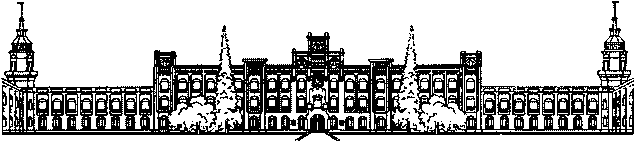
****

Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет інформатики та обчислювальної техніки

Кафедра інформаційних систем та технологій

Лабораторна робота №9

**З дисципліни «Технології Computer Vision»**

*СИНТЕЗ РЕАЛІСТИЧНИХ ОБ’ЄКТІВ ДОПОВНЕНОЇ РЕАЛЬНОСТІ*

| Виконала  студент кафедри ІСТ ФІОТ, групи ІА-12: |  | Перевірив:  пос. Баран Д. Р. |
| --- | --- | --- |
| Яковенко Д. О. |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Київ 2024

**І. Мета:**

Дослідити методологію і технології створення доповненої реальності.

**ІІ. Завдання:**

*Реалізація проекту триває та спрямовано на збільшення функціональності*

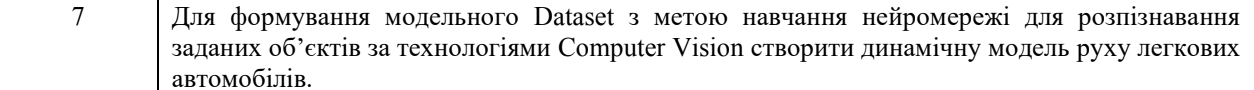
*програмної компоненти. Лабораторія провідної ІТ-компанії реалізує масштабний проект розробки універсальної платформи з цифрової обробки зображень для задач Computer Vision. Платформа передбачає розташування back-end компоненти на власному хмарному сервері з наданням повноважень користувачам заздалегідь адаптованого front-end функціоналу універсальної платформи. Цим формується унікальна для потреб замовника ERP система з технологіями Computer Visio. Замовниками ресурсів платформи є: державні та комерційні компанії, що розробляють медичне*

*обладнання з діагностування захворювань за візуальною інформацією; автоматизації аграрного бізнесу в аспекті обліку посівних територій за даними з БПЛА; візуального контролю безпекових заходів на об’єктах*

*критичної інфраструктури: аеропорти, торгівельно-розважальні центри, житлові комплекси тощо.*

**Завдання ІІ рівня складності – максимально 9 балів.**

**Реалізувати виконання завдання таблиці 1 додатку.**

****

Я виконала сьомий варіант другого рівня лабораторної роботи, оскільки мене в цілому цікавлять автомобілі і я хотіла б створити їх за допомогою інструментів OpenGL.

**ІІІ. Результати виконання лабораторної роботи.**

**3.1. Синтезована математична модель перетворень графічних об’єктів відповідно до індивідуального завдання.**

Відповідно до умов задачі синтезовано математичну модель для створення анімації 3D-сцени з автомобілями, які рухаються по дорозі з чотирма смугами. Вона використовує OpenGL для рендерингу графіки та GLUT для управління вікнами та подіями. Встановлюються початкові параметри OpenGL, такі як колір фону, режим проекції та освітлення. Вмикаються глибина та колірний матеріал для об’єктів. Потім завантажуються чотири різні моделі автомобілів з відповідних модулів (car1, car2, car3, car4) і кожен автомобіль отримує випадкові позиції, кольори та швидкості. Далі генерується дорога з чотирма смугами та кущами вздовж дороги з обої боків. Автомобілі розташовуються на дорозі випадковим чином і рухаються вперед або назад у залежності від їх початкового положення і смуги, де вони згенерувалися. Автомобілі малюються з використанням кольорів, визначених під час ініціалізації. Позиції автомобілів оновлюються у кожному кадрі, а також вони можуть змінювати швидкість, щоб уникнути зіткнень із попереду йдучими автомобілями. Коли авто виходить за межі екрану, він переналаштовується на протилежний кінець дороги з новою позицією та швидкістю.Обробляються події клавіатури для виходу з програми та виконується безперервне оновлення та перерисування сцени для створення анімації.

### Синтезована математична модель перетворень графічних об’єктів

**Математична модель перетворень** у програмі включає наступні компоненти:

1. **Трансляція**:
   * Кожен автомобіль транслюється вздовж осі X на відстань, яка залежить від його швидкості та напряму руху.
   * Наприклад, у функції draw\_scene() використовується glTranslatef(car['x\_position'], 0, car['lane']) для переміщення автомобіля в потрібне місце на дорозі.
2. **Ротація**:
   * Автомобілі, які рухаються в зворотному напрямку, обертаються на 180 градусів, щоб вони правильно відображалися.
   * Використовується glRotatef(180, 0, 1, 0) для обертання автомобіля.
3. **Згладжування та освітлення**:
   * Використовується згладжування (GL\_SMOOTH) для плавного переходу між кольорами поверхні об’єктів.
   * Освітлення налаштовується через джерела світла, що додають реалізму сцені.
4. **Моделі автомобілів**:
   * Моделі автомобілів включають геометричні форми та матеріали, які обробляються для рендерингу в різних кольорах.
5. **Кінематика автомобілів**:
   * Швидкість і напрямок руху автомобілів оновлюються за допомогою простої кінематичної моделі, яка враховує швидкість та відстань до інших автомобілів.
   * Приклад у функції update\_cars() показує, як автомобілі коректують свою швидкість для уникнення зіткнень.

**3.2. Блок схема алгоритму та її опис.**

Алгоритм програми починається з імпорту необхідних модулів OpenGL, GLUT, GLU, а також модулів, що відповідають за відображення різних автомобілів. Далі ініціалізується глобальний список автомобілів, в якому зберігаються їхні позиції та швидкості. Після цього визначаються кілька допоміжних функцій. Функція `generate\_main\_color()` генерує випадковий основний колір для автомобіля, використовуючи випадкові значення для компонентів кольору RGB. Функція `generate\_random\_color(main\_color)` створює другорядний колір, відрізняючи його від основного, щоб забезпечити візуальну різноманітність.

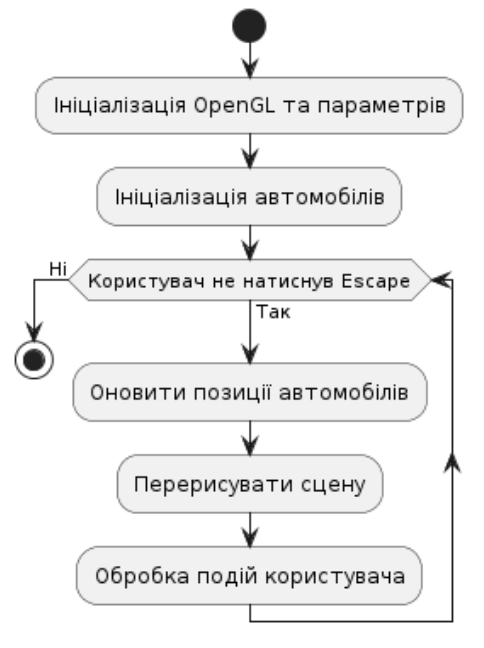
Для перевірки, чи вільна позиція для нового автомобіля, використовується функція `is\_position\_clear(x\_position, lane, min\_distance=6)`. Вона проходиться по всіх автомобілях та перевіряє, чи є достатня відстань між новою позицією та вже існуючими автомобілями в тій же смузі руху.

Основна функція для ініціалізації автомобілів, `init\_cars(num\_cars)`, генерує задану кількість автомобілів з випадковими параметрами, такими як початкова позиція, швидкість, колір, смуга руху та напрямок. Вона також обирає випадковий модуль автомобіля з доступних.

Функція `draw\_scene()` відповідає за малювання сцени. Вона малює дорогу, центральну суцільну лінію, пунктирні лінії для інших смуг, а також самі автомобілі з їхніми випадковими кольорами. Крім того, вона малює кущі вздовж дороги для додаткового візуального ефекту. Для малювання окремих кущів використовується функція `draw\_bush(x, z)`, яка малює сферу зеленого кольору на заданих координатах.

Функція `update\_cars()` відповідає за оновлення позицій та швидкостей автомобілів. Вона перевіряє, чи немає попереду автомобіля іншого автомобіля на тій же смузі руху і, у разі потреби, коригує швидкість. Функція також обробляє плавне прискорення та гальмування автомобілів. Якщо автомобіль виходить за межі екрану, він перевстановлюється на нову позицію з випадковими параметрами.

Функція `display()` очищає буфери, встановлює камеру та малює сцену. Функція `idle()` викликає оновлення позицій автомобілів та перерисовку сцени. Функція `keyboard(key, x, y)` обробляє натискання клавіш і дозволяє вийти з програми при натисканні клавіші Escape. Ініціалізація OpenGL відбувається у функції `init()`, де встановлюються параметри глибини, кольору фону, режиму проекції та освітлення. Головна функція `main()` ініціалізує GLUT, створює вікно, ініціалізує автомобілі та запускає головний цикл програми. Цикл програми продовжується до тих пір, поки користувач не натисне клавішу Escape. У циклі оновлюються позиції автомобілів, перерисовується сцена та обробляються події користувача.



| Рис.1. Блок-схема алгоритму основної програми. |
| --- |
|  |

Існує також чотири файли з побудовою модельок автомобілів. Спочатку встановлюються основні розміри автомобіля та його коліс, а також кольори для основних і другорядних частин автомобіля. Камера налаштована на обертання по осях X та Y.

Основна функція для малювання трапецієподібної кабіни `draw\_trapezoidal\_cabin` визначає вершини кабіни та використовує їх для малювання шести граней трапецієподібного об'єкта. Використовуючи `glBegin(GL\_QUADS)`, малюються передня, задня, бокові, верхня та нижня поверхні кабіни. Функція `draw\_wheel\_arch` малює арку колеса, використовуючи диск з бібліотеки GLU. Колір арки збігається з основним кольором автомобіля.

Функція `draw\_car\_body` відповідає за малювання основного корпусу автомобіля. Вона створює основний куб для корпусу та кілька додаткових деталей, таких як дах, капот, бампери, фари, задні ліхтарі та номерні знаки. Використовуючи різні кольори та пропорції, програма додає ці деталі до автомобіля. Функція `draw\_wheel` малює колеса автомобіля, використовуючи `glutSolidTorus` для створення тороїдальних об'єктів, що імітують форму колеса. Функція `draw\_car` відповідає за малювання всього автомобіля, включаючи корпус, кабіну та колеса. Вона викликає функції `draw\_car\_body`, `draw\_trapezoidal\_cabin` та `draw\_wheel` у відповідних позиціях, щоб створити повний автомобіль.

Функція `display` оновлює екран, очищаючи буфери та встановлюючи камеру з певними параметрами. Вона застосовує обертання на основі кутів `angle\_x` та `angle\_y`, а потім викликає функцію `draw\_car` для малювання автомобіля.

Функція `keyboard` обробляє натискання клавіш для обертання камери навколо автомобіля або виходу з програми при натисканні клавіші Escape. Вона коригує значення кутів `angle\_x` та `angle\_y` відповідно до натиснутих клавіш, а потім оновлює дисплей.

Функція `init` ініціалізує налаштування OpenGL, включаючи тест глибини, кольори фону, режим проекції та встановлює перспективу. Основна функція `main` ініціалізує GLUT, встановлює режим відображення, створює вікно, викликає функцію `init` для налаштування OpenGL, встановлює функції зворотного виклику для відображення та обробки клавіш, а потім запускає головний цикл GLUT.

В результаті, в програмах створюються тривимірні моделі автомобілів, які можна обертати за допомогою клавіш.

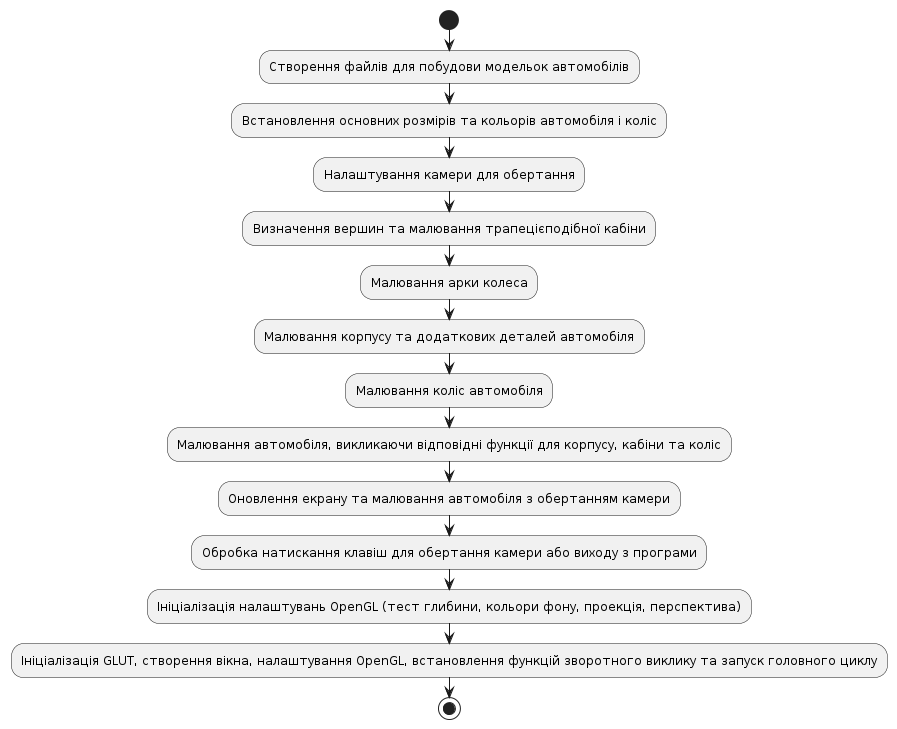


Рис.2. Блок-схема алгоритму для створення авто.

**3.3. Опис структури проекту програми в середовищі PyCharm.**

Для реалізації розробленого алгоритму мовою програмування Python з використанням можливостей інтегрованого середовища PyCharm сформовано проект.

Проект базується на лінійній бізнес-логіці функціонального програмування та має таку структуру.

|  |
| --- |
| Рис.3. Структура проекту. |

CVLab\_9 – головний каталог проекту

cars – папка з різними моделями автомобілів

car1.py, car2.py, car3.py, car4.py – файли з побудовою моделей авто

lab9.py – основний файл лабораторної роботи для відображення 3D сцени;

**3.4. Результати роботи програми відповідно до завдання.**

Результатом роботи програми є програма, в якій при її запуску отримуємо такий результат:

|  |
| --- |
| Рис.4. Результат виконання. |

**3.5. Програмний код.**

Програмний код послідовно реалізує алгоритми для виконання завдання другого рівня складності.

При цьому використано можливості Python бібліотек: openGL, random.

**Код основного файлу:**

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

import cars.car1 as car1

import cars.car2 as car2

import cars.car3 as car3

import cars.car4 as car4

import random

# Store car positions and speeds

cars = []

def generate\_main\_color():

return (random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1))

def generate\_random\_color(main\_color):

r = max(0, min(1, main\_color[0] - 0.1))

g = max(0, min(1, main\_color[1] - 0.1))

b = max(0, min(1, main\_color[2] - 0.1))

return (r, g, b)

def is\_position\_clear(x\_position, lane, min\_distance=6):

for car in cars:

if car['lane'] == lane and abs(car['x\_position'] - x\_position) < min\_distance:

return False

return True

def init\_cars(num\_cars):

global cars

cars = []

lane\_positions = [-4.5, -1.5, 1.5, 4.5] # Four lanes on the road

for \_ in range(num\_cars):

while True:

x\_position = random.uniform(-20, 20)

lane = random.choice(lane\_positions)

if is\_position\_clear(x\_position, lane):

break

speed = random.uniform(0.05, 0.3)

direction = 1 if lane > 0 else -1 # Set direction based on lane

car\_module = random.choice([car1, car2, car3, car4])

main\_color = generate\_main\_color()

second\_color = generate\_random\_color(main\_color)

car = {

'module': car\_module,

'x\_position': x\_position,

'speed': speed,

'main\_color': main\_color,

'second\_color': second\_color,

'wheel\_color': (0.5, 0.5, 0.5),

'lane': lane,

'target\_speed': speed,

'direction': direction

}

cars.append(car)

def draw\_scene():

# Draw road

glColor3f(0.3, 0.3, 0.3)

glBegin(GL\_QUADS)

glVertex3f(-30, 0, -6)

glVertex3f(30, 0, -6)

glVertex3f(30, 0, 6)

glVertex3f(-30, 0, 6)

glEnd()

# Draw solid double line in the middle

glColor3f(1.0, 1.0, 1.0)

glLineWidth(2)

glBegin(GL\_LINES)

glVertex3f(-20, 0.02, -0.1)

glVertex3f(20, 0.02, -0.1)

glVertex3f(-20, 0.02, 0.1)

glVertex3f(20, 0.02, 0.1)

glEnd()

# Draw dashed lines for other lanes

glLineWidth(1)

for z in [-3, 3]:

for i in range(-20, 20, 2):

glBegin(GL\_LINES)

glVertex3f(i, 0.02, z)

glVertex3f(i + 1, 0.02, z)

glEnd()

# Draw cars with random colors

for car in cars:

glPushMatrix()

glTranslatef(car['x\_position'], 0, car['lane'])

if car['direction'] == -1: # If car is moving backward, rotate it 180 degrees

glRotatef(180, 0, 1, 0)

car['module'].main\_color = car['main\_color']

car['module'].second\_color = car['second\_color']

car['module'].wheel\_color = car['wheel\_color']

car['module'].draw\_car()

glPopMatrix()

# Draw bushes along the road

for z in [-7, 7]:

for i in range(-20, 21, 2):

draw\_bush(i, z)

def draw\_bush(x, z):

glColor3f(0.0, 0.5, 0.0)

glPushMatrix()

glTranslatef(x, 0, z)

glutSolidSphere(1, 16, 16)

glPopMatrix()

def update\_cars():

for i, car in enumerate(cars):

# Reset target speed to the current speed

car['target\_speed'] = car['speed']

# Check for the car in front in the same lane and direction

for j, other\_car in enumerate(cars):

if i != j and car['lane'] == other\_car['lane'] and car['direction'] == other\_car['direction']:

distance = (other\_car['x\_position'] - car['x\_position']) \* car['direction']

if 0 < distance < 6:

if distance < 0.5:

car['target\_speed'] = 0

elif car['speed'] > other\_car['speed']:

car['target\_speed'] = other\_car['speed']

break

# Smooth acceleration and deceleration

if car['speed'] < car['target\_speed']:

car['speed'] = min(car['speed'] + 0.005, car['target\_speed'])

elif car['speed'] > car['target\_speed']:

car['speed'] = max(car['speed'] - 0.005, car['target\_speed'])

# Update car position based on direction

car['x\_position'] += car['speed'] \* car['direction']

if car['x\_position'] > 20 or car['x\_position'] < -20:

while True:

new\_x\_position = -20 if car['direction'] == 1 else 20

new\_lane = random.choice([-4.5, -1.5, 1.5, 4.5])

if is\_position\_clear(new\_x\_position, new\_lane):

break

car['x\_position'] = new\_x\_position

car['speed'] = random.uniform(0.05, 0.15)

car['target\_speed'] = car['speed']

car['module'] = random.choice([car1, car2, car3, car4])

car['main\_color'] = generate\_main\_color()

car['second\_color'] = generate\_random\_color(car['main\_color'])

car['lane'] = new\_lane

car['direction'] = 1 if new\_lane > 0 else -1 # Update direction

def display():

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity()

gluLookAt(-10.0, 30.0, 30.0, # Позиція камери

0.0, 0.0, 0.0, # Позиція, на яку дивиться камера

0.0, 1.0, 0.0) # Вектор, що вказує вертикальну орієнтацію камери

draw\_scene()

glutSwapBuffers()

def idle():

update\_cars()

glutPostRedisplay()

def keyboard(key, x, y):

if key == b'\x1b':

sys.exit()

def init():

glEnable(GL\_DEPTH\_TEST)

glClearColor(0.7, 0.4, 0.2, 1.0)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

gluPerspective(35, 1.0, 0.1, 100.0)

glMatrixMode(GL\_MODELVIEW)

# Set up lighting

glEnable(GL\_LIGHTING)

glEnable(GL\_LIGHT0)

glLightfv(GL\_LIGHT0, GL\_POSITION, [40.0, 25.0, 10.0, 0.0])

glLightfv(GL\_LIGHT0, GL\_AMBIENT, [0.2, 0.2, 0.2, 1.0])

glLightfv(GL\_LIGHT0, GL\_SPECULAR, [1.0, 5.0, 1.0, 1.0])

glEnable(GL\_COLOR\_MATERIAL)

glColorMaterial(GL\_FRONT, GL\_AMBIENT\_AND\_DIFFUSE)

glShadeModel(GL\_SMOOTH)

def main():

glutInit()

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH)

glutInitWindowSize(800, 600)

glutCreateWindow(b"3D Car Animation on Road")

init()

init\_cars(num\_cars=5)

glutDisplayFunc(display)

glutIdleFunc(idle)

glutKeyboardFunc(keyboard)

glutMainLoop()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Код побудови моделі авто:**

from OpenGL.GL import \*

from OpenGL.GLUT import \*

from OpenGL.GLU import \*

# Car body dimensions

car\_length = 4.0

car\_width = 2.0

car\_height = 1

# Wheel dimensions

wheel\_radius = 0.3

wheel\_width = 0.3

main\_color = (0.6, 0.2, 0.3)

second\_color = (0.5, 0.2, 0.3)

# Camera angles

angle\_y = 0

angle\_x = 0

def draw\_trapezoidal\_cabin(red, green, blue):

glPushMatrix()

glColor3f(red, green, blue)

# Define vertices for the trapezoidal cabin

cabin\_vertices = [

# Front face

[-car\_length / 1.5 / 4, 0, car\_width / 2],

[car\_length / 1.5 / 3.5, 0, car\_width / 2],

[car\_length / 1.5 / 6, car\_height / 1.1, car\_width / 2],

[-car\_length / 6, car\_height / 1.1, car\_width / 2],

# Back face

[-car\_length / 1.5 / 4, 0, -car\_width / 2],

[car\_length / 1.5 / 3.5, 0, -car\_width / 2],

[car\_length / 1.5 / 6, car\_height / 1.1, -car\_width / 2],

[-car\_length / 6, car\_height / 1.1, -car\_width / 2],

# Left face

[-car\_length / 1.5 / 4, 0, car\_width / 2],

[-car\_length / 1.5 / 4, 0, -car\_width / 2],

[-car\_length / 1.5 / 6, car\_height / 1.1, -car\_width / 2],

[-car\_length / 1.5 / 6, car\_height / 1.1, car\_width / 2],

# Right face

[car\_length / 1.5 / 4, 0, car\_width / 2],

[car\_length / 1.5 / 4, 0, -car\_width / 2],

[car\_length / 1.5 / 6, car\_height / 1.1, -car\_width / 2],

[car\_length / 1.5 / 6, car\_height / 1.1, car\_width / 2],

# Top face

[-car\_length / 1.5 / 6, car\_height / 1.1, car\_width / 2],

[car\_length / 1.5 / 6, car\_height / 1.1, car\_width / 2],

[car\_length / 1.5 / 6, car\_height / 1.1, -car\_width / 2],

[-car\_length / 1.5 / 6, car\_height / 1.1, -car\_width / 2],

# Bottom face

[-car\_length / 1.5 / 4, 0, car\_width / 2],

[car\_length / 1.5 / 4, 0, car\_width / 2],

[car\_length / 1.5 / 4, 0, -car\_width / 2],

[-car\_length / 1.5 / 4, 0, -car\_width / 2],

]

glTranslatef(0, car\_height, 0)

glBegin(GL\_QUADS)

for i in range(0, len(cabin\_vertices), 4):

for j in range(4):

glVertex3fv(cabin\_vertices[i + j])

glEnd()

glPopMatrix()

def draw\_wheel\_arch():

glPushMatrix()

glColor3f(main\_color[0], main\_color[1], main\_color[2]) # Same color as car body

glTranslatef(0, wheel\_radius, 0)

glRotatef(90, 1, 0, 0)

gluDisk(gluNewQuadric(), 0, wheel\_radius \* 1.5, 30, 1)

glPopMatrix()

def draw\_car\_body():

glPushMatrix()

# Main body

glColor3f(main\_color[0], main\_color[1], main\_color[2])

glScalef(car\_length, car\_height, car\_width)

glutSolidCube(1.0)

glPopMatrix()

# Second body

glPushMatrix()

glColor3f(second\_color[0], second\_color[1], second\_color[2])

glScalef(car\_length - 0.1, car\_height / 1.9, car\_width + 0.09)

glTranslatef(0, -car\_height / 2, 0)

glutSolidCube(1.0)

glPopMatrix()

# Roof

glPushMatrix()

glColor3f(second\_color[0], second\_color[1], second\_color[2])

glScalef(car\_length / 1.5, 0.1, car\_width - 0.3)

glTranslatef(-0.28, car\_height \* 13.5, 0)

glutSolidCube(1.0)

glPopMatrix()

# Hood

glPushMatrix()

glColor3f(second\_color[0], second\_color[1], second\_color[2])

glScalef(car\_length / 3, 0.2, car\_width - 0.2)

glTranslatef(0.99, car\_height \* 2.4, 0)

glutSolidCube(1.0)

glPopMatrix()

# Front bumper

glPushMatrix()

glColor3f(0.5, 0.5, 0.5) # Gray color

glScalef(car\_length / 4, 0.3, car\_width + 0.1)

glTranslatef(car\_length / 3 + 0.22, -car\_height \* 1.5, 0)

glutSolidCube(1.0)

glPopMatrix()

# Rear bumper

glPushMatrix()

glColor3f(0.5, 0.5, 0.5) # Gray color

glScalef(car\_length / 4, 0.3, car\_width + 0.1)

glTranslatef(-car\_length / 3 - 0.22, -car\_height \* 1.5, 0)

glutSolidCube(1.0)

glPopMatrix()

# Rear light left

glPushMatrix()

glColor3f(1.0, 0.0, 0.0)

glScalef(car\_length / 22, car\_height / 2.5, car\_width / 8)

glTranslatef(-car\_length - 7, car\_height / 2, -car\_width - 1.55)

glutSolidCube(1.0)

glPopMatrix()

# Rear light right

glPushMatrix()

glColor3f(1.0, 0.0, 0.0)

glScalef(car\_length / 22, car\_height / 2.5, car\_width / 8)

glTranslatef(-car\_length - 7, car\_height / 2, car\_width + 1.55)

glutSolidCube(1.0)

glPopMatrix()

# Rear number plate

glPushMatrix()

glColor3f(1.0, 1.0, 1.0)

glScalef(car\_length / 4.83, car\_height / 4, car\_width / 4)

glTranslatef(-car\_length / 2, car\_height / 2, 0)

glutSolidCube(1.0)

glPopMatrix()

# Front light left

glPushMatrix()

glColor3f(0.8, 0.8, 0.0)

glTranslatef(car\_length / 1.99, car\_height / 4, -car\_width / 2.8)

glRotatef(90, 0, 1, 0)

gluDisk(gluNewQuadric(), 0, 0.15, 30, 1)

glPopMatrix()

glPushMatrix()

glColor3f(0.8, 0.8, 0.0)

glTranslatef(car\_length / 1.99, car\_height / 5, -car\_width / 3.5)

glRotatef(90, 0, 1, 0)

gluDisk(gluNewQuadric(), 0, 0.1, 30, 1)

glPopMatrix()

# Front right left

glPushMatrix()

glColor3f(0.8, 0.8, 0.0)

glTranslatef(car\_length / 1.99, car\_height / 4, car\_width / 2.8)

glRotatef(90, 0, 1, 0)

gluDisk(gluNewQuadric(), 0, 0.15, 30, 1)

glPopMatrix()

glPushMatrix()

glColor3f(0.8, 0.8, 0.0)

glTranslatef(car\_length / 1.99, car\_height / 5, car\_width / 3.5)

glRotatef(90, 0, 1, 0)

gluDisk(gluNewQuadric(), 0, 0.1, 30, 1)

glPopMatrix()

# Orange lights

glPushMatrix()

glColor3f(0.9, 0.5, 0.0)

glScalef(car\_length / 30, car\_height / 4, car\_width / 20)

glTranslatef(car\_length \* 3.7, car\_height, car\_width \* 5)

glutSolidCube(1.0)

glPopMatrix()

glPushMatrix()

glColor3f(0.9, 0.5, 0.0)

glScalef(car\_length / 30, car\_height / 4, car\_width / 20)

glTranslatef(car\_length \* 3.7, car\_height, -car\_width \* 5)

glutSolidCube(1.0)

glPopMatrix()

glPushMatrix()

glColor3f(0.9, 0.5, 0.0)

glScalef(car\_length / 30, car\_height / 4, car\_width / 20)

glTranslatef(-car\_length \* 3.8, car\_height / 0.9, car\_width \* 5)

glutSolidCube(1.0)

glPopMatrix()

glPushMatrix()

glColor3f(0.9, 0.5, 0.0)

glScalef(car\_length / 30, car\_height / 4, car\_width / 20)

glTranslatef(-car\_length \* 3.8, car\_height / 0.9, -car\_width \* 5)

glutSolidCube(1.0)

glPopMatrix()

# Front number plate

glPushMatrix()

glColor3f(1.0, 1.0, 1.0)

glScalef(car\_length / 4.83, car\_height / 4, car\_width / 4)

glTranslatef(car\_length / 2, -car\_height, 0)

glutSolidCube(1.0)

glPopMatrix()

# Radiator grill

glPushMatrix()

glColor3f(0.0, 0.0, 0.0)

glScalef(car\_length / 4.84, car\_height / 2, car\_width / 5)

glTranslatef(car\_length / 2, car\_height - 0.7, -0.6)

glutSolidCube(1.0)

glPopMatrix()

glPushMatrix()

glColor3f(0.0, 0.0, 0.0)

glScalef(car\_length / 4.84, car\_height / 2, car\_width / 5)

glTranslatef(car\_length / 2, car\_height - 0.7, 0.6)

glutSolidCube(1.0)

glPopMatrix()

def draw\_wheel():

glPushMatrix()

# Wheel

glColor3f(0.2, 0.2, 0.2) # Dark gray color

glutSolidTorus(wheel\_width, wheel\_radius, 9, 9)

glPopMatrix()

def draw\_car():

# Draw car body

glPushMatrix()

glTranslatef(0, wheel\_radius + car\_height / 2, 0)

draw\_car\_body()

glPopMatrix()

# Draw trapezoidal cabin

glPushMatrix()

glTranslatef(-0.5, wheel\_radius, 0)

glScalef(car\_length - 1.7, car\_height / 1.04, car\_width / 2.3)

draw\_trapezoidal\_cabin(second\_color[0], second\_color[1], second\_color[2])

glPopMatrix()

glPushMatrix()

glTranslatef(-0.52, wheel\_radius, 0)

glScalef(car\_length - 1.65, car\_height / 1.04, car\_width / 2.4)

draw\_trapezoidal\_cabin(0, 0, 0)

glPopMatrix()

glPushMatrix()

glTranslatef(-0.5, wheel\_radius, 0)

glScalef(car\_length - 2, car\_height / 1.04, car\_width / 2.2)

draw\_trapezoidal\_cabin(0, 0, 0)

glPopMatrix()

# Draw wheels and wheel arches

for dx in [-car\_length / 2.5 + wheel\_radius, car\_length / 2.5 - wheel\_radius]:

for dz in [-car\_width / 1.8 + wheel\_radius, car\_width / 1.8 - wheel\_radius]:

glPushMatrix()

glColor3f(0.8, 0.8, 0.8)

glTranslatef(dx, wheel\_radius, dz \* 1.33)

gluDisk(gluNewQuadric(), 0, wheel\_radius, 30, 1)

glPopMatrix()

glPushMatrix()

glTranslatef(dx, wheel\_radius, dz)

draw\_wheel()

glTranslatef(0, wheel\_radius, 0)

glPopMatrix()

def display():

global angle\_x, angle\_y

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT)

glLoadIdentity()

gluLookAt(10.0, 5.0, 10.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)

# Apply rotations

glRotatef(angle\_y, 0, 1, 0)

glRotatef(angle\_x, 1, 0, 0)

# Draw car

draw\_car()

glutSwapBuffers()

def keyboard(key, x, y):

global angle\_x, angle\_y

if key == b'\x1b': # Escape key

sys.exit()

elif key == GLUT\_KEY\_UP:

angle\_x += 5

elif key == GLUT\_KEY\_DOWN:

angle\_x -= 5

elif key == GLUT\_KEY\_LEFT:

angle\_y -= 5

elif key == GLUT\_KEY\_RIGHT:

angle\_y += 5

glutPostRedisplay()

def init():

glEnable(GL\_DEPTH\_TEST)

glClearColor(1.0, 1.0, 1.0, 1.0)

glMatrixMode(GL\_PROJECTION)

glLoadIdentity()

gluPerspective(45, 1.0, 0.1, 100.0)

glMatrixMode(GL\_MODELVIEW)

def main():

glutInit(sys.argv)

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH)

glutInitWindowSize(800, 600)

glutCreateWindow(b"3D Car Animation")

init()

glutDisplayFunc(display)

glutSpecialFunc(keyboard)

glutMainLoop()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**3.6. Аналіз результатів відлагодження та верифікації результатів роботи програми.**

Результати відлагодження та тестування довели працездатність розробленого коду.

Верифікація функціоналу програмного коду, порівняння отриманих результатів з технічними умовами завдання на лабораторну роботу доводять, що завдання виконано у повному обсязі.

**IV. Висновки.**

У ході виконання лабораторної роботи я успішно створила динамічну модель руху легкових автомобілів з використанням технологій Computer Vision та бібліотеки OpenGL. Робота мала на меті формування модельного Dataset для навчання нейронних мереж для розпізнавання заданих об'єктів. Це дозволяє генерувати велику кількість зображень для навчання нейронних мереж, що можуть бути використані для різних задач Computer Vision, таких як розпізнавання та класифікація об'єктів.

Завдяки виконаній роботі я здобула практичний досвід використання бібліотек OpenGL та GLUT для створення тривимірних моделей та їх анімації. Отримані результати можуть бути використані для подальших досліджень у галузі Computer Vision, зокрема для навчання та тестування нейронних мереж.

Виконала: студент Яковенко Д.О.