Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

Лабораторна робота №9
з дисципліни
«Технології Computer Vision»
на тему
«Синтез реалістичних об’єктів - 3D Computer Vision»

Виконала:
Студентка групи ІМ-21
Кривохата Марія Юріївна
Номер у списку групи: 12

Перевірив: Баран Д. Р.

Київ 2024

Мета: дослідити методологію і технології створення доповненої реальності.

Завдання:

	Самостійно обраний процес моделювання

Було обрано таке завдання:

Для формування модельного Dataset з метою навчання нейромережі для розпізнавання заданих об’єктів за технологіями Computer Vision створити динамічну модель конвеєрної лінії ***з виробництва автомобільних колес***.

Результати виконання лабораторної роботи

Використані математичні підходи (моделі):

1. Модель основних компонентів колеса

- *Тороїдна модель шини:* Шина моделюється як тор з урахуванням зовнішнього та внутрішнього радіуса. Для створення профілю використовуються:
    - *Сегментація окружності:* Визначення кількості сегментів (segments) і бокових граней (sides) для створення плавного кола.
    - *Профіль із плоскою поверхнею:* Використовується параметр outer_flat_ratio для створення плоскої центральної частини шини і закруглених країв (плечей).
    - *Тригонометричні функції:* Обчислення координат і нормалей вершин для плавного переходу між сегментами.
- *Гвинтова текстура протектора:*
    - Текстура моделюється як серія вузьких канавок на поверхні шини.
    - Обчислення глибини та ширини кожного сегмента протектора з використанням кутових параметрів.

2. Модель диска (обод та спиці)

- *Обід (rim):*
    - Представлений у вигляді циліндра з внутрішнім і зовнішнім радіусами.

- Використовуються допоміжні функції для побудови товстих циліндрів (_create_cylinder_no_cups).

- *Спиці (spokes):*
  - Використовуються геометричні перетворення для створення кількох спиць:
    - *Обертання матриці* для розташування спиць по колу.
    - *Прямокутний профіль спиць:* Створюються коробчасті елементи (_create_box) зі змінними розмірами.
  - Розрахунок довжини, ширини та глибини спиць для забезпечення правильного співвідношення з іншими елементами.

3. ***Модель «серцевини» та барабана***

- *«Серцевина» (hub):*
  - Моделюється як циліндр із кришками з обох боків.
  - Для додаткового рельєфу додається ковпак, розрахований із використанням збільшеного радіусу.
- *Барабан (barrel):*
  - Циліндрична форма з невеликою увігнутістю для реалістичності.
  - Використовується параметр barrel_width для обмеження ширини і створення обідка на краях.

4. Використання нормалей

- *Обчислення нормалей:*
  - Для забезпечення правильного освітлення розраховуються нормалі для кожної вершини:
    - Для плоских секцій нормалі спрямовані прямо.
    - Для заокруглених країв враховуються кути нормалі з використанням синуса і косинуса.
  - Нормалі нормалізуються для уникнення некоректного освітлення.

5. Модель освітлення та матеріалів

- *Освітлення:*
    - Використовуються джерела світла (GL_LIGHT0), щоб імітувати природне освітлення.
    - Настроюються компоненти світла: AMBIENT, DIFFUSE, SPECULAR.
- *Матеріали:*
    - Колеса складаються з різних матеріалів (шина, обід, серцевина, спиці), кожен із яких має свої властивості:
        - Колір.
        - Блиск.

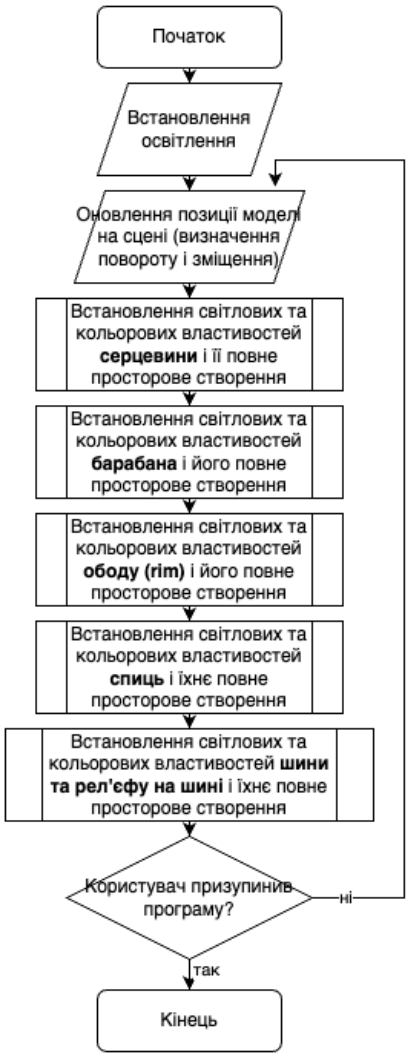
6. Побудова геометричних примітивів

- *Тригонометричні функції:* Використовуються для побудови вершин циліндрів, тороїдів і прямокутників.
- *Підхід "Quad Strip":*
    - Для створення гладких поверхонь (шина, обід) використовується метод побудови смуг із чотирикутників.
- *З'єднання сегментів:*
    - Верхні, нижні та бічні кришки додаються для завершення моделі циліндрів і ободів.

7. Рух і анімація

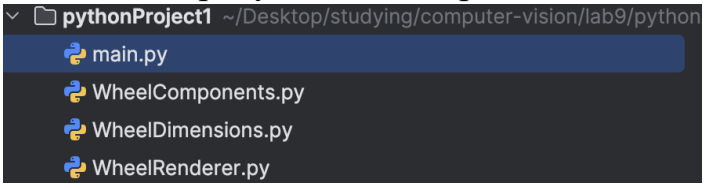
- *Обертання:*
    - Колесо обертається навколо своєї осі з урахуванням зміни кута в часі.
- *Трансляція:*
    - Колесо рухається в просторі з використанням зміщення, щоб імітувати реалістичний рух.

Результати архітектурного проектування:



Опис структури проекту програми:

main.py – головний файл програми
WheelDimensions.py – файл, в якому міститься клас із параметрами розміру для автомобільного колеса
WheelComponents.py – файл, в якому міститься клас, що відмальовує всі компоненти колеса
WheelRender.py – файл, в якому міститься клас, що забезпечує переміщення колеса по екрану та його обертання



Результати роботи програми:

Світло було виставлено за таким принципом:

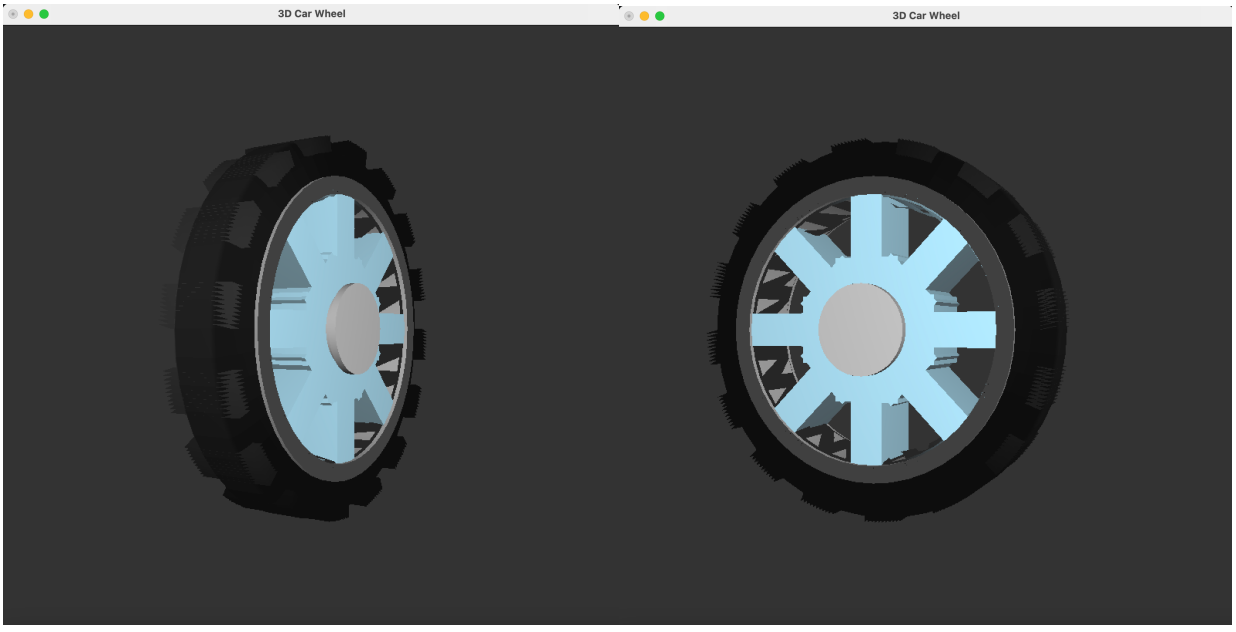
```
# Lighting setup
glLightfv(GL_LIGHT0, GL_POSITION, (10, 10, 10, 1))
glLightfv(GL_LIGHT0, GL_AMBIENT, (0.3, 0.3, 0.3, 1))
glLightfv(GL_LIGHT0, GL_DIFFUSE, (1, 1, 1, 1))
glLightfv(GL_LIGHT0, GL_SPECULAR, (1, 1, 1, 1))
```

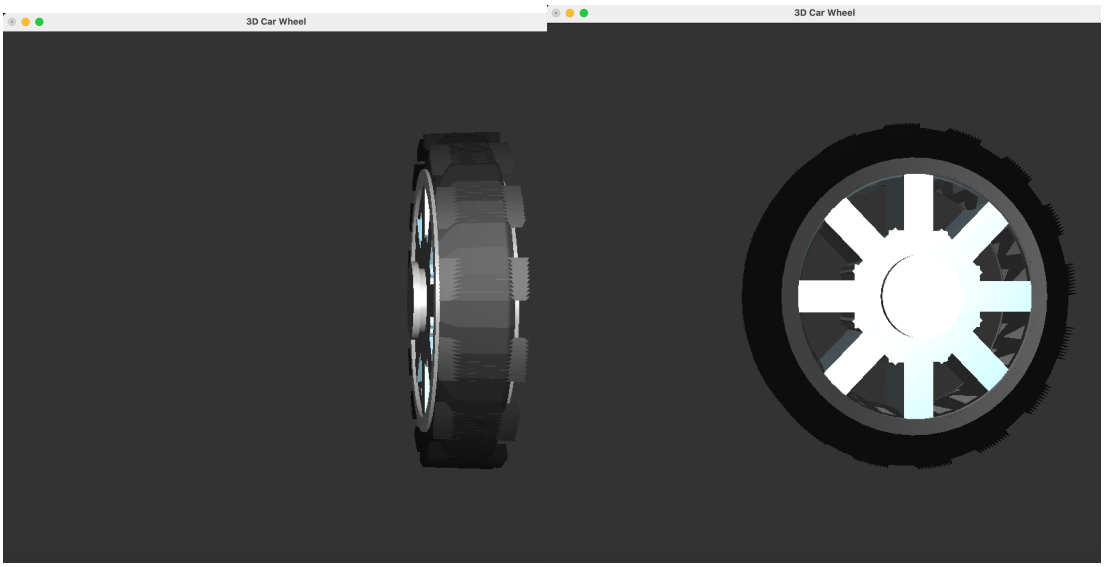
Рух колеса відбувається з одної частини екрана в іншу у вічному циклі. Також колесо обертається довкола своєї осі, щоб можна було точно впевнитись, що всі компоненти відмальовані правильно. Зміна

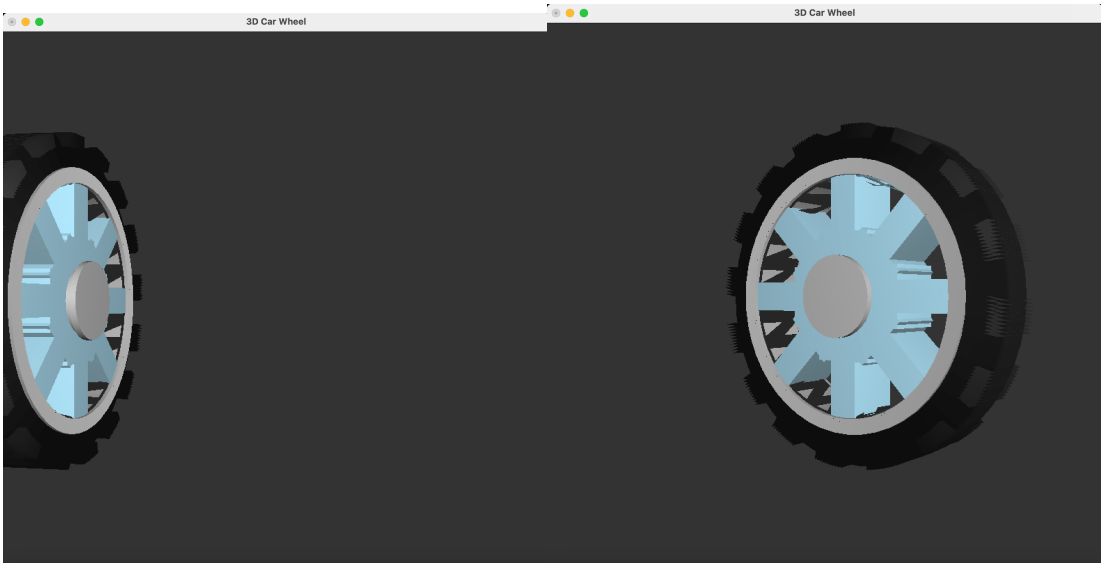
На наступних скриншотах можна побачити як виглядає модель із усіма відмальованими компонентами.

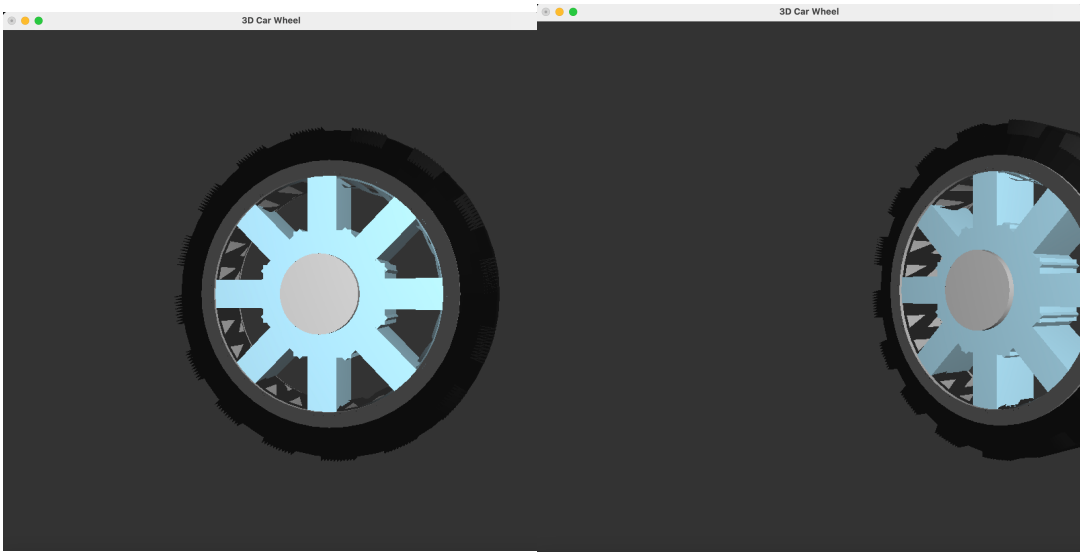
Як бачимо, у моделі наявні основні характерні для колеса елементи. Хоча й у реальному колесі їх набагато більше, але відмалювати більш деталізовану версію достатньо складно. Тож в рамках роботи більшу частину уваги було приділено саме принципам відмальовки 3d об’єктів, а не гіпер реалістичності.

Також можна добре помітити, що відображення елементів підлаштовується під освітлення, а завдяки параметру, що регулює блиск, деякі елементи відблискують більше за інші, даючи краще розуміння властивостей матеріалу, з якого зроблена та чи інша частина колеса.



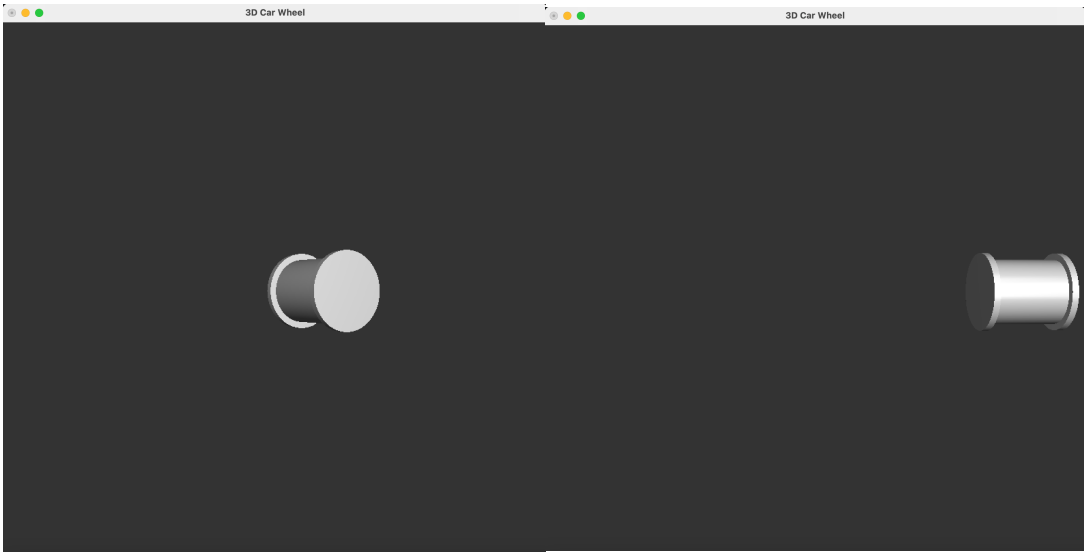

3D Car Wheel
3D Car Wheel


3D Car Wheel
3D Car Wheel

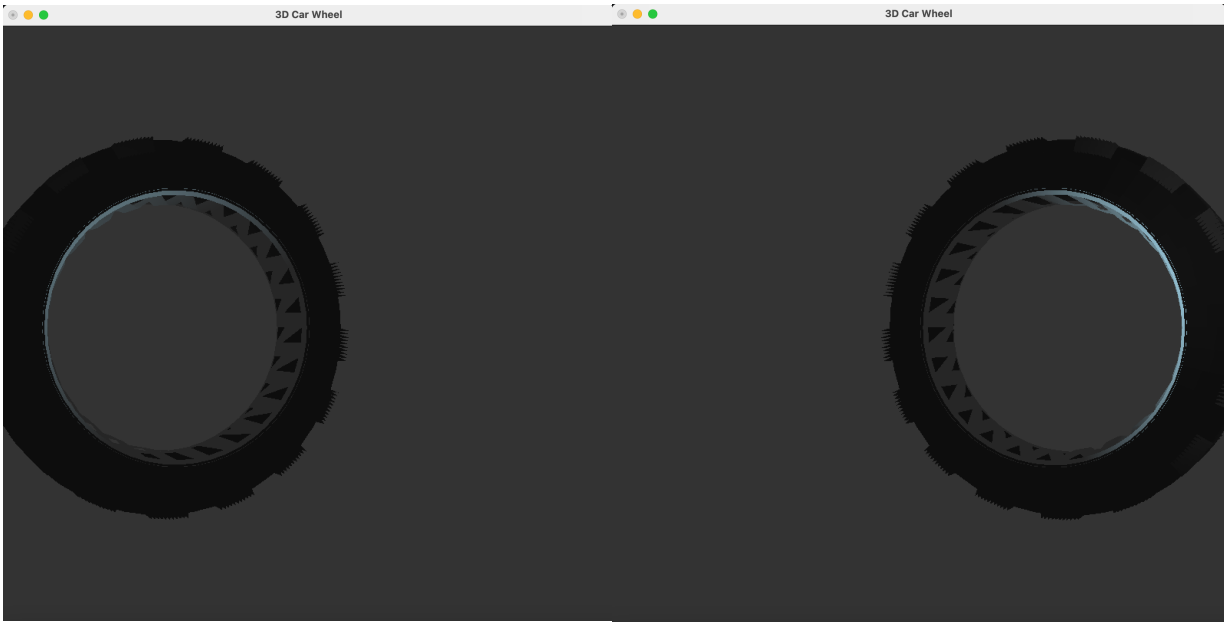

3D Car Wheel
3D Car Wheel

Так як деякі компоненти перекриваються іншими під час відмальовки, то додатково можна подивитись як вони виглядають самостійно:

Серцевина:



Барабан і шина:



Програмний код, що забезпечує отримання результату:

main.py

```
from OpenGL.GLUT import *

from WheelRenderer import WheelRenderer
```

```
def main():
    glutInit()
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
    glutInitWindowSize(800, 800)
    glutCreateWindow(b"3D Car Wheel")

    renderer = WheelRenderer()
    renderer.init_gl()

    glutDisplayFunc(renderer.draw)
    glutReshapeFunc(renderer.reshape)
    glutMainLoop()


if __name__ == '__main__':
    main()
```

WheelDimensions.py

```
from dataclasses import dataclass

@dataclass
class WheelDimensions:
    rim_radius: float = 8.0
    tire_width: float = 4.0
    tire_radius: float = 10.0
    hub_radius: float = 2.0
    spoke_count: int = 8
    spoke_width: float = 1.8
```

WheelComponents.py

```
from math import pi, cos, sin, sqrt

from OpenGL.raw.GL.VERSION.GL_1_1 import *
from WheelDimensions import WheelDimensions


class WheelComponents:
    def __init__(self, dimensions: WheelDimensions):
        self.dims = dimensions
        # Material colors
        self.tire_color = (0.1, 0.1, 0.1, 1.0)  # Black
        self.tire_pattern_color = (0.2, 0.2, 0.2, 1.0) # Dark grey
        self.rim_color = (0.5, 0.5, 0.5, 1.0) # Grey
        self.hub_color = (0.75, 0.75, 0.75, 1.0)  # Light grey
        self.spoke_color = (0.7, 0.9, 1.0, 0.5)  # Light Blue
        self.barrel_color = (0.7, 0.9, 1.0, 0.5) # Light Blue

    def create_wheel(self):
        """Creates the complete wheel"""
        self.create_hub()
        self.create_barrel()
```

```
        self.create_rim()
        self.create_spokes()
        self.create_tire()

    def create_tire(self):
        """Creates a tire with flat outer surface"""
        glDisable(GL_COLOR_MATERIAL)
        glMaterialfv(GL_FRONT, GL_AMBIENT, (0.1, 0.1, 0.1, 1.0))
        glMaterialfv(GL_FRONT, GL_DIFFUSE, self.tire_color)
        glMaterialfv(GL_FRONT, GL_SPECULAR, (0.3, 0.3, 0.3, 1.0))
        glMaterialf(GL_FRONT, GL_SHININESS, 10.0)

        # Create main tire body with flat profile
        self._create_tire_body()

        # Tread pattern
        glMaterialfv(GL_FRONT, GL_DIFFUSE, self.tire_pattern_color)
        self._create_tire_tread()

    def _create_tire_body(self):
        """Creates a tire body"""
        segments = 50
        sides = 20

        # Parameters for tire profile
        outer_flat_ratio = 0.5  # How much of the outer surface should be
flat
        shoulder_radius = self.dims.tire_width * 0.7  # Radius for the
shoulder curve

        glBegin(GL_QUAD_STRIP)
        for i in range(segments + 1):
            angle = i * 2 * pi / segments

            # Create cross-section points
            for j in range(sides + 1):
                t = j / sides

                # Calculate radius and height for this point
                if t < (1 - outer_flat_ratio) / 2:  # Inner shoulder
                    # Create curved transition from sidewall to tread
                    shoulder_t = t / ((1 - outer_flat_ratio) / 2)
                    r = self.dims.tire_radius - shoulder_radius * (1 -
sin(shoulder_t * pi / 2))
                    z = self.dims.tire_width * (-0.5 + t)
                elif t > (1 + outer_flat_ratio) / 2:  # Outer shoulder
                    # Create curved transition from tread to sidewall
                    shoulder_t = (1 - t) / ((1 - outer_flat_ratio) / 2)
                    r = self.dims.tire_radius - shoulder_radius * (1 -
sin(shoulder_t * pi / 2))
                    z = self.dims.tire_width * (-0.5 + t)
                else:  # Flat tread section
                    r = self.dims.tire_radius
                    z = self.dims.tire_width * (-0.5 + t)

                # Calculate position
                x = r * cos(angle)
```

```
                y = r * sin(angle)

                # Calculate normal
                if t < (1 - outer_flat_ratio) / 2 or t > (1 +
outer_flat_ratio) / 2:
                    # For shoulders, normal points outward at an angle
                    normal_angle = angle
                    normal_z = 0.3 if t < 0.5 else -0.3
                    normal_x = cos(normal_angle)
                    normal_y = sin(normal_angle)
                else:
                    # For flat section, normal points straight outward
                    normal_x = cos(angle)
                    normal_y = sin(angle)
                    normal_z = 0

                # Normalize the normal vector
                length = sqrt(normal_x ** 2 + normal_y ** 2 + normal_z ** 2)
                normal_x /= length
                normal_y /= length
                normal_z /= length

                glNormal3f(normal_x, normal_y, normal_z)
                glVertex3f(x, y, z)

                # Calculate next point for proper quad strip formation
                next_angle = (i + 1) * 2 * pi / segments
                x = r * cos(next_angle)
                y = r * sin(next_angle)
                glVertex3f(x, y, z)
        glEnd()

    def _create_tire_tread(self):
        """Creates a tire tread pattern"""
        tread_count = 16  # number of tread segments
        tread_depth = 0.8  # depth
        angle_width = 0.2  # width of each tread

        for i in range(tread_count):
            angle_start = i * 2 * pi / tread_count
            angle_end = angle_start + angle_width

            # Create main tread groove
            self._create_tread_segment(angle_start, angle_end, tread_depth)

    def _create_tread_segment(self, angle_start, angle_end, depth):
        """Creates a single tread segment"""
        segments = 10
        r1 = self.dims.tire_radius
        r2 = r1 - depth

        glBegin(GL_QUAD_STRIP)
        for i in range(segments + 1):
            t = i / segments
            angle = angle_start + (angle_end - angle_start) * t

            # Outer point
```

```
            x1 = r1 * cos(angle)
            y1 = r1 * sin(angle)
            glNormal3f(cos(angle), sin(angle), 0)
            glVertex3f(x1, y1, self.dims.tire_width / 2)
            glVertex3f(x1, y1, -self.dims.tire_width / 2)

            # Inner point
            x2 = r2 * cos(angle)
            y2 = r2 * sin(angle)
            glNormal3f(cos(angle), sin(angle), 0)
            glVertex3f(x2, y2, self.dims.tire_width / 2)
            glVertex3f(x2, y2, -self.dims.tire_width / 2)
        glEnd()

    def create_spokes(self):
        """Creates wheel spokes"""
        glMaterialfv(GL_FRONT, GL_AMBIENT, (0.3, 0.3, 0.3, 1.0))
        glMaterialfv(GL_FRONT, GL_DIFFUSE, self.spoke_color)
        glMaterialfv(GL_FRONT, GL_SPECULAR, (1.0, 1.0, 1.0, 1.0))
        glMaterialf(GL_FRONT, GL_SHININESS, 95.0)

        spoke_length = (self.dims.rim_radius - self.dims.hub_radius) * 0.9
        spoke_width = self.dims.spoke_width
        spoke_depth = self.dims.tire_width * 0.9

        for i in range(self.dims.spoke_count):
            angle = i * 2 * pi / self.dims.spoke_count

            glPushMatrix()
            glRotatef(angle * 180 / pi, 0, 0, 1)
            glTranslatef(self.dims.hub_radius + spoke_length / 2, 0, 0)

            # Main spoke body
            glPushMatrix()
            glScalef(spoke_length, spoke_width, spoke_depth)
            _create_box()
            glPopMatrix()

            # Spoke reinforcement near hub
            glPushMatrix()
            glTranslatef(-spoke_length / 2, 0, 0)
            glScalef(spoke_width * 2, spoke_width * 2, spoke_depth)
            _create_box()
            glPopMatrix()

            glPopMatrix()

    def create_rim(self):
        """Creates the wheel rim"""
        glMaterialfv(GL_FRONT, GL_AMBIENT, self.rim_color)
        glMaterialfv(GL_FRONT, GL_DIFFUSE, self.rim_color)
        glMaterialfv(GL_FRONT, GL_SPECULAR, (1.0, 1.0, 1.0, 1.0))
        glMaterialf(GL_FRONT, GL_SHININESS, 95.0)

        _create_cylinder_no_cups(self.dims.rim_radius, self.dims.rim_radius -
0.9, self.dims.tire_width)
```

```
    def create_hub(self):
        """Creates the central hub"""
        glMaterialfv(GL_FRONT, GL_AMBIENT, (0.3, 0.3, 0.3, 1.0))
        glMaterialfv(GL_FRONT, GL_DIFFUSE, self.hub_color)
        glMaterialfv(GL_FRONT, GL_SPECULAR, (1.0, 1.0, 1.0, 1.0))
        glMaterialf(GL_FRONT, GL_SHININESS, 50.0)

        # Main hub cylinder
        _create_cylinder(self.dims.hub_radius, self.dims.tire_width * 1.2)

        # Hub cap
        glPushMatrix()
        glTranslatef(0, 0, self.dims.tire_width * 0.6)
        _create_cylinder(self.dims.hub_radius * 1.2, self.dims.tire_width *
0.1)
        glPopMatrix()

        glPushMatrix()
        glTranslatef(0, 0, -self.dims.tire_width * 0.6)
        _create_cylinder(self.dims.hub_radius * 1.2, self.dims.tire_width *
0.1)
        glPopMatrix()

    def create_barrel(self):
        """Creates a detailed wheel barrel"""
        glMaterialfv(GL_FRONT, GL_AMBIENT, (0.3, 0.3, 0.3, 1.0))
        glMaterialfv(GL_FRONT, GL_DIFFUSE, self.barrel_color)
        glMaterialfv(GL_FRONT, GL_SPECULAR, (1.0, 1.0, 1.0, 1.0))
        glMaterialf(GL_FRONT, GL_SHININESS, 90.0)

        barrel_radius = self.dims.rim_radius - 1 # Slightly smaller than rim
        barrel_width = self.dims.tire_width * 0.8  # 80% of tire width
        lip_height = self.dims.tire_width * 0.1

        # Create main barrel body
        segments = 50
        glBegin(GL_QUAD_STRIP)
        for i in range(segments + 1):
            angle = i * 2 * pi / segments
            x = barrel_radius * cos(angle)
            y = barrel_radius * sin(angle)

            # Create slight concave shape
            offset = 0.3 * sin(pi * (barrel_width / 2) / barrel_width)

            # Inside vertex (with concave shape)
            inner_radius = barrel_radius - offset
            x_in = inner_radius * cos(angle)
            y_in = inner_radius * sin(angle)

            glNormal3f(cos(angle), sin(angle), 0)
            glVertex3f(x, y, barrel_width / 2)
            glVertex3f(x_in, y_in, 0)

            # Outside vertex
            glVertex3f(x, y, -barrel_width / 2)
        glEnd()
```

```
        # Create barrel lips (outer edges)
        for z in [-barrel_width / 2, barrel_width / 2]:
            glBegin(GL_QUAD_STRIP)
            lip_outer_radius = barrel_radius + 0.3
            for i in range(segments + 1):
                angle = i * 2 * pi / segments
                # Inner lip vertex
                x_in = barrel_radius * cos(angle)
                y_in = barrel_radius * sin(angle)
                # Outer lip vertex
                x_out = lip_outer_radius * cos(angle)
                y_out = lip_outer_radius * sin(angle)

                glNormal3f(cos(angle), sin(angle), 0.2 if z > 0 else -0.2)
                glVertex3f(x_in, y_in, z)
                glVertex3f(x_out, y_out, z + (lip_height if z > 0 else -
lip_height))
            glEnd()


def _create_cylinder(radius, height):
    """Helper method to create a cylinder"""
    segments = 50
    glBegin(GL_QUAD_STRIP)
    for i in range(segments + 1):
        angle = i * 2 * pi / segments
        x = radius * cos(angle)
        y = radius * sin(angle)

        glNormal3f(cos(angle), sin(angle), 0)
        glVertex3f(x, y, height / 2)
        glVertex3f(x, y, -height / 2)
    glEnd()

    # End caps
    for z in [-height / 2, height / 2]:
        glBegin(GL_TRIANGLE_FAN)
        glNormal3f(0, 0, 1 if z > 0 else -1)
        glVertex3f(0, 0, z)
        for i in range(segments + 1):
            angle = i * 2 * pi / segments
            x = radius * cos(angle)
            y = radius * sin(angle)
            glVertex3f(x, y, z)
        glEnd()

def _create_cylinder_no_cups(outer_radius, inner_radius, height):
    """Helper method to create a thick cylinder without cups"""
    segments = 50

    # Outer surface of the cylinder
    glBegin(GL_QUAD_STRIP)
    for i in range(segments + 1):
        angle = i * 2 * pi / segments
        x = outer_radius * cos(angle)
        y = outer_radius * sin(angle)
```

```
        glNormal3f(cos(angle), sin(angle), 0)
        glVertex3f(x, y, height / 2)
        glVertex3f(x, y, -height / 2)
    glEnd()

    # Inner surface of the cylinder
    glBegin(GL_QUAD_STRIP)
    for i in range(segments + 1):
        angle = i * 2 * pi / segments
        x = inner_radius * cos(angle)
        y = inner_radius * sin(angle)

        glNormal3f(-cos(angle), -sin(angle), 0)
        glVertex3f(x, y, height / 2)
        glVertex3f(x, y, -height / 2)
    glEnd()

    # Connect the outer and inner surfaces
    glBegin(GL_QUADS)
    for i in range(segments):
        angle1 = i * 2 * pi / segments
        angle2 = (i + 1) * 2 * pi / segments

        x1_outer, y1_outer = outer_radius * cos(angle1), outer_radius *
sin(angle1)
        x2_outer, y2_outer = outer_radius * cos(angle2), outer_radius *
sin(angle2)
        x1_inner, y1_inner = inner_radius * cos(angle1), inner_radius *
sin(angle1)
        x2_inner, y2_inner = inner_radius * cos(angle2), inner_radius *
sin(angle2)

        # Top cap
        glVertex3f(x1_outer, y1_outer, height / 2)
        glVertex3f(x2_outer, y2_outer, height / 2)
        glVertex3f(x2_inner, y2_inner, height / 2)
        glVertex3f(x1_inner, y1_inner, height / 2)

        # Bottom cap
        glVertex3f(x1_outer, y1_outer, -height / 2)
        glVertex3f(x2_outer, y2_outer, -height / 2)
        glVertex3f(x2_inner, y2_inner, -height / 2)
        glVertex3f(x1_inner, y1_inner, -height / 2)
    glEnd()



def _create_box():
    """Helper method to create a unit cube"""
    vertices = [
        (0.5, 0.5, 0.5), (-0.5, 0.5, 0.5),
        (-0.5, -0.5, 0.5), (0.5, -0.5, 0.5),
        (0.5, 0.5, -0.5), (-0.5, 0.5, -0.5),
        (-0.5, -0.5, -0.5), (0.5, -0.5, -0.5)
    ]
    faces = [
```

```
        (0, 1, 2, 3), (5, 4, 7, 6), (4, 0, 3, 7),
        (1, 5, 6, 2), (4, 5, 1, 0), (3, 2, 6, 7)
    ]
    normals = [
        (0, 0, 1), (0, 0, -1), (1, 0, 0),
        (-1, 0, 0), (0, 1, 0), (0, -1, 0)
    ]

    glBegin(GL_QUADS)
    for i, face in enumerate(faces):
        glNormal3fv(normals[i])
        for vertex in face:
            glVertex3fv(vertices[vertex])
    glEnd()
```

WheelRender.py

```
from WheelComponents import WheelComponents
from OpenGL.GL import *
from OpenGL.GLUT import *
import time
from OpenGL.raw.GLU import gluLookAt, gluPerspective
from WheelDimensions import WheelDimensions

class WheelRenderer:
    def __init__(self, width=800, height=800):
        self.width = width
        self.height = height
        self.rotation = 0
        self.translation = -20  # Start position of the wheel (off-screen to
the left)
        self.direction = 1  # Moving direction (1 = right, -1 = left)
        self.wheel = WheelComponents(WheelDimensions())
        self.display_list = None

    def init_gl(self):
        glClearColor(0.2, 0.2, 0.2, 1.0)
        glEnable(GL_DEPTH_TEST)
        glEnable(GL_LIGHTING)
        glEnable(GL_LIGHT0)
        glEnable(GL_NORMALIZE)
        glDisable(GL_COLOR_MATERIAL)

        # Lighting setup
        glLightfv(GL_LIGHT0, GL_POSITION, (10, 10, 10, 1))
        glLightfv(GL_LIGHT0, GL_AMBIENT, (0.3, 0.3, 0.3, 1))
        glLightfv(GL_LIGHT0, GL_DIFFUSE, (1, 1, 1, 1))
        glLightfv(GL_LIGHT0, GL_SPECULAR, (1, 1, 1, 1))

        # Create display list
        self.display_list = glGenLists(1)
        glNewList(self.display_list, GL_COMPILE)
        self.wheel.create_wheel()
        glEndList()
```

```
    def reshape(self, width, height):
        self.width = width
        self.height = height
        glViewport(0, 0, width, height)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(45, width / height, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)

    def draw(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        gluLookAt(0, 0, 40, 0, 0, 0, 0, 1, 0)

        glPushMatrix()
        # Translate the wheel horizontally and rotate it
        glTranslatef(self.translation, 0, 0)  # Move along the X-axis
        glRotatef(self.rotation, 0, 1, 0)  # Rotate around the Y-axis
        glCallList(self.display_list)
        glPopMatrix()

        # Update rotation
        self.rotation += 1
        if self.rotation >= 360:
            self.rotation = 0

        # Update translation
        self.translation += 0.1 * self.direction  # Increment position
        if self.translation > 20:  # If it moves off-screen to the right
            self.direction = -1  # Reverse direction (move left)
        elif self.translation < -20:  # If it moves off-screen to the left
            self.direction = 1  # Reverse direction (move right)

        glutSwapBuffers()
        time.sleep(0.01)
        glutPostRedisplay()
```

Висновки: Виконавши лабораторну роботу №9, я навчилась реалізовувати алгоритми, що забезпечують створення реалістичних 3d моделей, навчилась управляти світлом та ракурсом спостереження за створеними моделями і попрактикувалась працювати із бібліотекою OpenGL. Ця робота точно підвищила моє розуміння створення з нуля різних об’єктів у просторі і дала базове розуміння принципів роботи із подібними завданнями, що в подальшому може значно полегшити роботу із завданнями, де подібні 3d моделі мають використовуватись як допоміжні, наприклад, для навчання нейромережі.