НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ

імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ Кафедра інформатики та програмної інженерії

ДИСЦИПЛІНА

«КОМП’ЮТЕРНА ГРАФІКА ТА ОБРОБКА ЗОБРАЖЕНЬ»

ЗВІТ

З лабораторної роботи №2

Тема: «Створення мінімальної 3D-програми в Delphi з використанням графічної бібліотеки OpenGL»

Роботу виконала студентка

Групи ІС – 34

Ященко Олександра Олегівна

Викладач: к.т.н., с.н.с.

Щебланін Юрій Миколайович

Київ 2025

## ВСТУП

Тривимірна графіка є невід'ємною частиною сучасних комп'ютерних систем. Від ігрової індустрії до наукових візуалізацій, від архітектурного проектування до медичної діагностики - скрізь використовуються технології 3D-графіки.

OpenGL (Open Graphics Library) - це кросплатформний програмний інтерфейс для роботи з двовимірною та тривимірною графікою. Бібліотека надає розробникам набір функцій для відображення складних тривимірних сцен з простих примітивів.

У цій лабораторній роботі досліджуються базові принципи роботи з OpenGL: побудова геометричних примітивів, налаштування освітлення, різні режими відображення об'єктів та базова анімація через обертання сцени.

ЗМІСТ

1. Завдання 1.1 - Побудова куба
2. Завдання 1.2 - Побудова правильної призми
3. Завдання 1.3 - Побудова правильної піраміди
4. Завдання 1.4 - Режими відображення
5. Аналіз результатів

ОСНОВНА ЧАСТИНА

**Мета роботи:** Створити мінімальну 3D-програму з використанням графічної бібліотеки OpenGL, яка відображає базові тривимірні об'єкти (куб, призму, піраміду) з можливістю перемикання між різними режимами візуалізації.

Структура програми

Програма складається з трьох основних компонентів:

1. **OpenGLWidget** - віджет для відображення 3D-графіки
2. **MainWindow** - головне вікно з інтерфейсом управління
3. **Геометричні об'єкти** - методи побудови куба, призми та піраміди

Ініціалізація OpenGL

При створенні віджета виконується налаштування базових параметрів OpenGL:

def initializeGL(self):

glEnable(GL\_LIGHTING)

glEnable(GL\_LIGHT0)

glEnable(GL\_DEPTH\_TEST)

glEnable(GL\_COLOR\_MATERIAL)

glClearColor(0.1, 0.0, 0.2, 0.0)

Ці налаштування забезпечують коректне відображення об'ємних об'єктів з освітленням та правильну обробку глибини сцени.

Налаштування проекції

def resizeGL(self, w, h):

glViewport(0, 0, w, h)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

aspect = w / h if h != 0 else 1

w\_scale = 1.5 \* aspect

glOrtho(-w\_scale, w\_scale, -1.5, 1.5, 1, 10)

glMatrixMode(GL\_MODELVIEW)

glLoadIdentity()

Використовується ортогональна проекція, яка зберігає паралельність ліній та відстані між об'єктами.

ЗАВДАННЯ 1.1 - ПОБУДОВА КУБА

Куб - це правильний шестигранник, у якого всі грані є квадратами. Для побудови куба необхідно визначити координати восьми вершин та шість граней.

Математична модель

Координати вершин куба з ребром 2h (де h=1.0):

* Передні вершини: (±h, ±h, h)
* Задні вершини: (±h, ±h, -h)

Реалізація

Кожна грань куба визначається чотирма вершинами. Вершини обходяться проти годинникової стрілки для правильного визначення лицьової сторони:

def drawCube(self):

h = 1.0

glColor3f(1.0, 0.0, 0.0)

glBegin(GL\_QUADS)

glNormal3f(0.0, 0.0, 1.0)

glVertex3f(h, h, h)

glVertex3f(-h, h, h)

glVertex3f(-h, -h, h)

glVertex3f(h, -h, h)

glEnd()

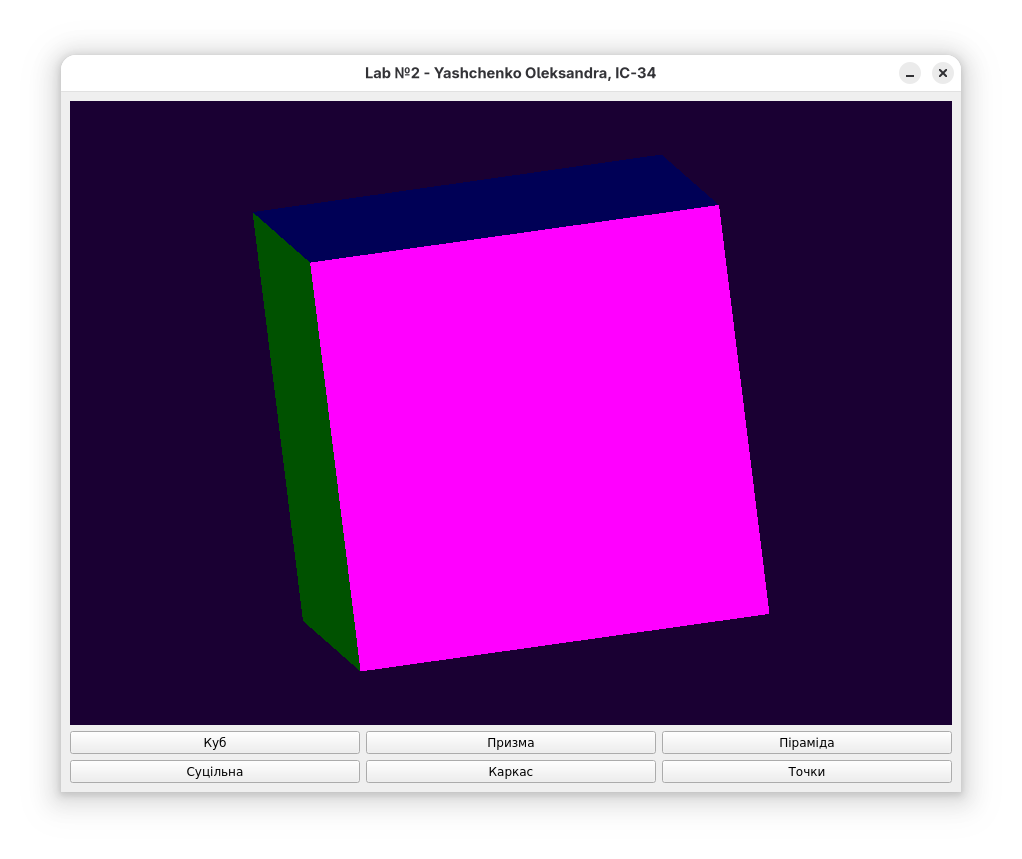
# Аналогічно для інших 5 граней...

Вектори нормалей

Для кожної грані задається вектор нормалі, який вказує назовні від центру куба:

* Передня грань: (0, 0, 1)
* Задня грань: (0, 0, -1)
* Права грань: (1, 0, 0)
* Ліва грань: (-1, 0, 0)
* Верхня грань: (0, 1, 0)
* Нижня грань: (0, -1, 0)

Вектори нормалей необхідні для правильного розрахунку освітлення поверхонь.

****

ЗАВДАННЯ 1.2 - ПОБУДОВА ПРАВИЛЬНОЇ ПРИЗМИ

Правильна n-кутна призма складається з двох паралельних правильних n-кутників (основ) та n прямокутних бічних граней.

Математична модель

Для побудови 16-гранної призми використовуються параметричні рівняння кола:

* x = r × cos(φ)
* y = r × sin(φ)
* φ = i × (2π/n), де i = 0..n-1

Параметри призми:

* n = 16 (кількість граней)
* r = 0.5 (радіус основи)
* h = 1.0 (висота)

Реалізація

def drawPrism(self):

n = 16

h = 1.0

r = 0.5

delta\_fi = 2 \* math.pi / n

glColor3f(0.0, 1.0, 0.0)

glBegin(GL\_QUADS)

for i in range(n):

fi = i \* delta\_fi

nx = math.cos(fi + delta\_fi / 2)

ny = math.sin(fi + delta\_fi / 2)

glNormal3f(nx, ny, 0.0)

glVertex3f(r \* math.cos(fi), r \* math.sin(fi), h / 2)

glVertex3f(r \* math.cos(fi), r \* math.sin(fi), -h / 2)

glVertex3f(r \* math.cos(fi + delta\_fi), r \* math.sin(fi + delta\_fi), -h / 2)

glVertex3f(r \* math.cos(fi + delta\_fi), r \* math.sin(fi + delta\_fi), h / 2)

glEnd()

glBegin(GL\_POLYGON)

glColor3f(1.0, 1.0, 0.0)

glNormal3f(0, 0, 1)

for i in range(n):

fi = i \* delta\_fi

glVertex3f(r \* math.cos(fi), r \* math.sin(fi), h / 2)

glEnd()

glBegin(GL\_POLYGON)

glColor3f(1.0, 0.5, 0.0)

glNormal3f(0, 0, -1)

for i in range(n - 1, -1, -1):

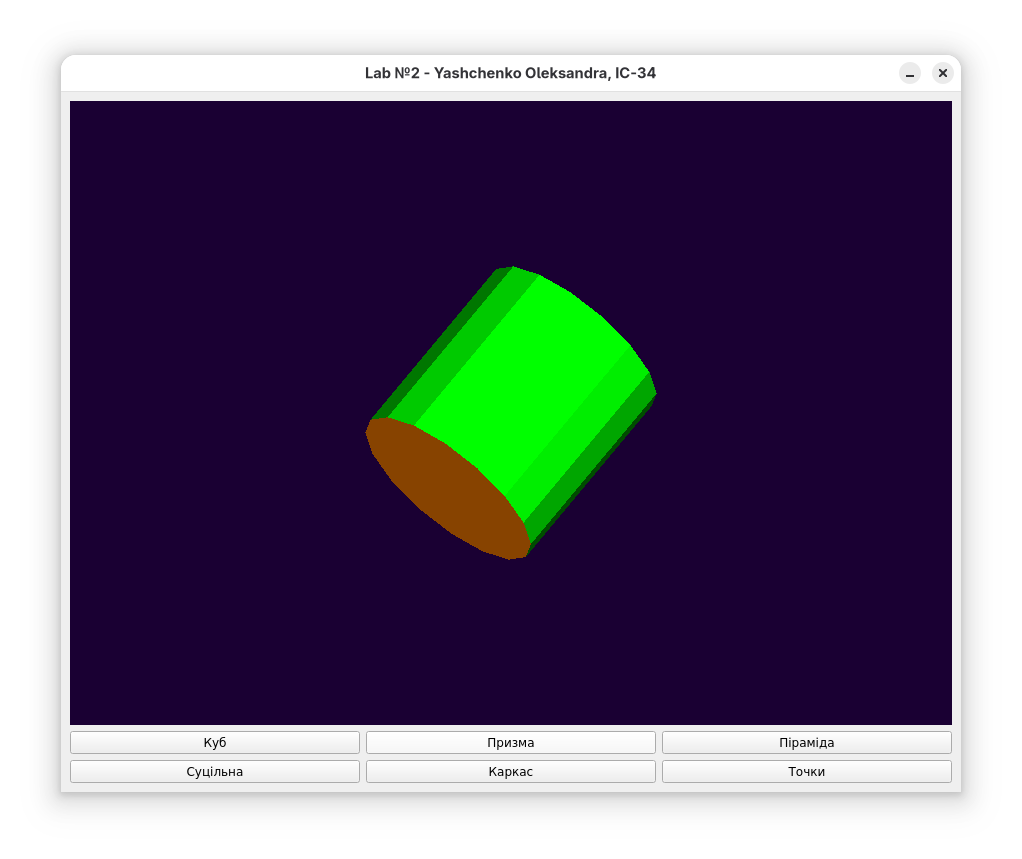
fi = i \* delta\_fi

glVertex3f(r \* math.cos(fi), r \* math.sin(fi), -h / 2)

glEnd()

Особливості реалізації

1. **Нормалі бічних граней** розраховуються як напрям від центру до середини грані
2. **Порядок обходу вершин** - проти годинникової стрілки для лицьових граней
3. **GL\_POLYGON** використовується для малювання багатокутників з довільною кількістю вершин

****

ЗАВДАННЯ 1.3 - ПОБУДОВА ПРАВИЛЬНОЇ ПІРАМІДИ

Правильна n-кутна піраміда складається з правильного n-кутника в основі та n трикутних бічних граней, що сходяться у вершині.

Математична модель

Параметри піраміди:

* n = 16 (кількість бічних граней)
* r = 0.5 (радіус описаного кола основи)
* h = 1.0 (висота піраміди)

Вершина піраміди: (0, 0, h)

Вершини основи: (r×cos(φ), r×sin(φ), 0), де φ = i×(2π/n)

Розрахунок нормалей

Для бічних граней нормаль розраховується з урахуванням кута нахилу:

θ = arctg(h/r)

Компоненти нормалі:

* nx = cos(φ + Δφ/2) × sin(θ)
* ny = sin(φ + Δφ/2) × sin(θ)
* nz = cos(θ)

Реалізація

def drawPyramid(self):

n = 16

h = 1.0

r = 0.5

delta\_fi = 2 \* math.pi / n

teta = math.atan(h / r)

glBegin(GL\_TRIANGLES)

for i in range(n):

fi = i \* delta\_fi

glColor3f((i % 2), ((i % 3) / 2), ((i % 5) / 4))

nx = math.cos(fi + delta\_fi / 2) \* math.sin(teta)

ny = math.sin(fi + delta\_fi / 2) \* math.sin(teta)

nz = math.cos(teta)

glNormal3f(nx, ny, nz)

glVertex3f(0, 0, h)

glVertex3f(r \* math.cos(fi), r \* math.sin(fi), 0)

glVertex3f(r \* math.cos(fi + delta\_fi), r \* math.sin(fi + delta\_fi), 0)

glEnd()

glBegin(GL\_POLYGON)

glColor3f(0.8, 0.8, 0.0)

glNormal3f(0, 0, -1)

for i in range(n - 1, -1, -1):

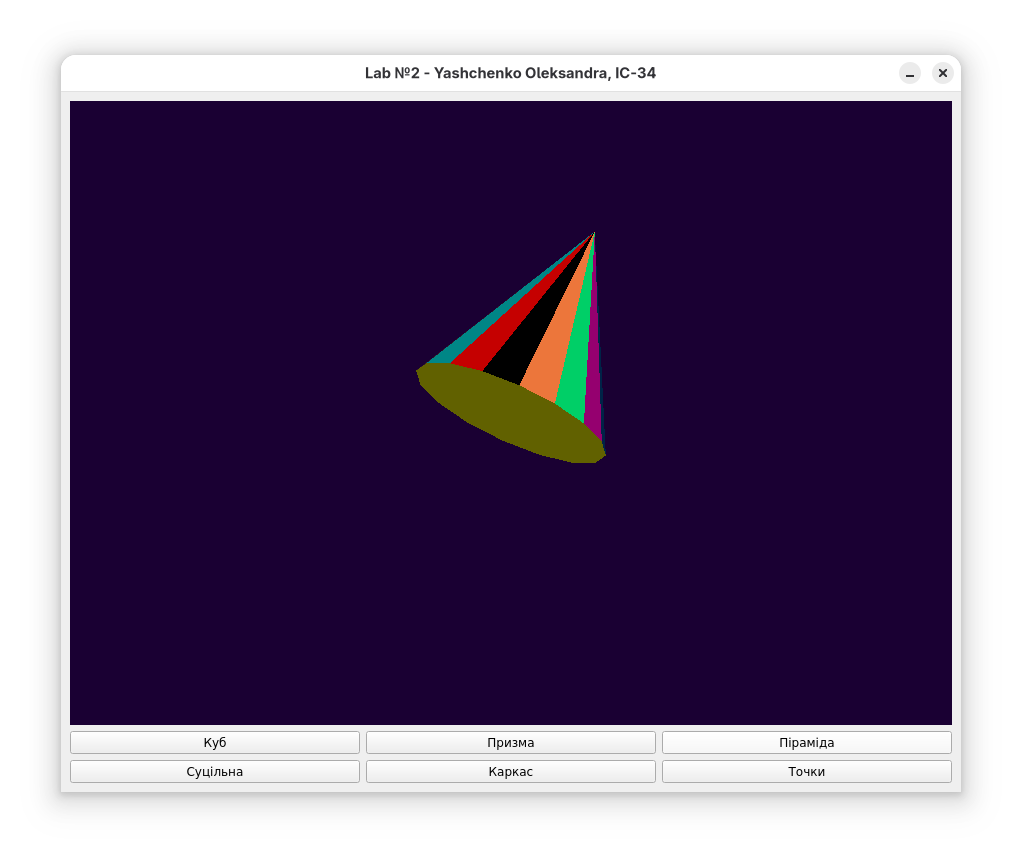
fi = i \* delta\_fi

glVertex3f(r \* math.cos(fi), r \* math.sin(fi), 0)

glEnd()

Особливості

1. **Різнокольорові грані** - для кращої візуалізації кожна грань має свій колір
2. **GL\_TRIANGLES** - примітив для малювання трикутників
3. **Складніший розрахунок нормалей** через нахилені поверхні

****

ЗАВДАННЯ 1.4 - РЕЖИМИ ВІДОБРАЖЕННЯ

OpenGL підтримує три основні режими відображення полігонів:

1. Суцільний режим (GL\_FILL)

Стандартний режим, де всі грані заповнюються кольором з урахуванням освітлення:

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL)

2. Каркасний режим (GL\_LINE)

Відображаються тільки ребра полігонів:

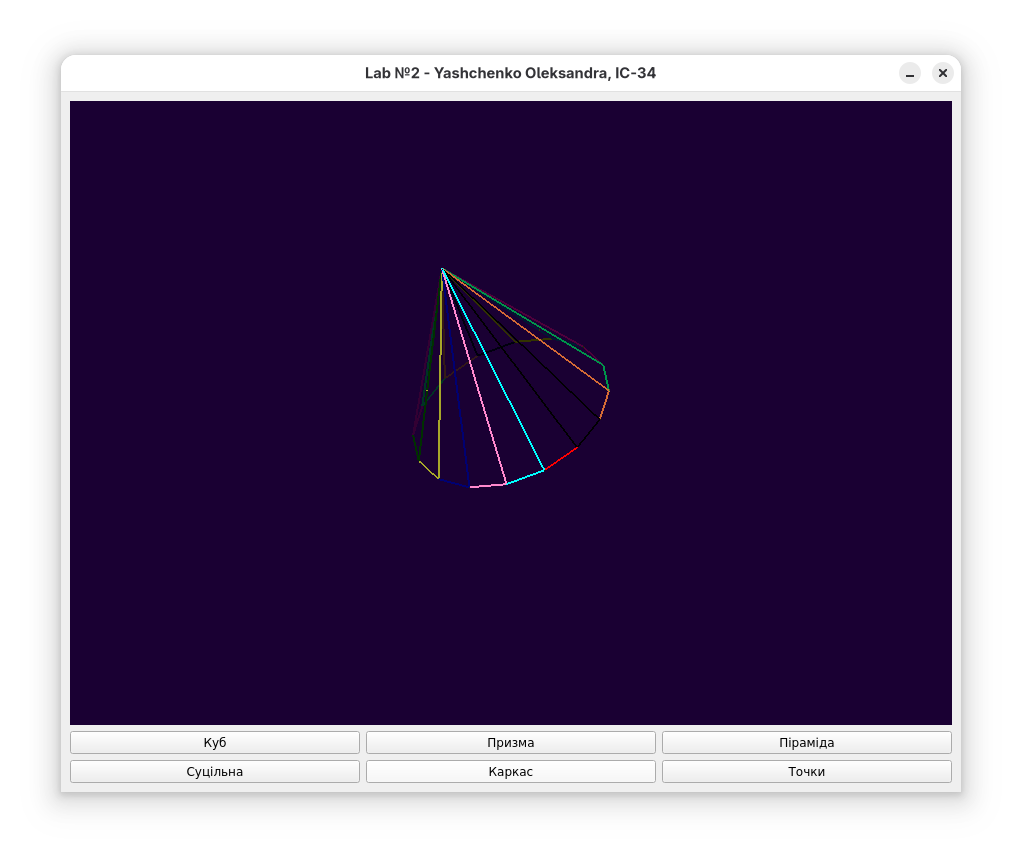
glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)

glLineWidth(2.0)

glEnable(GL\_LINE\_SMOOTH)

Цей режим корисний для:

* Аналізу геометрії об'єкта
* Налагодження моделей
* Перегляду топології сітки

****

3. Точковий режим (GL\_POINT)

Відображаються тільки вершини полігонів:

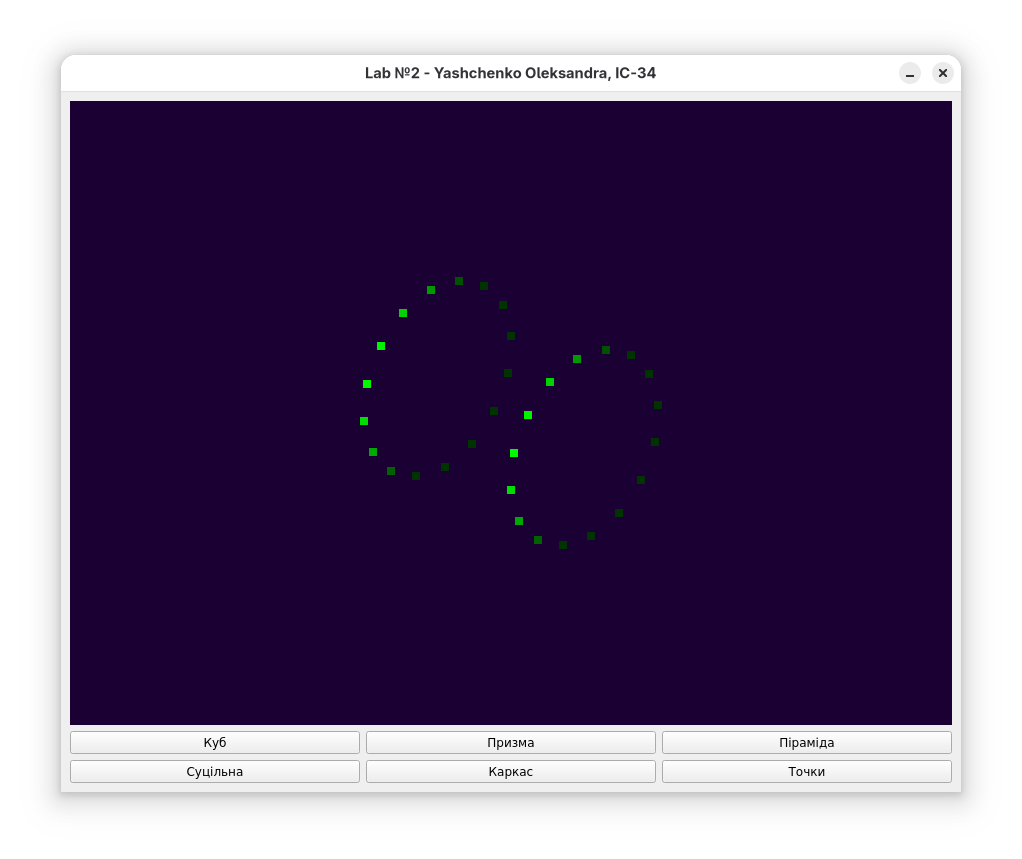
glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT)

glPointSize(8.0)

glEnable(GL\_POINT\_SMOOTH)

Використовується для:

* Візуалізації вершин моделі
* Аналізу щільності сітки
* Debugging геометрії

****

Перемикання режимів

def setRenderMode(self, mode):

self.render\_mode = mode

self.update()

У методі paintGL() перед малюванням об'єкта встановлюється потрібний режим:

if self.render\_mode == 'point':

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_POINT)

glPointSize(8.0)

glEnable(GL\_POINT\_SMOOTH)

elif self.render\_mode == 'line':

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_LINE)

glLineWidth(2.0)

glEnable(GL\_LINE\_SMOOTH)

else:

glPolygonMode(GL\_FRONT\_AND\_BACK, GL\_FILL)

АНІМАЦІЯ ТА ІНТЕРАКТИВНІСТЬ

Обертання сцени

Для створення анімації використовується таймер, який оновлюється кожні 20 мс:

self.timer = QTimer(self)

self.timer.timeout.connect(self.updateRotation)

self.timer.start(20)

Метод оновлення кута:

def updateRotation(self):

self.angle += 1.0

if self.angle >= 360.0:

self.angle = 0.0

self.update()

У методі малювання застосовується обертання:

glRotatef(self.angle, 1.0, 1.0, 1.0)

Об'єкт обертається навколо вектора (1, 1, 1), що створює цікавий ефект обертання в просторі.

Інтерфейс користувача

Програма містить два ряди кнопок:

**Вибір фігури:**

* Куб
* Призма
* Піраміда

**Режим відображення:**

* Суцільна
* Каркас
* Точки

cube\_btn = QPushButton("Куб")

cube\_btn.clicked.connect(lambda: self.opengl\_widget.setShape('cube'))

shape\_layout.addWidget(cube\_btn)

АНАЛІЗ РЕЗУЛЬТАТІВ

Порівняння об'єктів

| **Параметр** | **Куб** | **Призма** | **Піраміда** |
| --- | --- | --- | --- |
| Кількість вершин | 8 | 32 | 17 |
| Кількість граней | 6 | 18 | 17 |
| Складність нормалей | Проста | Середня | Складна |
| Тип граней | Квадрати | Прямокутники + n-кутники | Трикутники + n-кутник |

Продуктивність

При кількості граней 16 для призми та піраміди програма працює плавно (50 FPS). Обертання відбувається без затримок завдяки:

* Використанню подвійної буферизації
* Апаратному прискоренню OpenGL
* Оптимізованому коду малювання

Переваги реалізації

1. **Мінімалістичний код** - програма містить тільки необхідний функціонал
2. **Зрозуміла структура** - чітке розділення на компоненти
3. **Розширюваність** - легко додати нові фігури або режими
4. **Кросплатформеність** - працює на Windows, Linux, macOS

Можливі покращення

1. Додати можливість зміни кількості граней (n) через інтерфейс
2. Реалізувати керування камерою мишею
3. Додати текстурування об'єктів
4. Впровадити збереження зображень у файл
5. Додати більше варіантів освітлення

ВИСНОВКИ

У ході виконання лабораторної роботи було успішно створено програму для візуалізації тривимірних геометричних об'єктів з використанням бібліотеки OpenGL та фреймворку PyQt5.

**Основні досягнення:**

1. **Освоєно базові концепції OpenGL:** ініціалізація контексту, налаштування проекції, управління матрицями трансформації, використання буферів глибини та подвійної буферизації.
2. **Реалізовано три типи геометричних примітивів:** куб (на основі квадратів), 16-гранна призма (комбінація прямокутників та багатокутників) та 16-гранна піраміда (на основі трикутників).
3. **Застосовано правильний розрахунок освітлення:** для кожного типу граней визначено коректні вектори нормалей, що забезпечує реалістичне відображення об'ємних об'єктів.
4. **Впроваджено три режими візуалізації:** суцільний (для фінального перегляду), каркасний (для аналізу структури) та точковий (для перегляду вершин), що дозволяє всебічно досліджувати геометрію об'єктів.
5. **Створено інтуїтивний інтерфейс:** програма має зручну систему перемикання між фігурами та режимами відображення, а постійне обертання об'єктів дозволяє краще оцінити їх тривимірну структуру.

**Практична цінність:**

Розроблена програма демонструє фундаментальні принципи роботи з 3D-графікою, які використовуються в професійних системах автоматизованого проектування (CAD), ігрових движках, системах наукової візуалізації та інших сферах комп'ютерної графіки.

Мінімалістичний підхід до реалізації робить код зрозумілим для навчання, при цьому зберігаючи всі ключові аспекти роботи з OpenGL. Програма може служити базою для більш складних проектів з 3D-візуалізації.

**Набуті навички:**

* Робота з графічною бібліотекою OpenGL
* Математичні розрахунки для 3D-геометрії
* Розуміння систем координат та трансформацій
* Програмування графічних інтерфейсів з PyQt5
* Оптимізація продуктивності графічних застосунків

Виконання лабораторної роботи сформувало міцний фундамент для подальшого вивчення комп'ютерної графіки та створення більш складних тривимірних візуалізацій.

ПЕРЕЛІК ПОСИЛАНЬ

1. OpenGL Programming Guide [Електронний ресурс] // Khronos Group. – 2025. – Режим доступу: <https://www.opengl.org/documentation/>
2. PyOpenGL Documentation [Електронний ресурс] // PyOpenGL Project. – 2025. – Режим доступу: <http://pyopengl.sourceforge.net/documentation/>
3. PyQt5 Reference Guide [Електронний ресурс] // Riverbank Computing. – 2025. – Режим доступу: <https://www.riverbankcomputing.com/static/Docs/PyQt5/>
4. OpenGL Programming/Modern OpenGL Tutorial [Електронний ресурс] // Wikibooks. – 2025. – Режим доступу: <https://en.wikibooks.org/wiki/OpenGL_Programming>
5. Angel, E. Interactive Computer Graphics: A Top-Down Approach with OpenGL / E. Angel. – 7th ed. – Pearson, 2014. – 800 p.
6. Shreiner, D. OpenGL Programming Guide: The Official Guide to Learning OpenGL / D. Shreiner, G. Sellers, J. Kessenich, B. Licea-Kane. – 9th ed. – Addison-Wesley, 2017. – 976 p.
7. Математичні основи комп'ютерної графіки [Електронний ресурс] // Math Insight. – 2025. – Режим доступу: <https://mathinsight.org/>