, наприклад, для виготовлення наконечників або деталей машин.

Тор використовується для моделювання ободів, кілець або декоративних елементів. Піраміда і клин дозволяють будувати похилі або загострені конструкції, які важко створити за допомогою інших примітивів.

Крім простого використання, примітиви можна комбінувати між собою, змінювати їхні розміри, обертати, копіювати та редагувати за допомогою спеціальних команд. Це робить їх надзвичайно гнучким інструментом для побудови складних інженерних моделей, архітектурних об’єктів або елементів дизайну.

Таким чином, використання примітивів дозволяє швидко та ефективно створювати як прості, так і складні тривимірні об’єкти, що є основою сучасного 3D-моделювання в AutoCAD.

Для редагування тривимірних тіл в AutoCAD застосовують різноманітні операції, які допомагають формувати складну геометрію шляхом поєднання чи модифікації простих об’єктів. Такі операції дозволяють швидко отримувати нові форми без необхідності моделювати їх з нуля.

Найпоширенішою операцією є **об’єднання (Union)**. За її допомогою можна поєднати кілька окремих об’єктів у єдине суцільне тіло. Це зручно, коли потрібно об’єднати окремі деталі в одну складову частину конструкції або модель, що полегшує подальше редагування (рис. 2.5).

Інша важлива операція — **віднімання (Subtract)**. Вона дозволяє від одного тіла відняти об’єм іншого. Наприклад, можна зробити отвір у кубі, віднявши від нього циліндр. Це дуже корисно для створення пазів, отворів, виїмок і складних внутрішніх порожнин у деталях (рис. 2.6).

Операція **перетин (Intersect)** використовується для виділення спільної частини двох або більше тіл. У результаті зберігається лише об’єм, що належить всім вибраним об’єктам одночасно. Цей інструмент ефективний для створення складних контурів і вузлів, де важливо визначити зону перетину конструктивних елементів (рис. 2.7).

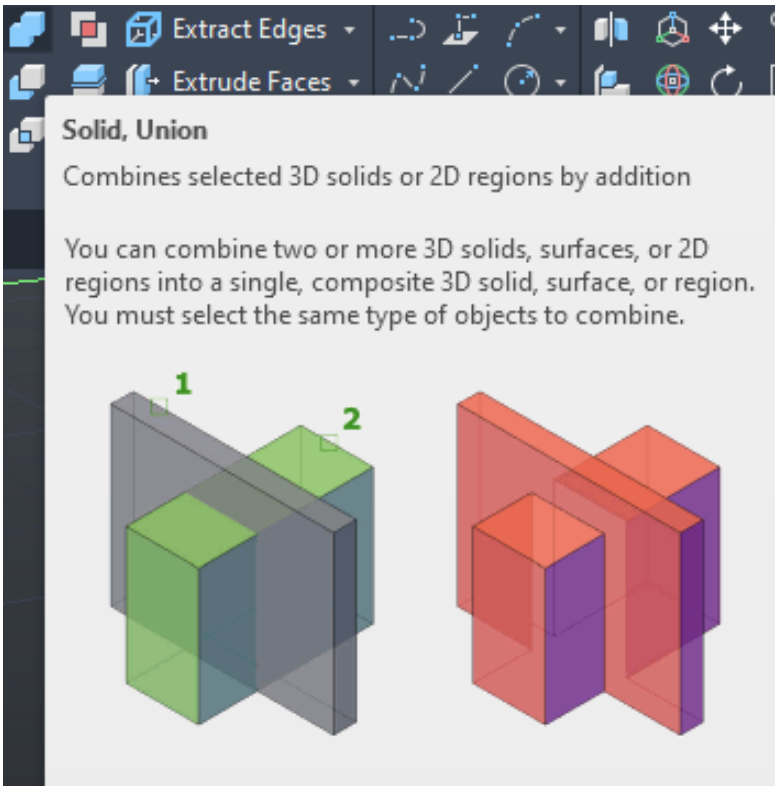


Рисунок 2.5 – Об’єднання (Union)

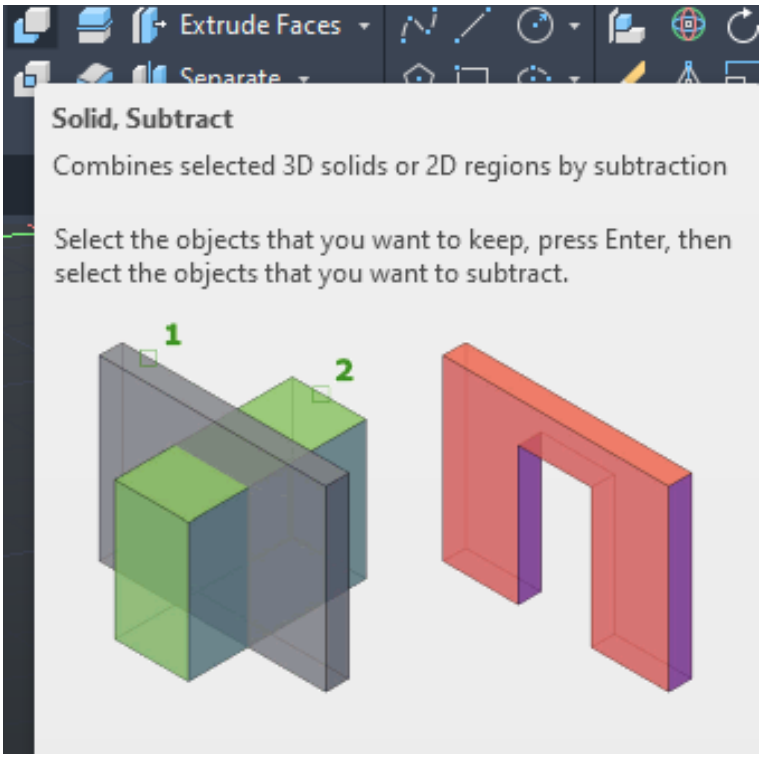


Рисунок 2.6 – Віднімання (Subtract)

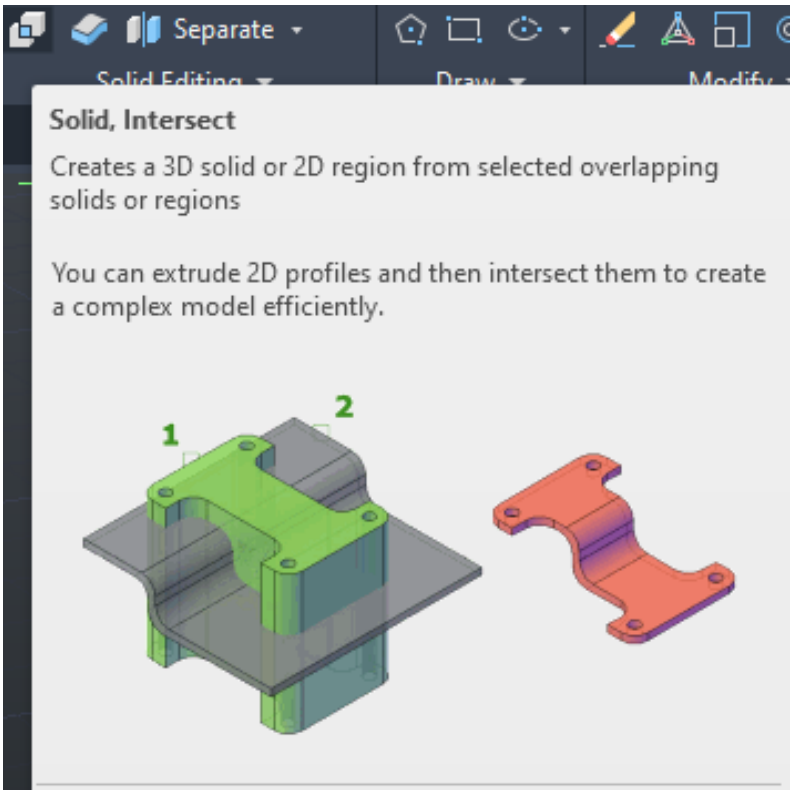


Рисунок 2.7 – Перетин (Intersect)

Крім основних, у редагуванні тривимірних тіл можна використовувати й інші операції: розділення тіл на частини, обтинання зайвих ділянок, а також поєднання операцій для побудови складних форм. Застосування таких інструментів робить процес моделювання гнучким та дозволяє втілювати різноманітні інженерні й дизайнерські рішення.

Серед інструментів побудови важливу роль відіграє **команда Extrude**. Вона використовується для витягування плоского профілю уздовж прямої лінії, перетворюючи двовимірний контур на тривимірну фігуру з заданою висотою. Завдяки цьому можна швидко створювати прості об’ємні об’єкти, такі як балки, колони, коробчасті конструкції або будь-які деталі з постійним поперечним перерізом (рис. 2.8).

Команда Extrude підтримує також додаткові можливості: можна задавати кут нахилу стінок (taper angle), що дозволяє робити конусоподібні

форми, або створювати витягування вздовж криволінійної траєкторії у поєднанні з командою Path. Це робить інструмент універсальним для побудови як простих, так і більш складних об’ємних моделей.

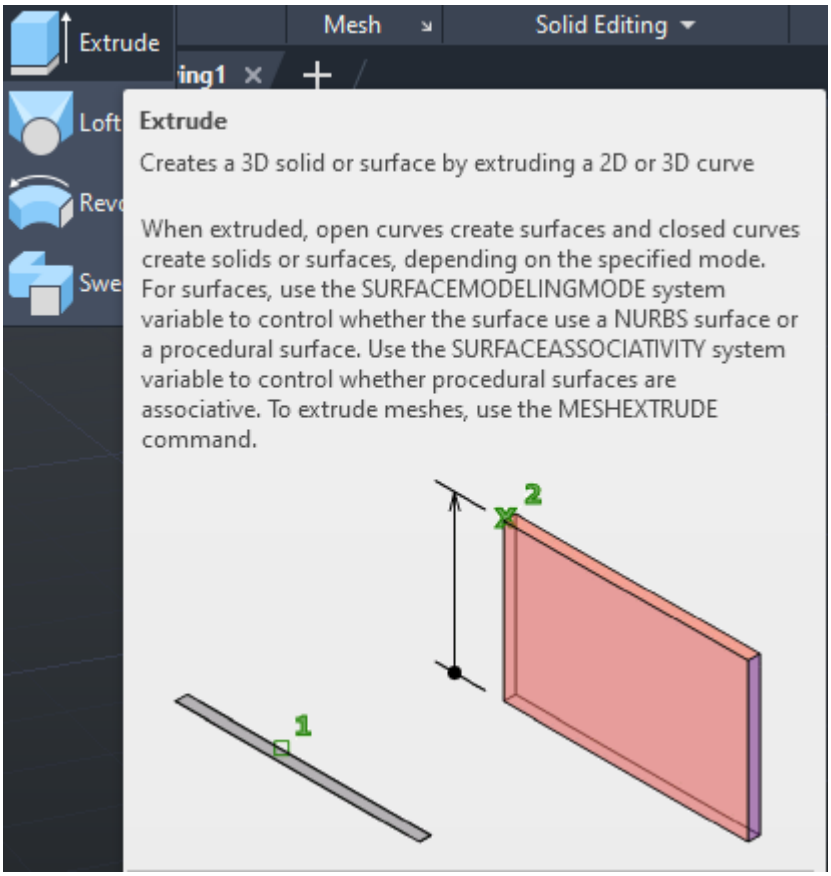


Рисунок 2.8 – Команда Extrude

Використання команди Extrude значно пришвидшує процес моделювання, оскільки дозволяє перетворювати креслення та контури у повноцінні 3D-об’єкти буквально за кілька кроків. Такий підхід широко застосовується в архітектурному проектуванні, машинобудуванні та дизайні інтер’єрів.

Команда Revolve дозволяє створювати тіла обертання шляхом обертання плоского профілю навколо заданої осі. У результаті формується тривимірний об’єкт, симетричний відносно цієї осі. Такий метод побудови

особливо зручний для моделювання деталей, що мають обертальну симетрію, — наприклад, ваз, дисків, колес, шківів, пляшок, ручок, лінз тощо (рис. 2.9).

Профіль, який обертається, може бути довільної форми, і чим складнішим є його контур, тим цікавішу і складнішу форму можна отримати після обертання. Осі обертання можуть бути як горизонтальними, так і вертикальними, або навіть похилими, що дає змогу створювати асиметричні елементи.

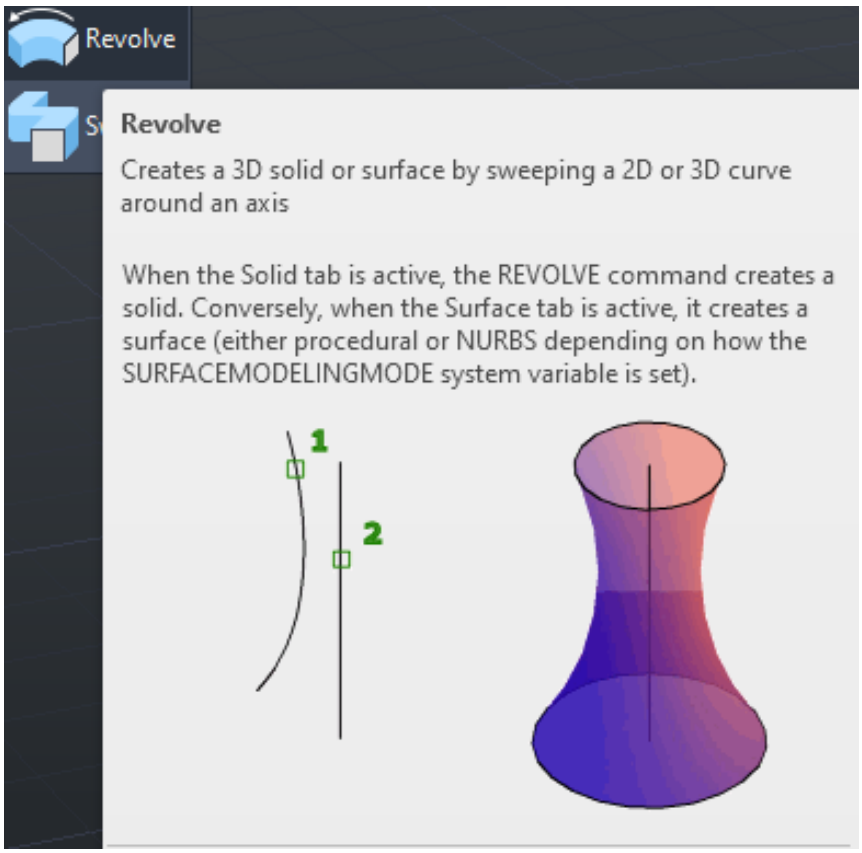


Рисунок 2.9 – Команда Extrude

Користувач може задати кут обертання (наприклад, 360° для повного тіла або менше — для сегментів), що дозволяє будувати часткові об’єкти, вирізи або декоративні елементи. Також доступна можливість створення тіла зі зсувом або обертання навколо відкритих дуг.

Команда Revolve є потужним інструментом у 3D-моделюванні, особливо в машинобудуванні, при створенні деталей веретеноподібної форми, а також у промисловому дизайні та ювелірному проектуванні.

За допомогою **інструмента Sweep** можна витягнути плоский профіль уздовж довільної кривої траєкторії, утворюючи складні тривимірні форми. Цей метод особливо корисний, коли об’єкт має повторюваний поперечний переріз, але складну просторову форму або вигнуту лінію (рис. 2.10).

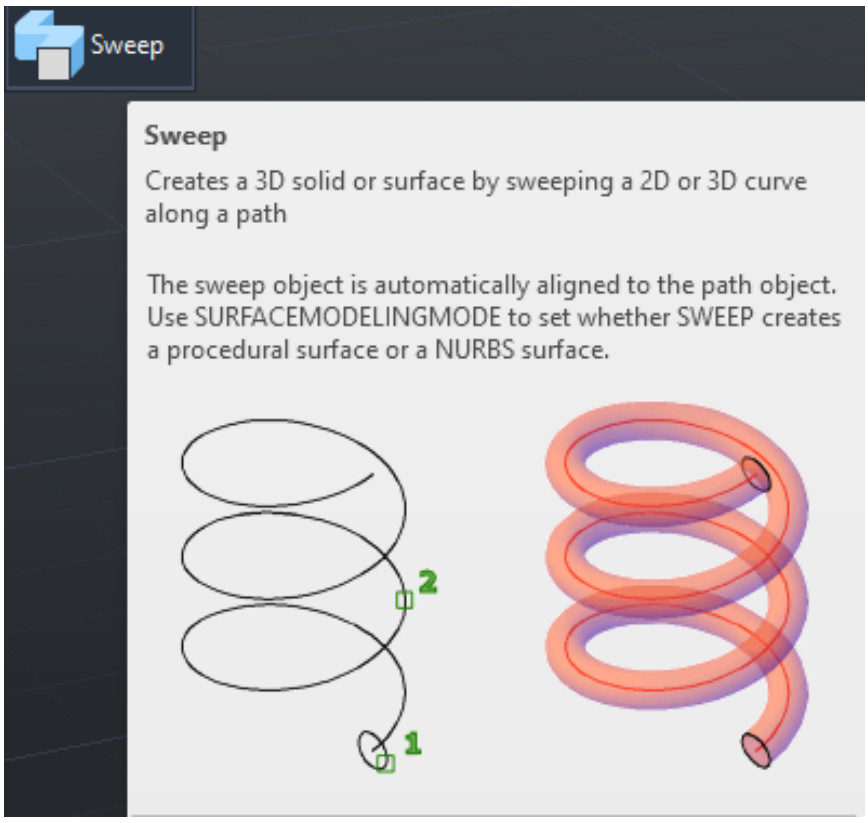


Рисунок 2.10 – Команда Extrude

Найчастіше Sweep застосовують для моделювання трубопроводів, кабелів, дротів, поручнів, архітектурних молдингів, трубок і шлангів. Користувач спочатку створює контур профілю (наприклад, круг або довільну замкнену криву) і окремо — шлях (траєкторію), яким цей профіль «прокочується» у просторі.

Інструмент дозволяє працювати з прямими, кривими та навіть складними сплайнами як траєкторіями, що забезпечує гнучкість у проектуванні нестандартних геометрій. Також можна регулювати параметри орієнтації профілю під час руху вздовж шляху, що дає змогу уникнути викривлень або небажаних перекручувань.

Sweep є надзвичайно потужним засобом для створення конструктивних і декоративних елементів із повторюваним поперечним перерізом, особливо у сфері будівництва, машинобудування та інтер'єрного дизайну.

Команда Loft дає можливість побудувати складні об'єкти, які плавно проходять через кілька профілів, розташованих у просторі. Такий спосіб моделювання особливо цінний для створення об'єктів зі змінним поперечним перерізом, де форма поступово переходить від одного профілю до іншого (рис. 2.11).

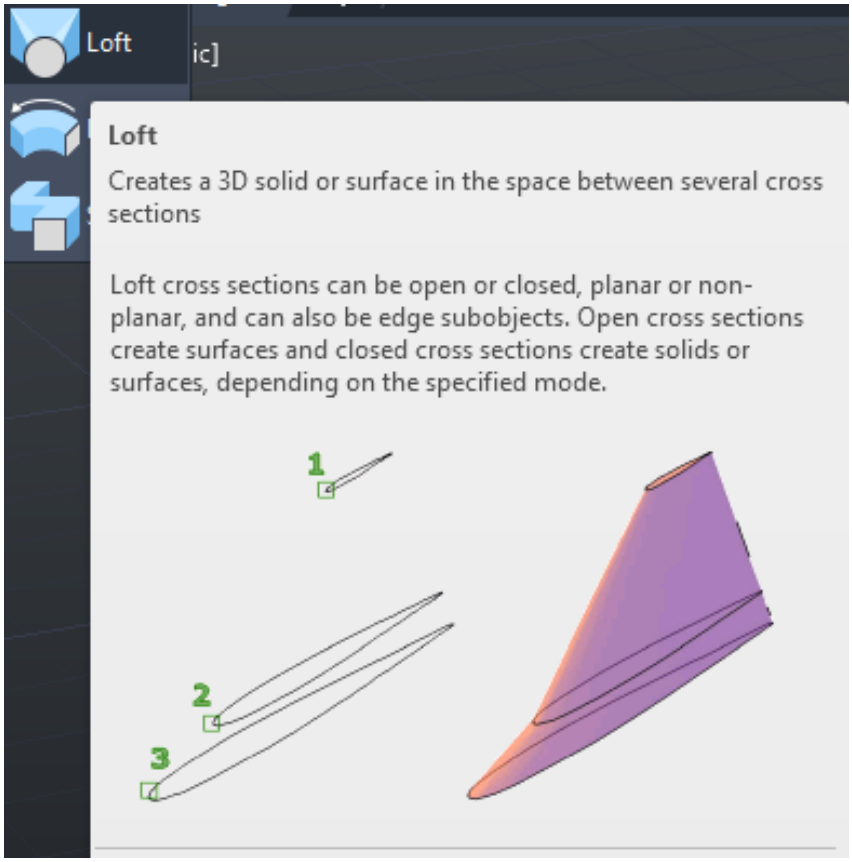


Рисунок 2.11 – Команда Extrude

Профілі для Loft можуть мати будь-яку замкнену або відкриту форму — від простих кіл і прямокутників до складних кривих і сплайнів. Розміщуючи їх на різних рівнях або у довільних положеннях у просторі, користувач формує каркас, за яким AutoCAD автоматично будує плавну поверхню або тверде тіло.

Крім основних профілів, можна використовувати напрямні криві (Guide Curves) для точнішого контролю форми між профілями або «доріжки» (Path) для зміни траєкторії між перетинами. Це дозволяє отримувати органічні, обтічні та ергономічні форми, які важко або неможливо створити іншими командами.

Loft широко використовується в промисловому дизайні, автомобілебудуванні, суднобудуванні та у сфері архітектурних проектів, де потрібно моделювати кузови, корпуси, обтічники чи дизайнерські оболонки складної геометрії.

Крім основних інструментів побудови, для деталізації та обробки 3D-моделей в AutoCAD використовують додаткові команди: **фаски (Chamfer)** — створюють скошені краї на ребрах; **скруглення (Fillet)** — виконують плавні заокруглення кутів; **обрізка (Slice)** — розрізає тіло на частини для подальшого редагування або аналізу. Ці команди допомагають надавати моделям точності, реалістичності та відповідності виробничим вимогам.

2.2 Приклад виконання індивідуального завдання

Розглянемо приклад побудови деталі в AutoCAD показаної на рисунку 2.12.

Для побудови цієї деталі в AutoCAD спочатку переключаємося у відповідний режим тривимірного моделювання та встановлюємо вигляд зверху для зручності побудови основи. Далі створюємо прямокутник із розмірами, зазначеними на кресленні, наприклад 120 на 80 міліметрів, і розмі-

щуємо його у потрібній точці координатної системи. Після цього застосовуємо команду витягування, щоб надати плиті товщину 15 міліметрів, утворюючи основу деталі (рис. 2.13).

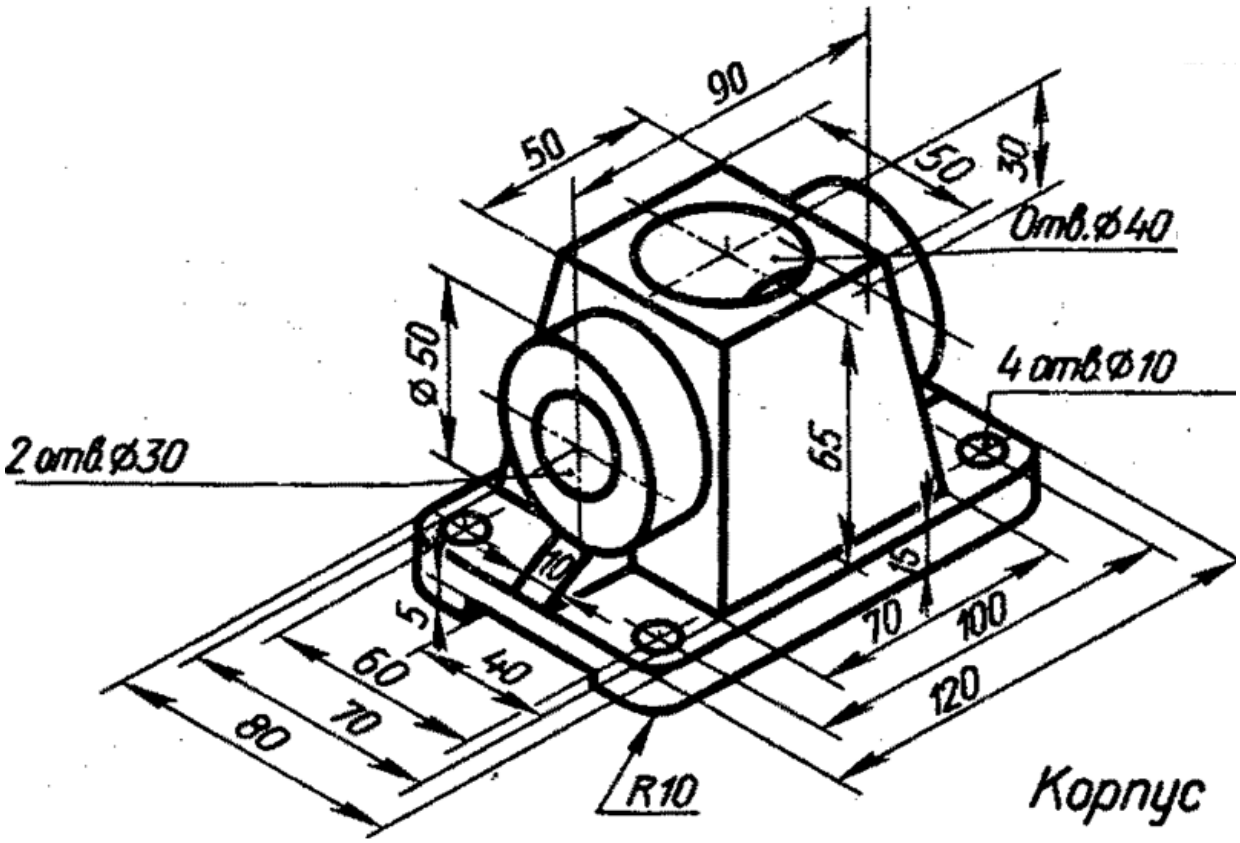


Рисунок 2.12 – Приклад деталі

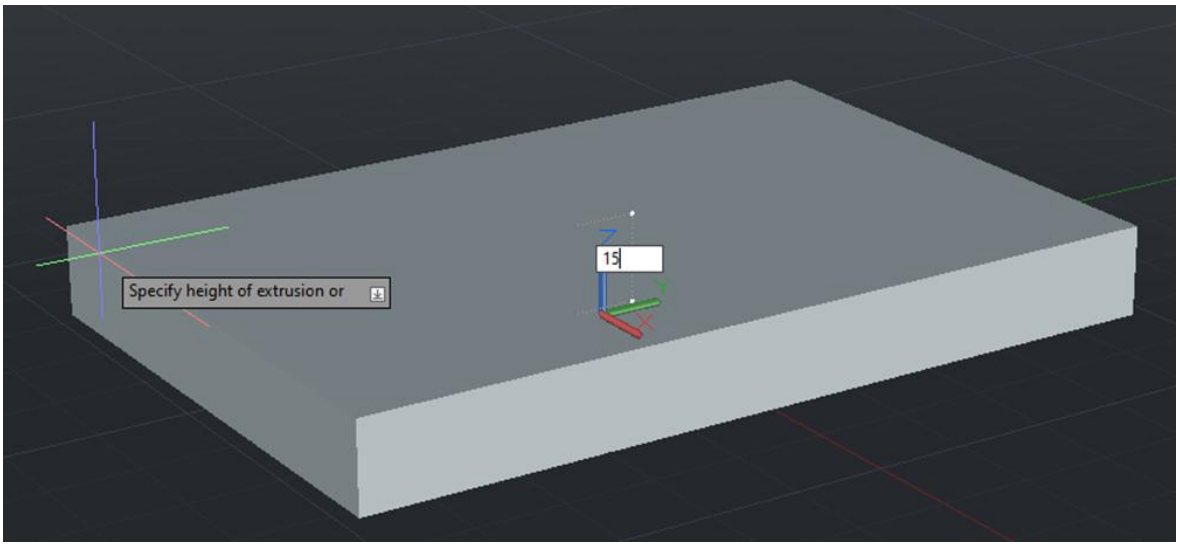


Рисунок 2.13 – Витягування основи

Далі, щоб створити центральну похилу частину деталі у вигляді зрізаної піраміди, використовуємо команду Loft. Для цього на верхній площині основної плити креслимо два квадрати: перший квадрат зі стороною 70 мм (розташований біля основи) і другий квадрат зі стороною 50 мм (розташований вище, на потрібній висоті, наприклад на 65 мм від бази) (рис. 2.14). Розміщуємо обидва квадрати співвісно один над одним. Потім застосовуємо команду Loft, вибираємо обидва профілі — і AutoCAD автоматично побудує тіло, що плавно з'єднує обидві фігури, утворюючи похилу пірамідальну частину корпусу.

Після побудови основної плити та похилої піраміди потрібно зробити їх єдиним тілом. Для цього використовуємо команду Union. Спочатку виділяємо обидва об'єкти — плиту (основу) та Loft-піраміду — потім виконуємо команду Union (Об'єднання) (рис. 2.15). В результаті ці два тіла об'єднаються в одне суцільне тривимірне тіло. Цей крок важливий для подальшого створення інших деталей моделі, оскільки всі модифікації будуть застосовуватись до єдиного об'єкта.

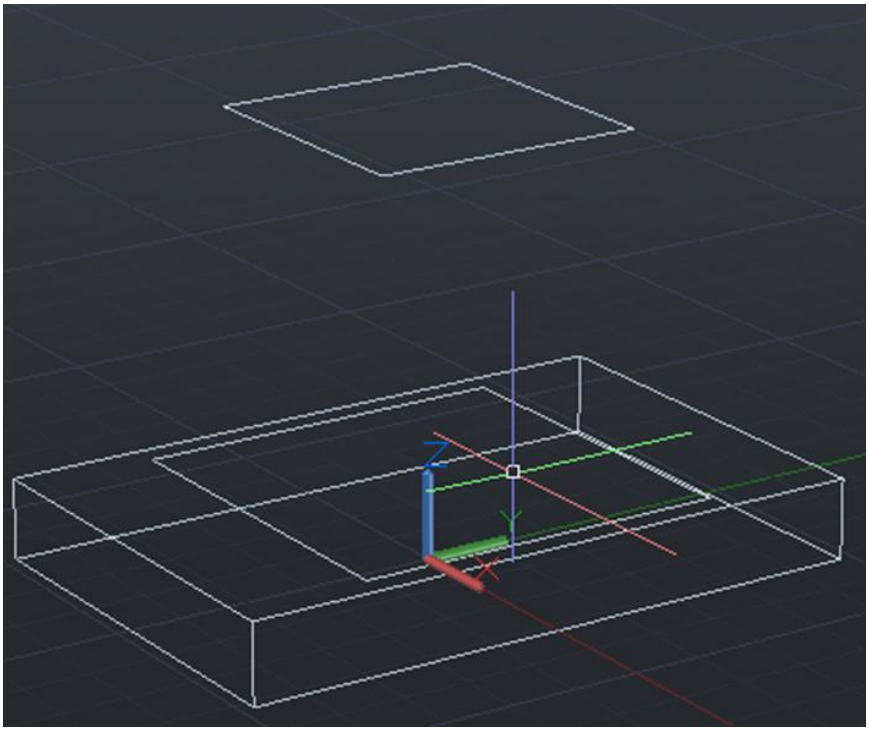


Рисунок 2.14 – Побудова піраміди командою Loft

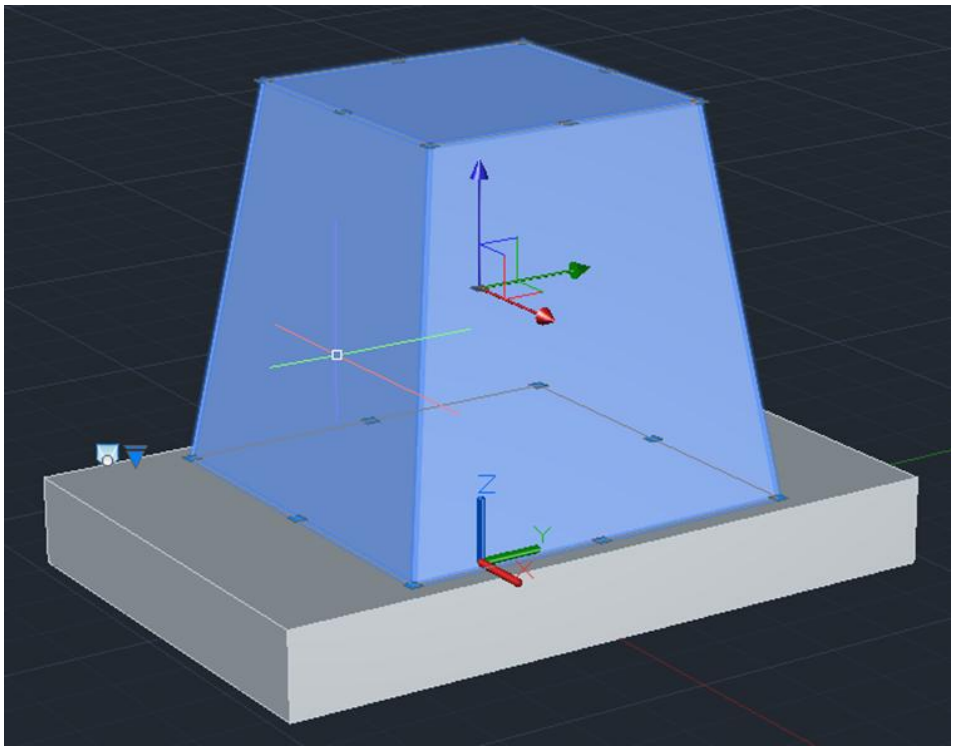


Рисунок 2.15 – Команда Union

Для зручності побудови наступних елементів моделі перемикаємо вигляд у проекцію Front (Спереду) (рис. 2.16). Це дозволяє зручно працювати з об’єктами у площині YZ та правильно розміщувати нові профілі, отвори чи допоміжні контури відносно вже створеної моделі.

Перейшовши у проекцію Front, створюємо нове коло діаметром 50 мм (рис. 2.17). Його центр розміщуємо у заданому місці згідно з кресленням. Це коло буде використовуватись для створення додаткового циліндричного елемента (рис. 2.18).

Для побудови ребра жорсткості видавимо невеличкий прямокутник та скористаємося командою Fillet Edge (Кромка), щоб згладити або скосити краї ребра (рис. 2.19).

Після створення ребра жорсткості використовуємо команду 3D Mirror (Відзеркалення у просторі), щоб швидко отримати друге ребро симетрично відносно центральної площини деталі (рис. 2.20). Для цього вибираємо побудоване ребро, задаємо площину симетрії (наприклад, центральну вертикальну площину моделі) — і AutoCAD створює копію ребра з протилежного боку.

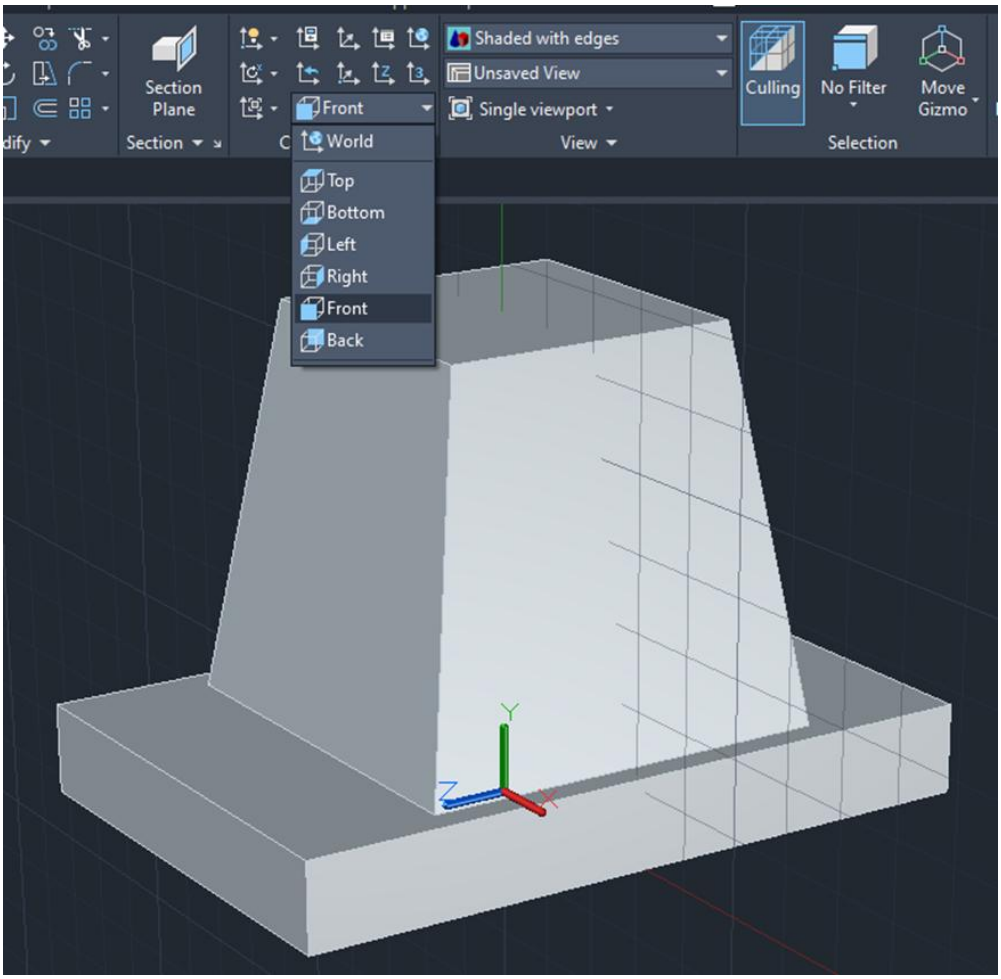


Рисунок 2.16 – Перемикання вигляду у проекцію Front (Спереду)

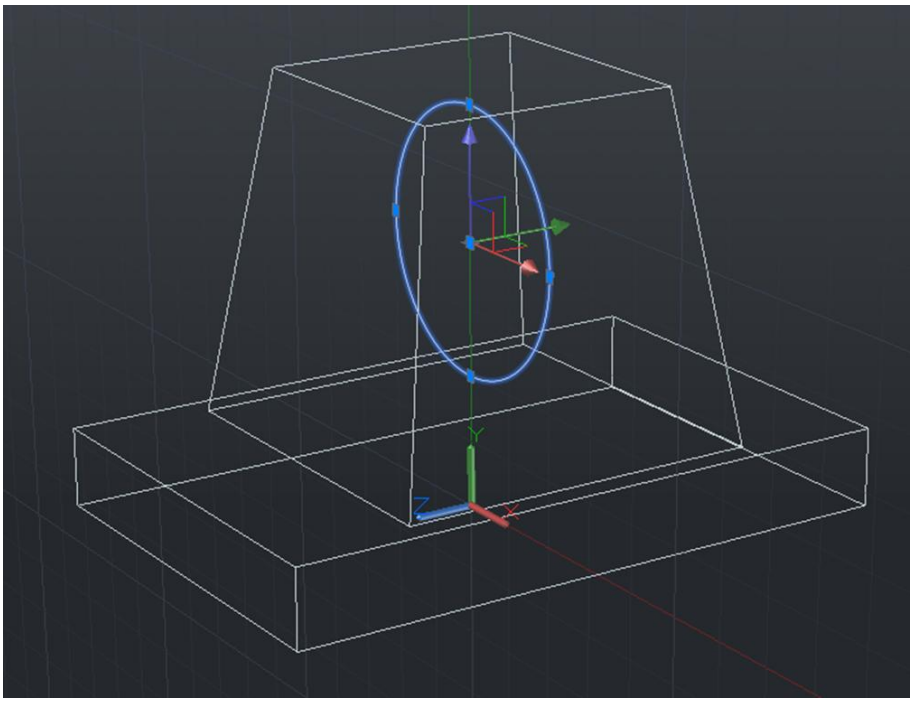


Рисунок 2.17 – Створення кола діаметром 50 мм

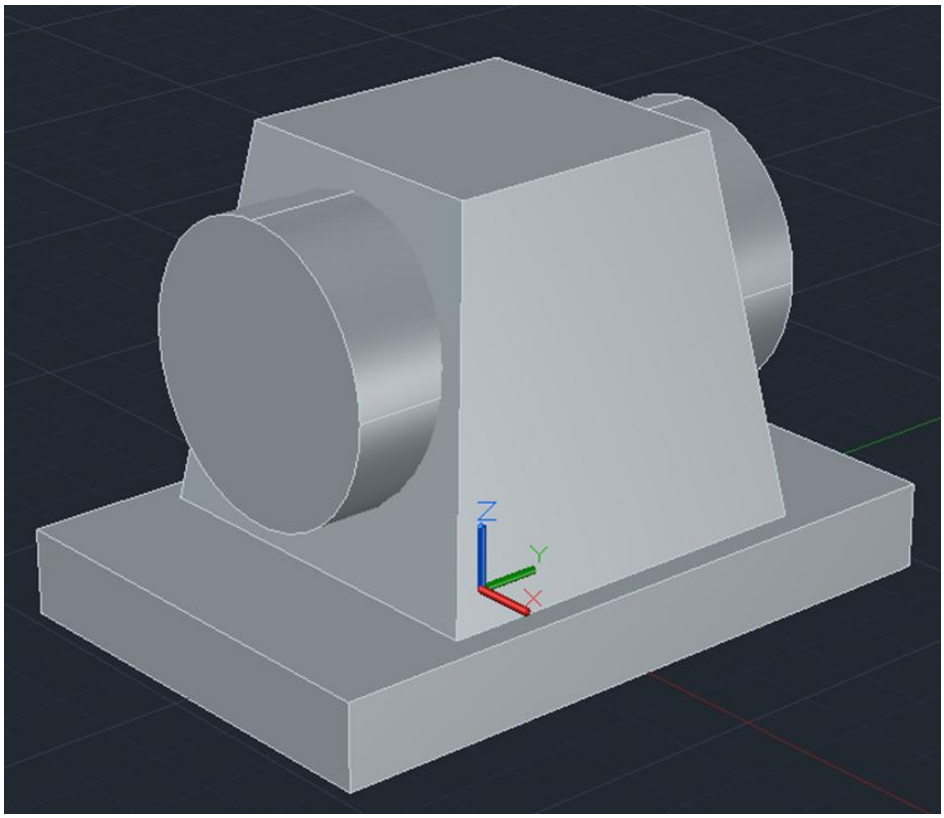


Рисунок 2.18 – Додатковий циліндричний елемент

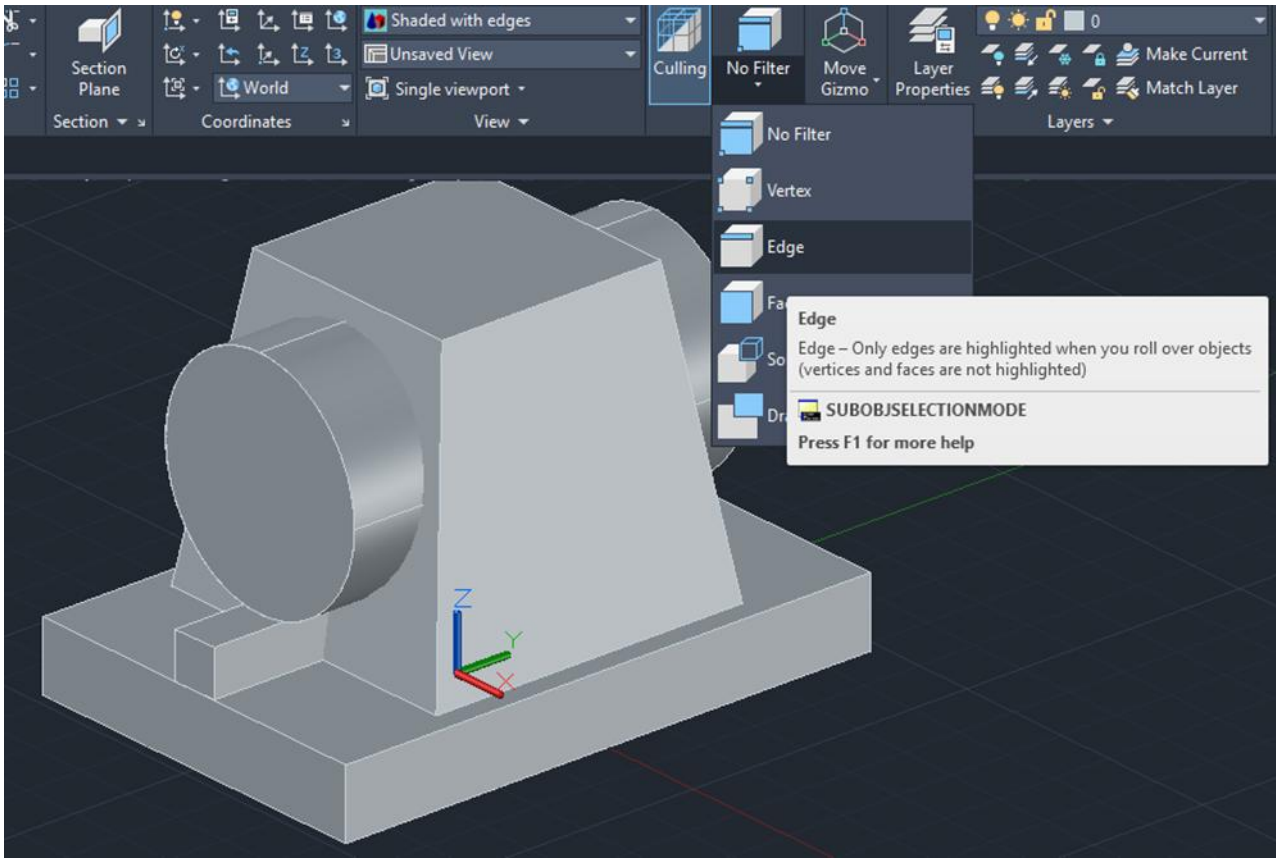


Рисунок 2.19 – Команда Fillet Edge (Кромка)

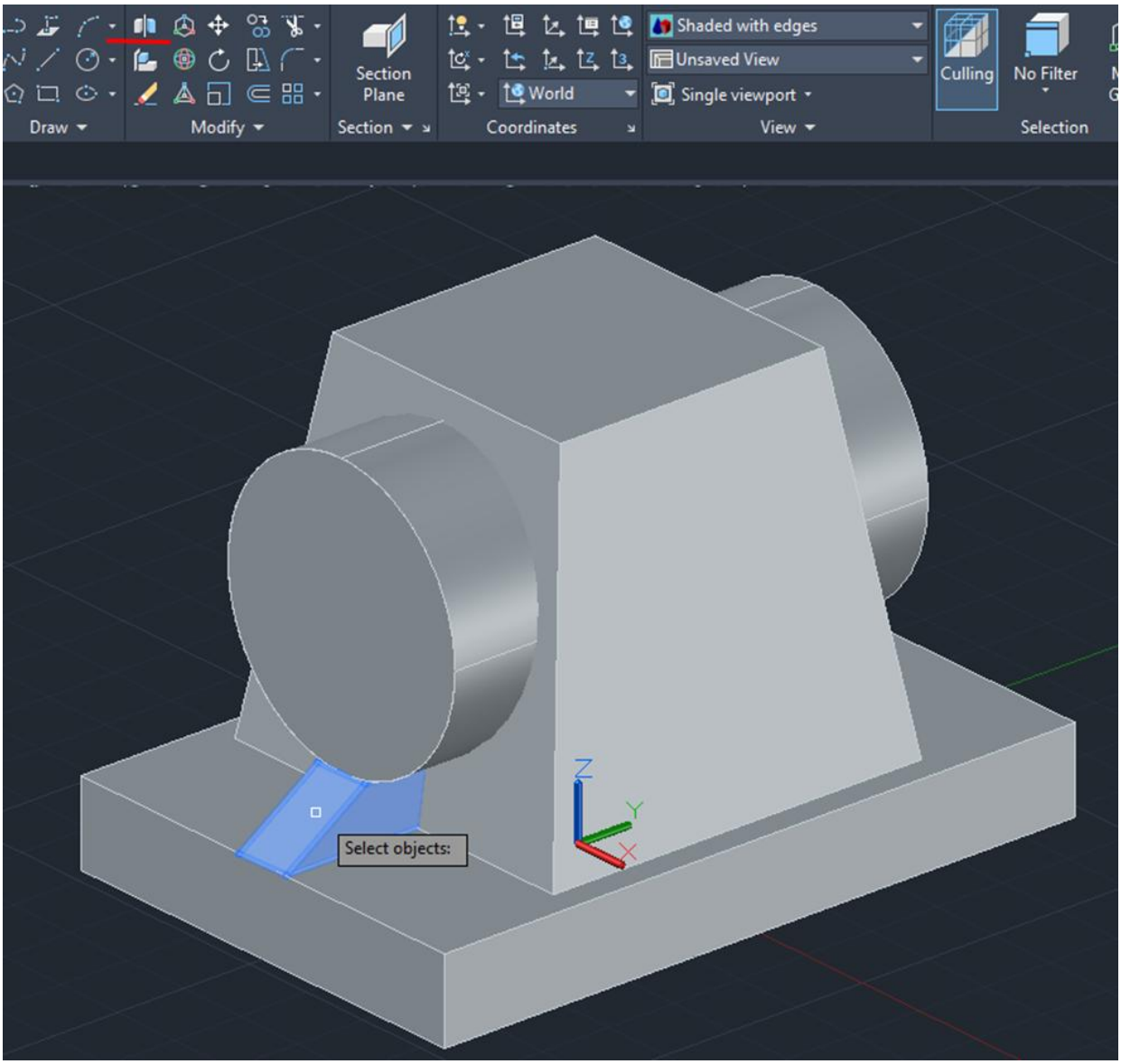


Рисунок 2.20 – Команау 3D Mirror (Відзеркалення у просторі)

Робимо всі необхідні отвори: для цього у потрібних місцях моделі креслимо відповідні профілі (кола, прямокутники чи інші контури), після чого використовуємо команди Extrude та Subtract, щоб витягнути профілі крізь тіло й вирізати матеріал. Такий спосіб дозволяє швидко та точно створити наскрізні або глухі отвори потрібної форми й розміру, відповідно до вимог креслення та конструкції деталі (рис. 2.21).

На остаток закруглюємо грані основи за допомогою команди Fillet Edge (Закруглити кромку) (рис. 2.22). Готова тривимірна модель показана на рисунку 2.23.

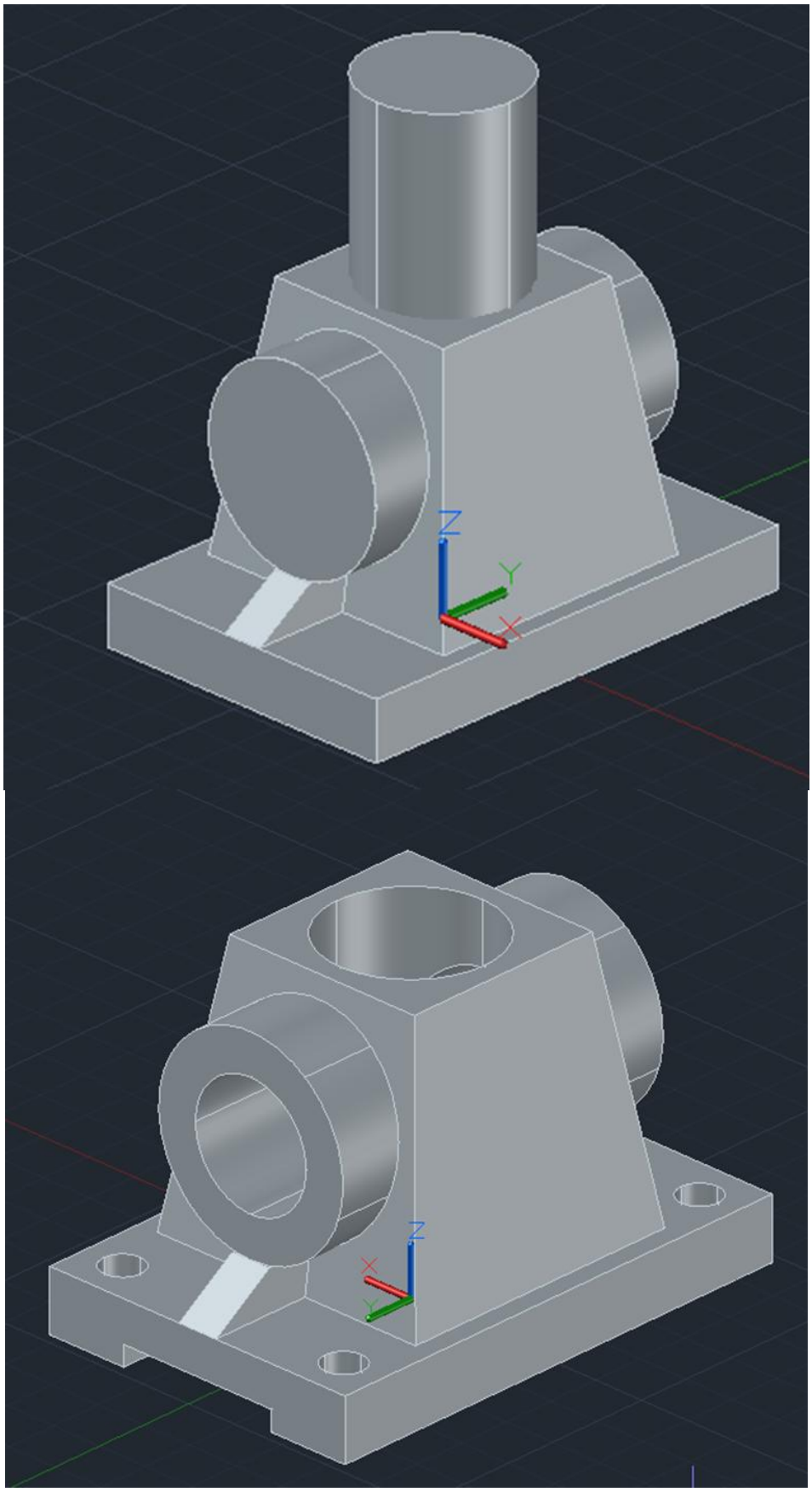


Рисунок 2.21 – Команди Extrude та Subtract

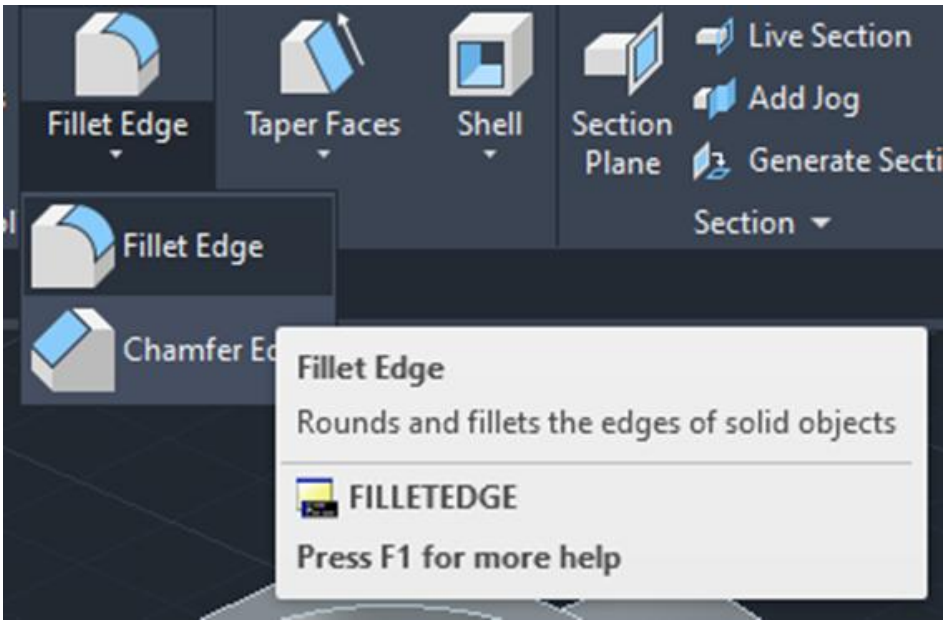


Рисунок 2.22 – Команда Fillet Edge (Закруглити кромку)

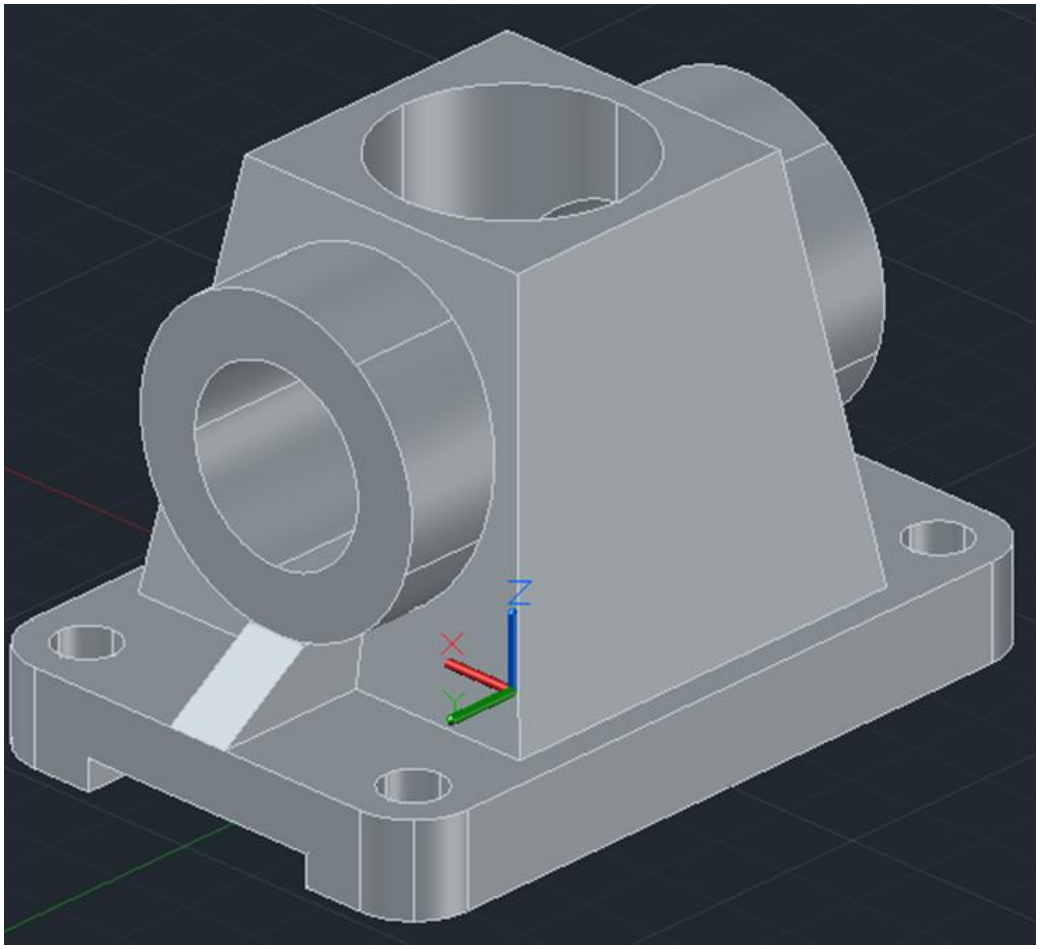


Рисунок 2.23 – Готова тривимірна модель

3 ІНДИВІДУАЛЬНІ ЗАВДАННЯ

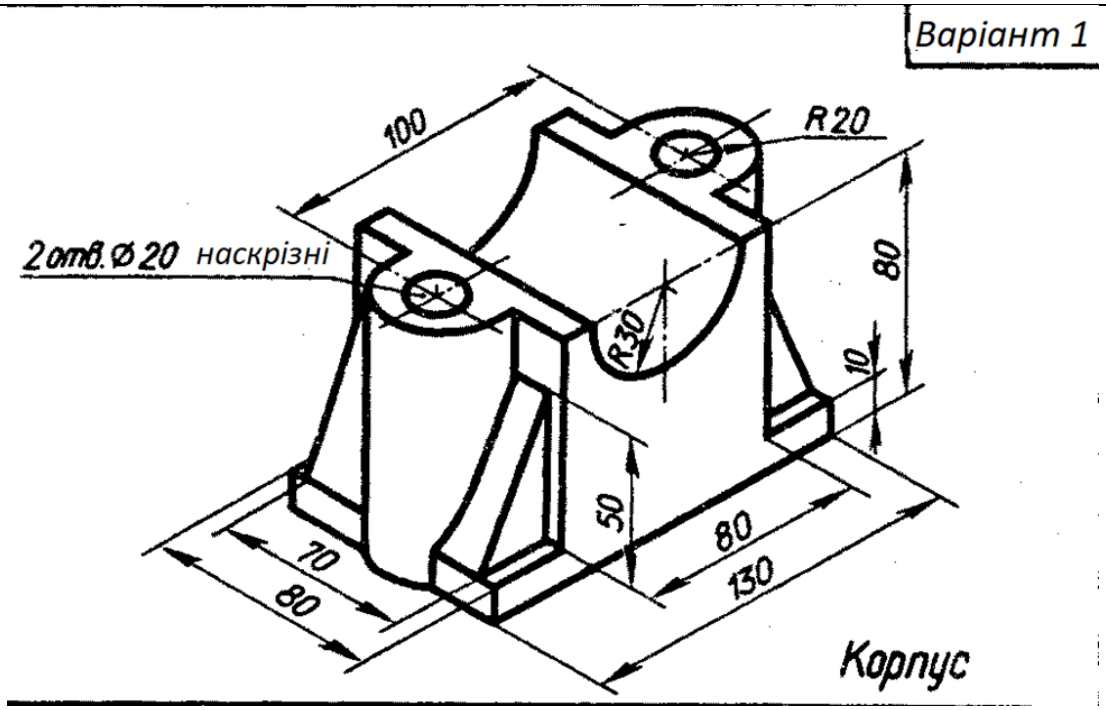
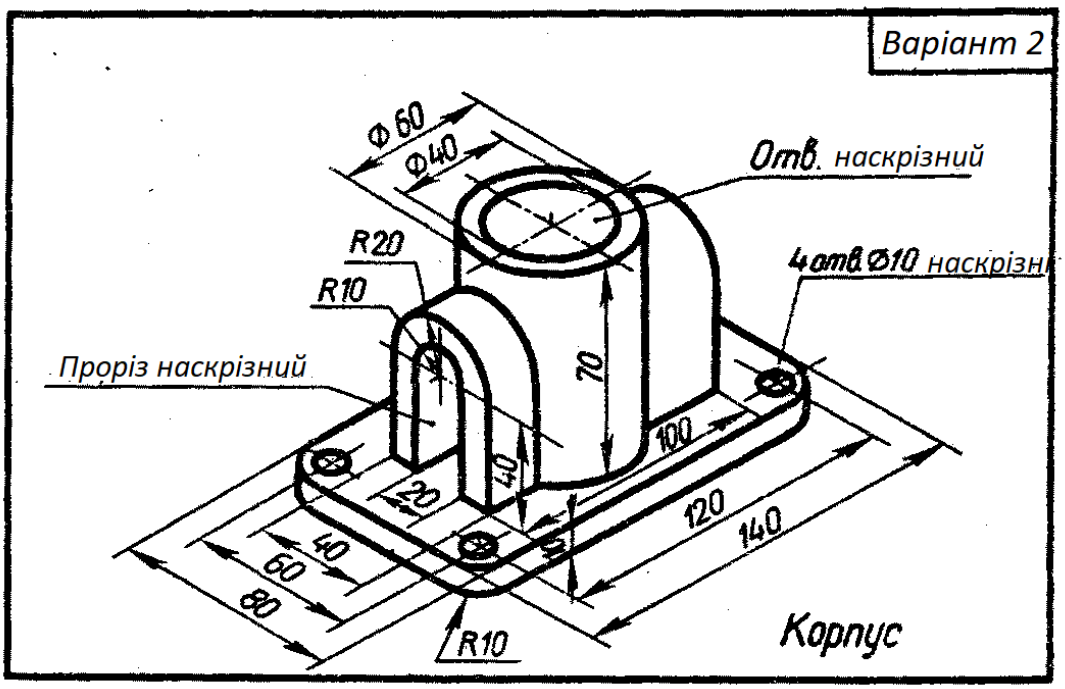
Розробите додаток на C# для візуалізації функції відповідно до одного з варіантів наведених у таблиці 3.1

Таблиця 3.1 — Функції для візуалізації

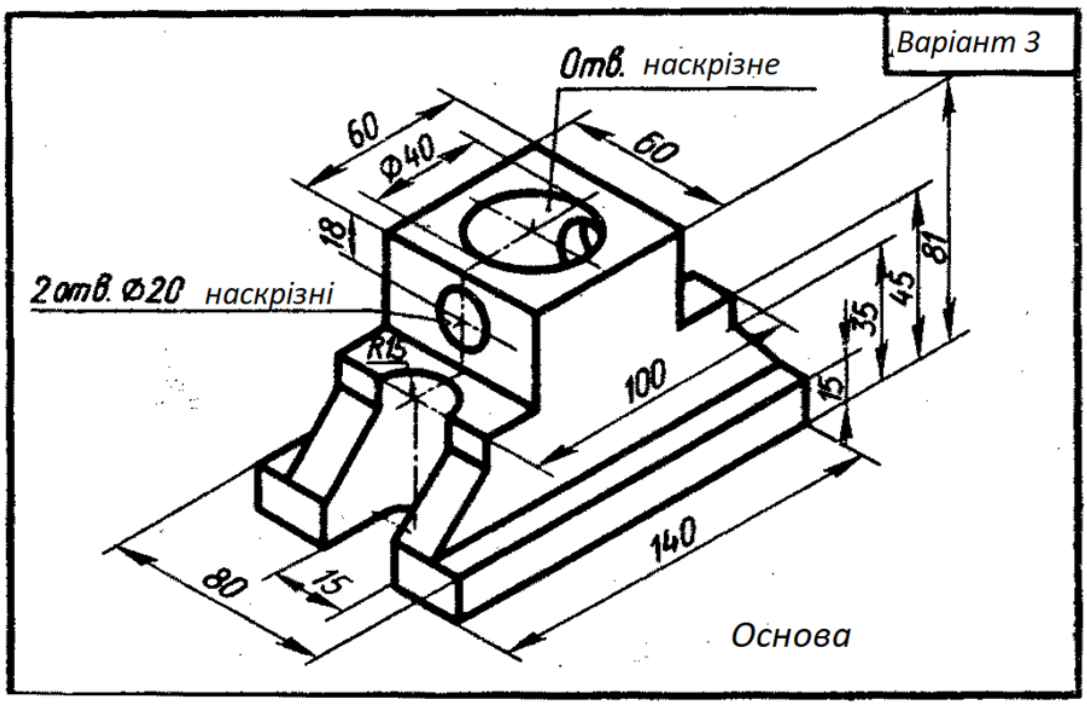
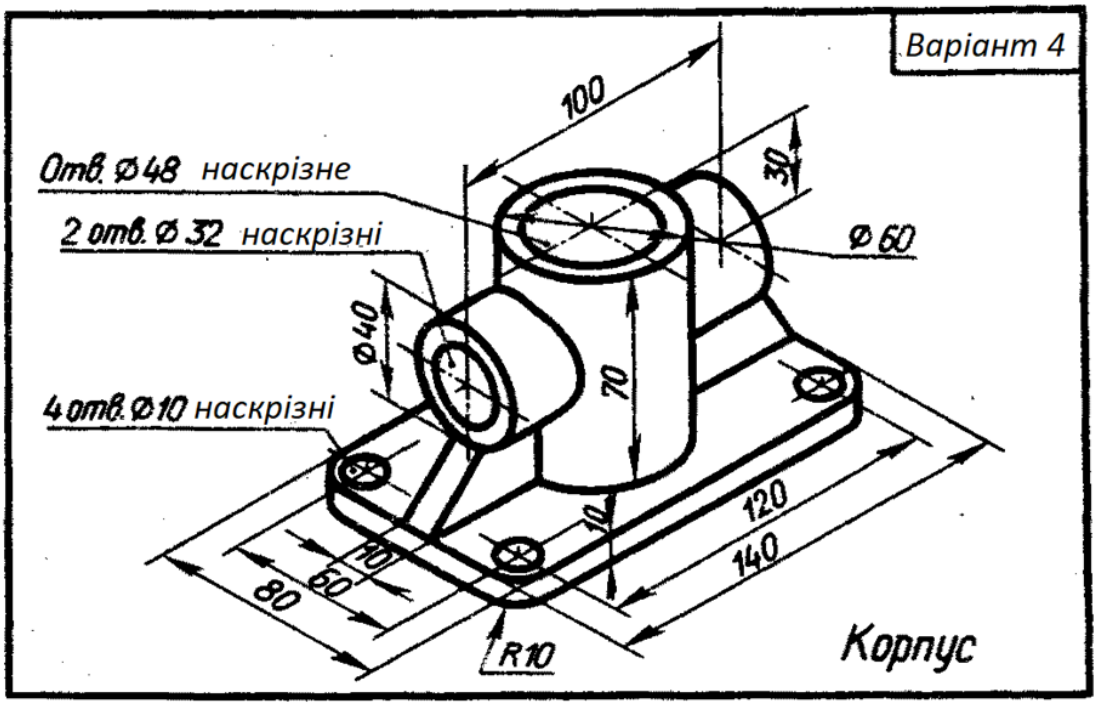
варі-ант	Завдання
1	y = tg(x) на інтервалі [-π +π] (тангенс)
2	y = 2x2-3x-8 на інтервалі [-2; +3] (парабола)
3	y = 3x2-x3 на інтервалі [-2; +4] (кубічна парабола)
4	y = x2(x-2)2 на інтервалі [-1; +3] (багаточлен 4 ступеня)
5	r = 5 sin(17α) на інтервалі [0; 2π]
6	y = 1/x на інтервалі [-4; +4] (гіперболу)
7	y = ex на інтервалі [-1; +3] (експонента)
8	y = sin(x)/x на інтервалі [-10; 10] (перша чудова межа)
9	r = k * φ на інтервалі φ [0; 10π] (архімедова спіраль, полярна система координат);
10	r = a(1-cos(φ)) на інтервалі φ [0; 2π] (кардіоїда, полярна система координат)
11	y = tg(x) на інтервалі [-π +π] (тангенс)
12	y = 2x2-3x-8 на інтервалі [-2; +3] (парабола)
13	y = 3x2-x3 на інтервалі [-2; +4] (кубічна парабола)
14	y = x2(x-2)2 на інтервалі [-1; +3] (багаточлен 4 ступеня)
15	r = 5 sin(17α) на інтервалі [0; 2π]
16	y = 1/x на інтервалі [-4; +4] (гіперболу)
17	y = ex на інтервалі [-1; +3] (експонента)
18	y = sin(x)/x на інтервалі [-10; 10] (перша чудова межа)
19	r = k * φ на інтервалі φ [0; 10π] (архімедова спіраль, полярна система координат);
20	r = a(1-cos(φ)) на інтервалі φ [0; 2π] (кардіоїда)

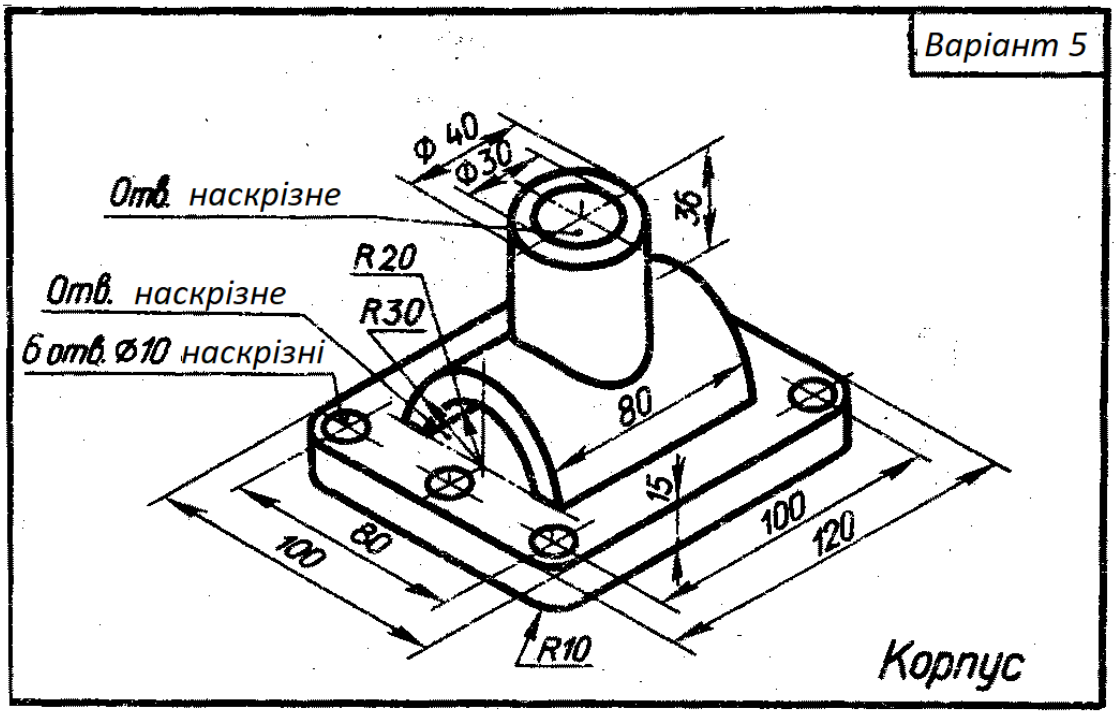
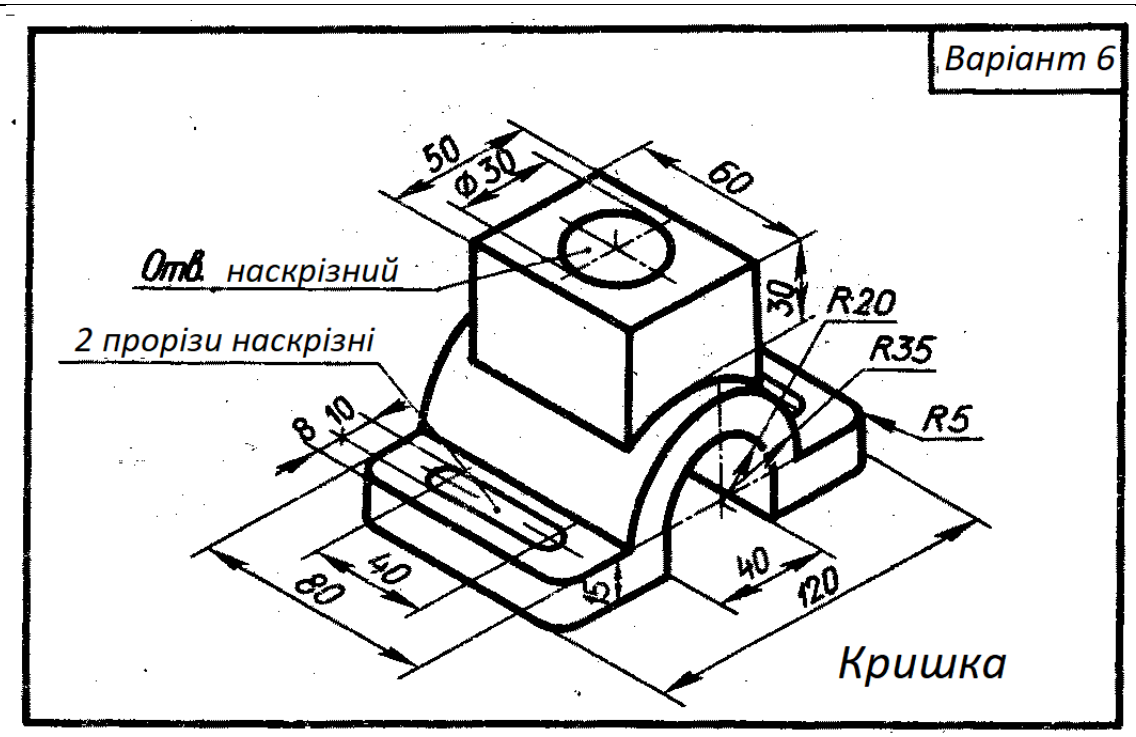
Побудувати тривимірні моделі деталей, аксонометричні види яких наведені у таблиці 3.2.

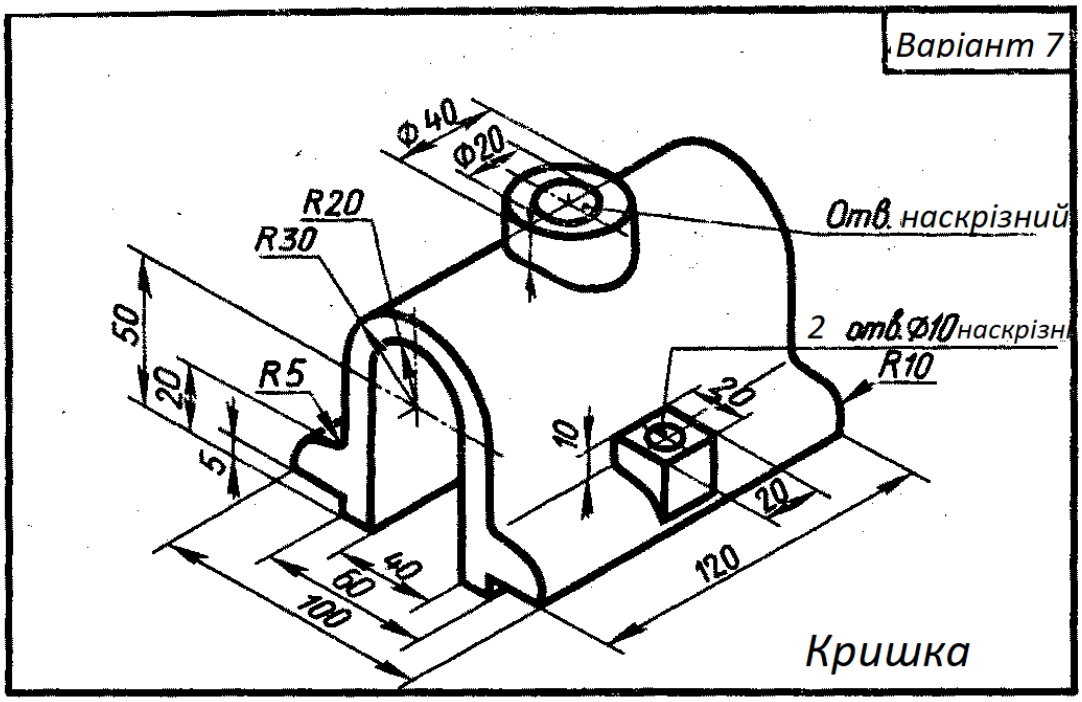
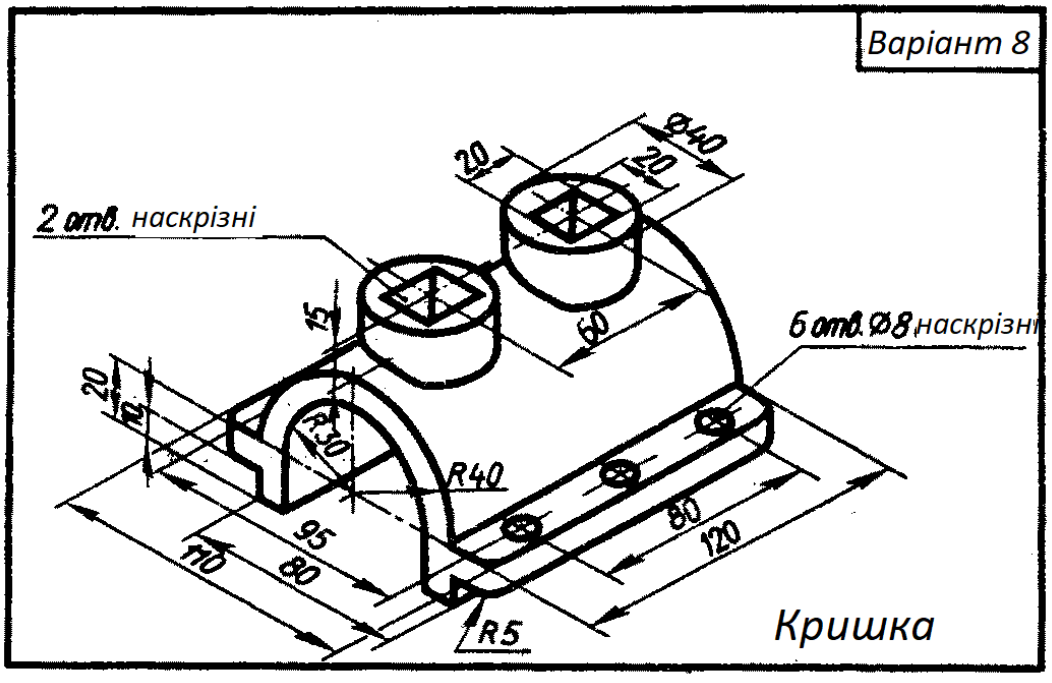
Таблиця 3.2 — Тривимірні моделі деталей

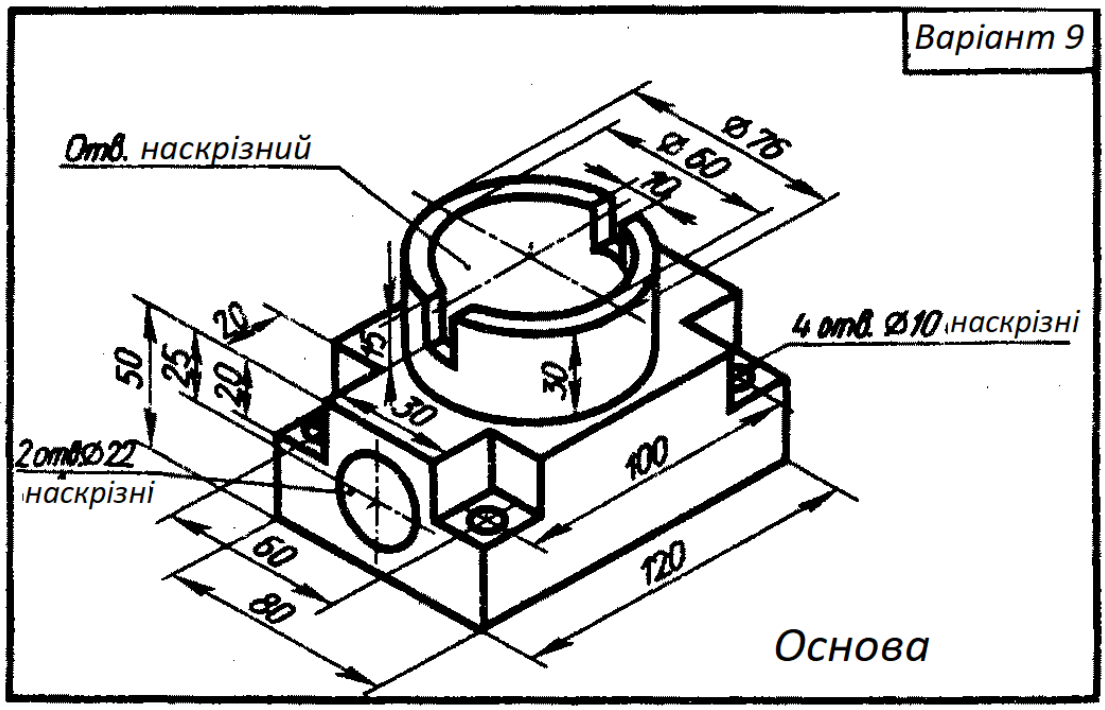
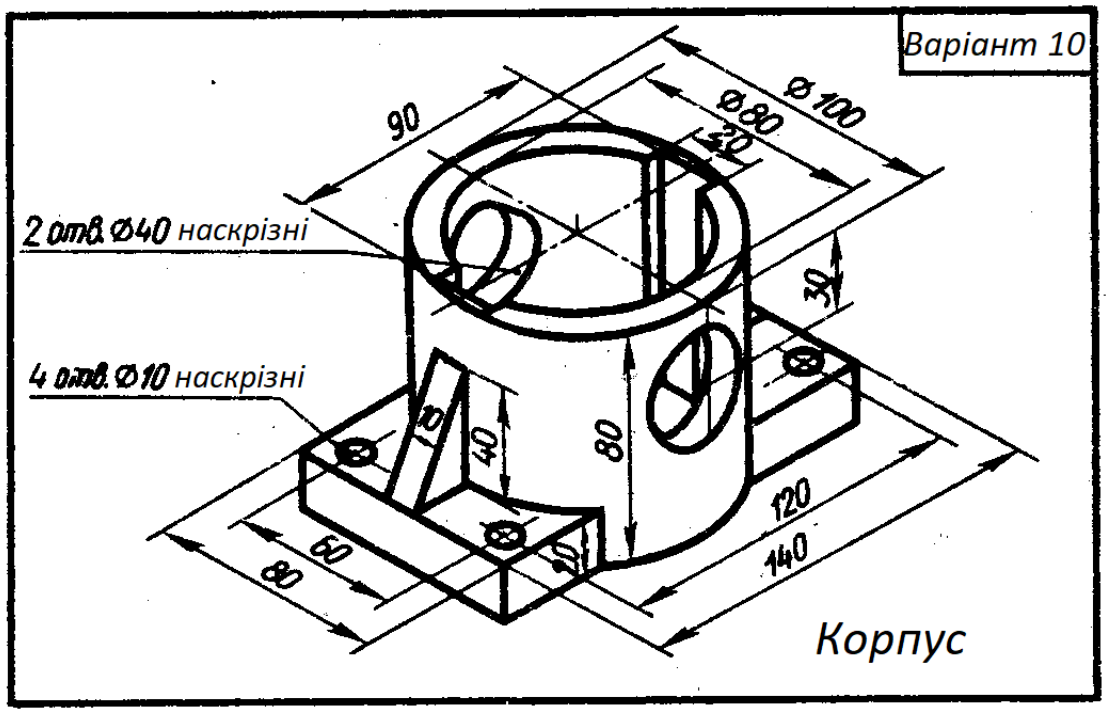
варі-ант	Завдання
1	
2	

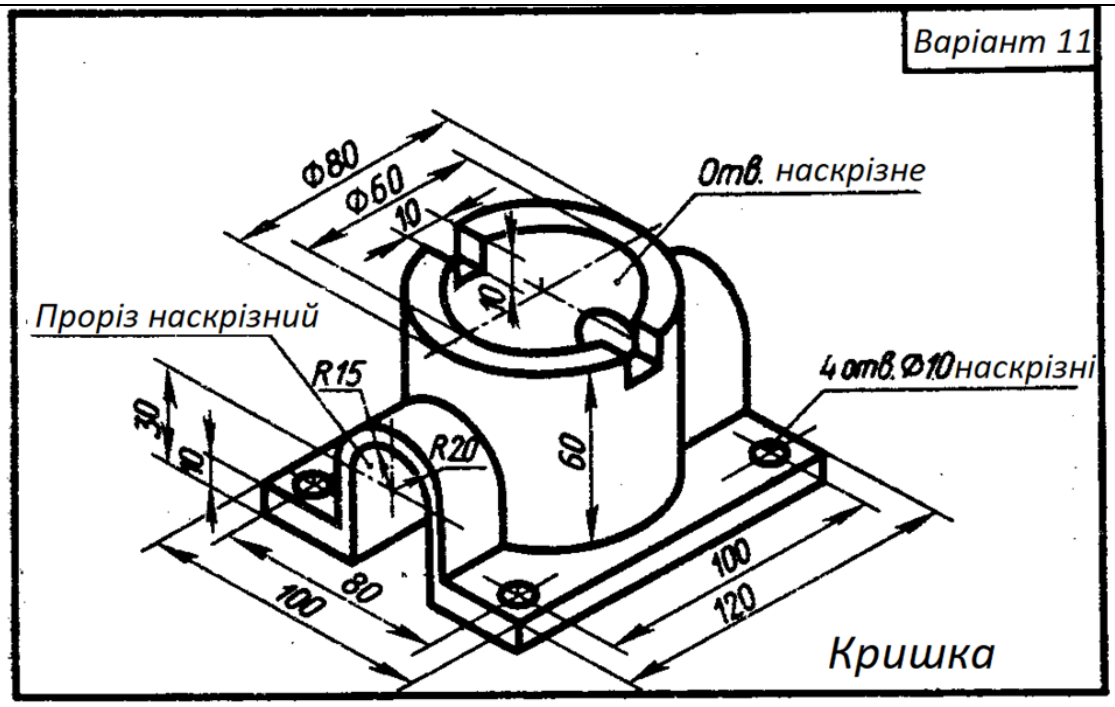
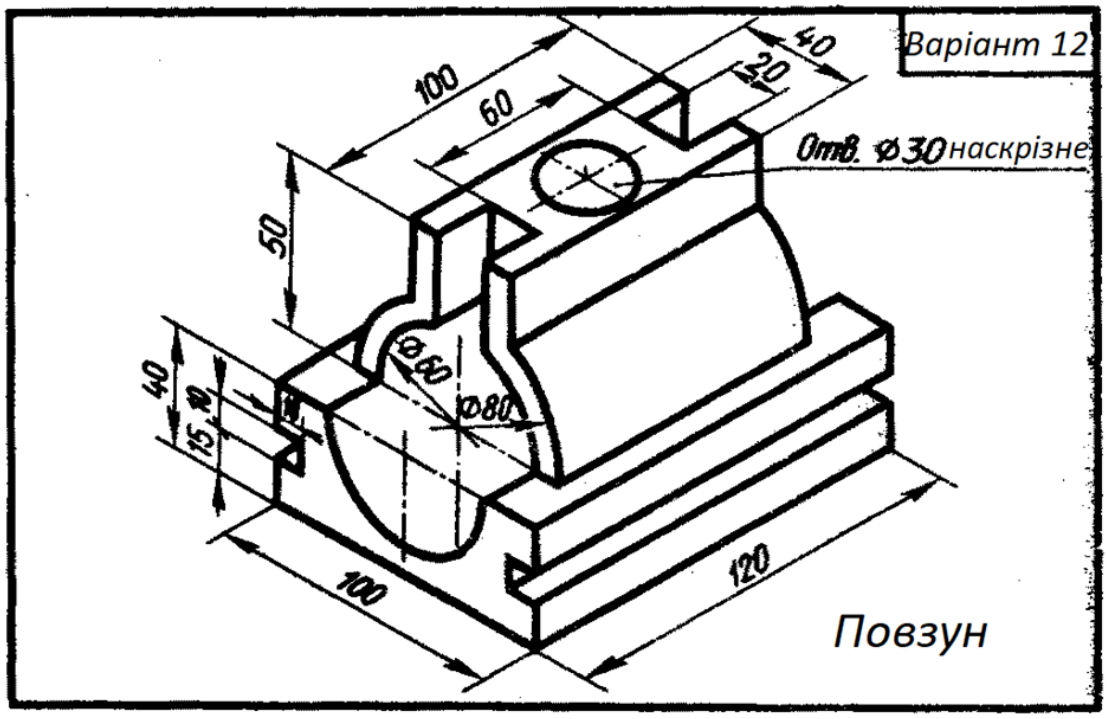
Продовження таблиці 3.2

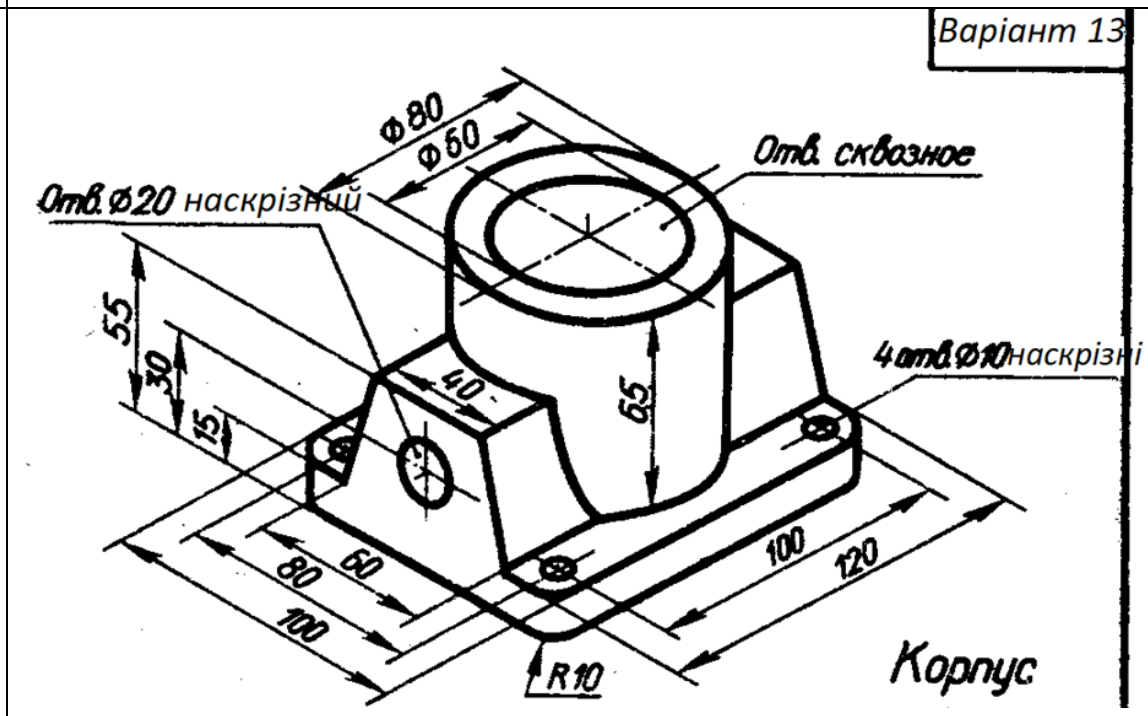
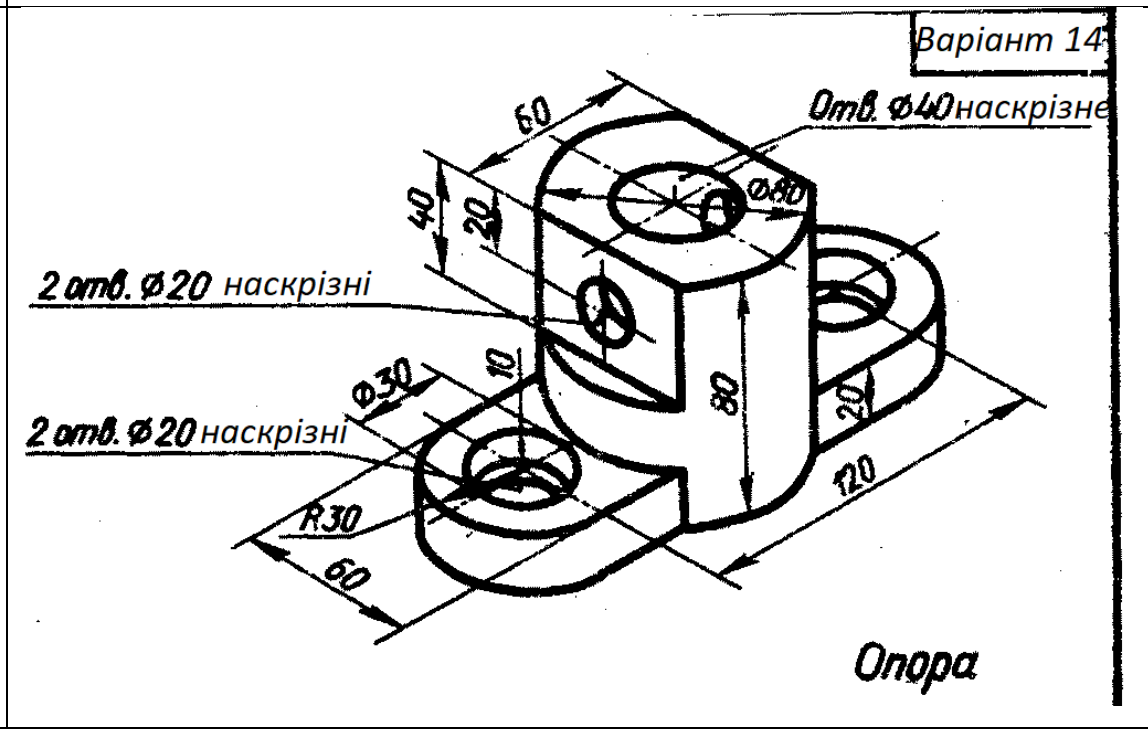
варі-ант	Завдання
3	
4	

варі-ант	Завдання
5	
6	

варі-ант	Завдання
7	
8	

варі-ант	Завдання
9	
10	

варі-ант	Завдання
11	
12	

варі-ант	Завдання
13	
14	

ЛІТЕРАТУРА

1. Введення в комп’ютерну графіку та дизайн : навчальний посібник для студентів спеціальності 174 «Автоматизація, комп’ютерно-інтегровані технології та робототехніка» / укладачі: О. В. Тотосько, П. Д. Стухляк, А. Г. Микитишин, В. В. Левицький, Р. З. Золотий. Тернопіль : ФОП Паляниця В. А., 2023. 304 с.
2. Шабала Є. Є. Комп’ютерна графіка та моделювання : конспект лекцій. Київ : КНУБА, 2022. 108 с.
3. Ворощук В. Я., Вітенько Т. М. Solidworks у завданнях 3D моделювання та інжинірингу технічних систем : навч. посібник. Тернопіль : ФОП Паляниця В. А., 2021. 164 с.
4. Клятченко Я. М., Тарасенко-Клятченко О. В. Комп’ютерна графіка: конспект лекцій : навч. посіб. для здобувачів ступеня бакалавра за спец. 123 Комп’ютерна інженерія. Київ : КПІ ім. Ігоря Сікорського, 2024. 128 с.
5. Комп’ютерна графіка : конспект лекцій для студентів усіх форм навчання спеціальностей 122 «Комп’ютерні науки» та 123 «Комп’ютерна інженерія» з курсу «Комп’ютерна графіка» / уклад.: О. П. Скиба. Тернопіль : Тернопільський національний технічний університет імені Івана Пулюя, 2019. 88 с.
6. Журавчак Л. М., Левченко О. М. Програмування комп’ютерної графіки та мультимедійні засоби : навч. посібник. Львів : Видавництво Львівської політехніки, 2019. 276 с.
7. Мельник О. С. Комп’ютерна анімація та 3D-моделювання : навчальний посібник. Умань : УДПУ імені Павла Тичини, 2018. 141 с.
8. Ledin J. Modern Color Science and Digital Imaging. 1st ed. Taylor and Francis, 2024. 303 p. URL: https://read.kortext.com/library/books/3170083
9. Kothari D. P., Awari G., Shrimankar D., Bhende A. Mathematics for Computer Graphics and Game Programming. Mercury Learning and Information, 2024. 412 p. . URL:

https://read.kortext.com/library/books/3032852

9. Verma C. S., Purohit R., Koyel D. G., Verma H. Computer Graphics and CAD. First Edition. Taylor and Francis, 2024. 306 p. URL: https://read.kortext.com/library/books/2580077

10. Гурковська С. С., Міхєєнко Д. Ю. Застосування програмного забезпечення AutoCAD у сучасній інженерній практиці. *Сучасні інформаційні технології, засоби автоматизації та електропривод* : матеріали VIII Всеукраїнської науково-практичної конференції, 18–20 квітня 2024 р. Краматорськ – Тернопіль : ДДМА, 2024. С. 234-236.

Навчально-методичне видання

Міхєєнко Денис Юрійович

КОМП’ЮТЕРНА ГРАФІКА ТА 3D-МОДЕЛЮВАННЯ:

методичні рекомендації до виконання індивідуального завдання

Самостійне електронне мережеве видання

Публікується в авторській редакції