

КПІ ім. Ігоря Сікорського
Інститут прикладного системного аналізу
Кафедра Системного проектування

Лабораторна робота №1
з дисципліни «Комп'ютерна графіка»

«Побудова двовимірної графіки»

Виконав:
Студент групи
ДА-81
ННК «ІПСА»
Дзюбчик
Олександр
Варіант № 8

Київ – 2020

Мета роботи: отримати навички створення графічних програм. Ознайомитись з можливостями OpenGL або обрати іншу графічну бібліотеку.

Завдання:

- 1) Ознайомитися з принципами побудови двовимірної системи координат.
- 2) Використовуючи обрану графічну бібліотеку, на основі примітивів зобразити істоту за варіантом з таблиці
- 3) Використовуючи бібліотеку що відповідає за системний рівень операцій вводу-виводу - реалізувати рух істоти у заданому векторі (див. таблицю).
Управляючі клавіші - ADWS
- 4) Розібратися з принципами роботи функцій:

glViewport,

glMatrixMode,

glLoadIdentity,

glOrtho,

glClearColor,

glClear,

glColor3ub,

glBegin,

glEnd,

glutInit;

glutInitDisplayMode;

glutInitWindowSize;

glutCreateWindow;

glutDisplayFunc;

glutReshapeFunc;

glutKeyboardFunc;

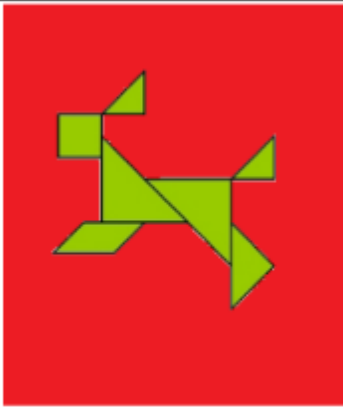
glutMainLoop;

glutMouseFunc,

glutSpecialFunc,

glutIdleFunc,

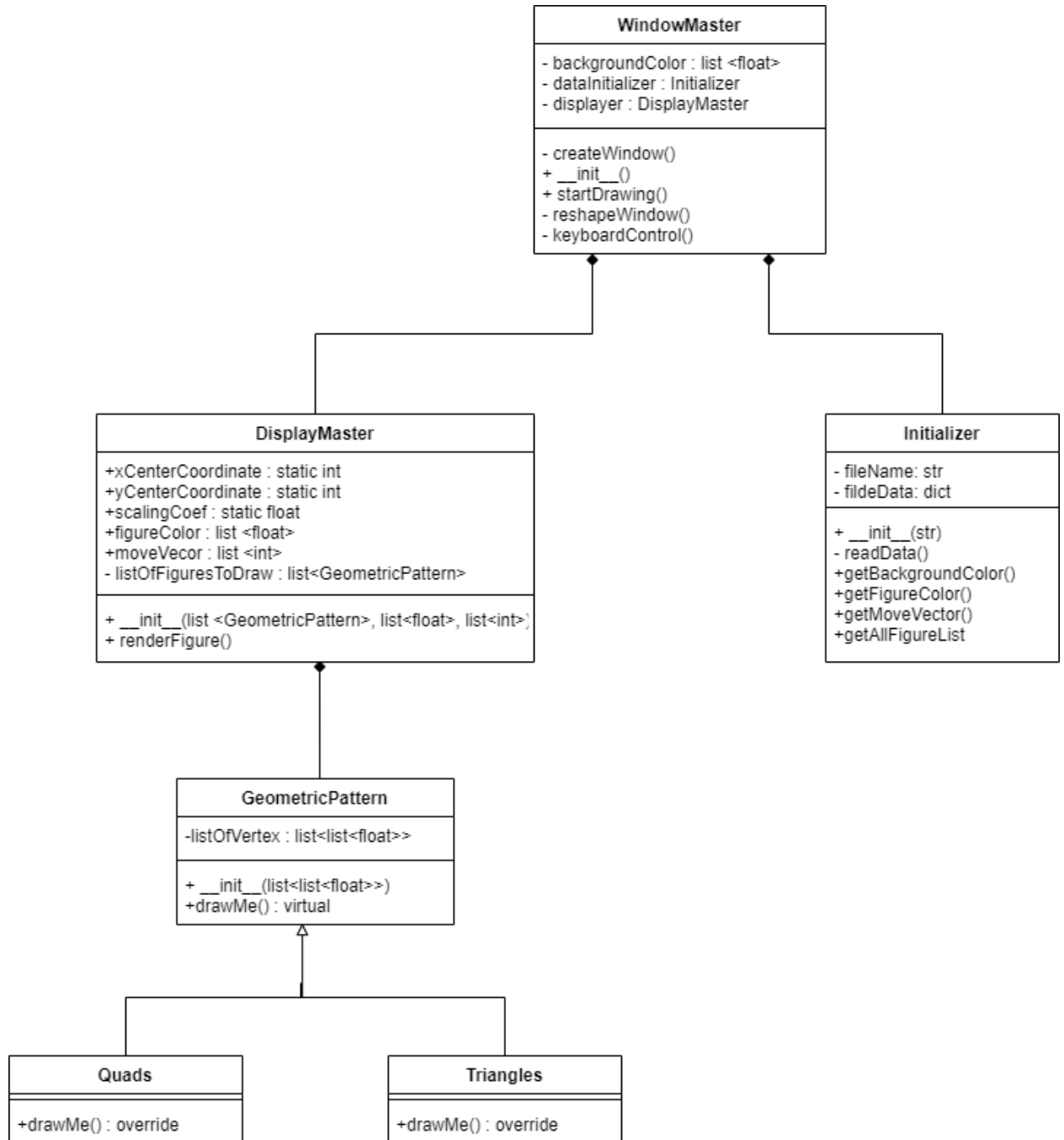
5) Знайти чим представлений аналогічний функціонал(з п.4) у обраній вами графічній бібліотеці

8		(1,-1)
---	--	--------

Опис обраної графічної бібліотеки

Для виконання лабораторної роботи було обрано OpenGL. Вибір зумовлено наявністю певного досвіду використання бібліотеки та її платформонезалежністю, оскільки на базовому рівні OpenGL – специфікація, на основі якої може бути створена реалізація (бібліотека) на різних мовах програмування. Програмний код для даної роботи написано на мові Python, відповідно з використанням PyOpenGL.

UML діаграма проекту



Опис роботи програми

Клас **WindowMaster** відповідає за:

- Створення вікна – функція *create_window()*, яка викликається у конструкторі.
- Запуск основного циклу – функція *start_drawing()*, викликається для об'єкту класу у головній функції програми.
- Обробку вікна – функції *reshape_window()* (зміна розмірів вікна) і *keyboard_control()* (керуванням рисунком за допомогою клавіатури)

У конструкторі класу **WindowMaster** створюються об'єкти класів **Initializer** і **DisplayMaster**.

Клас **Initializer** потрібен для зчитування даних з JSON файлу і передачу їх класам **WindowMaster** та **DisplayMaster**.

У конструктор **Initializer** передається назва файлу. Метод *read_data()* зчитує дані з файлу і зберігає їх у полі *file_data* як структуру «словник». Методи *get_move_vector()*, *get_figure_color()*, *get_background_color()* і їм подібні повертають значення отримане зі словнику за відповідним ключем. Метод *get_all_figure_list()* повертає список об'єктів класів, що наслідуються від **GeometricPattern**.

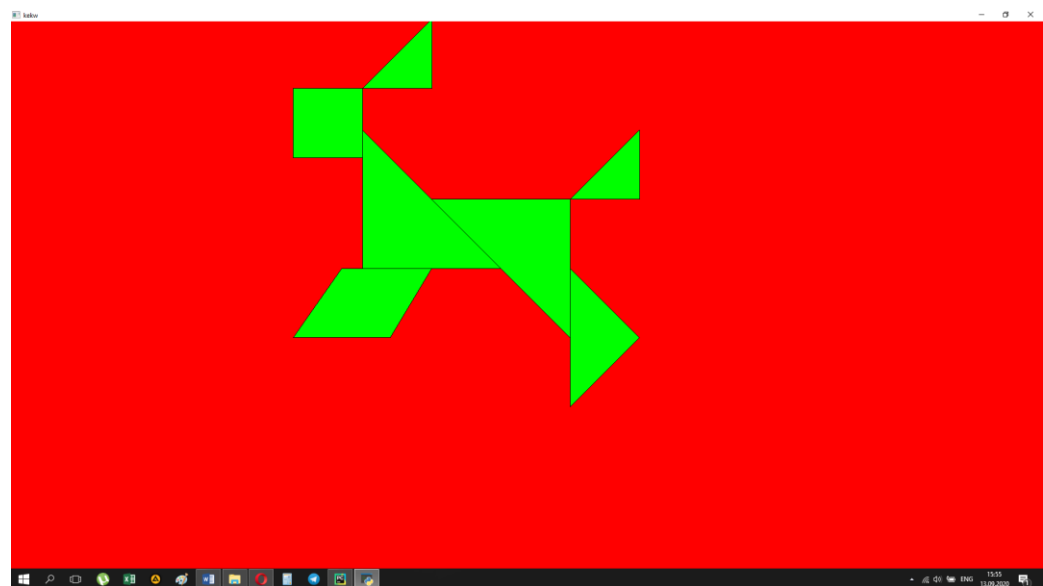
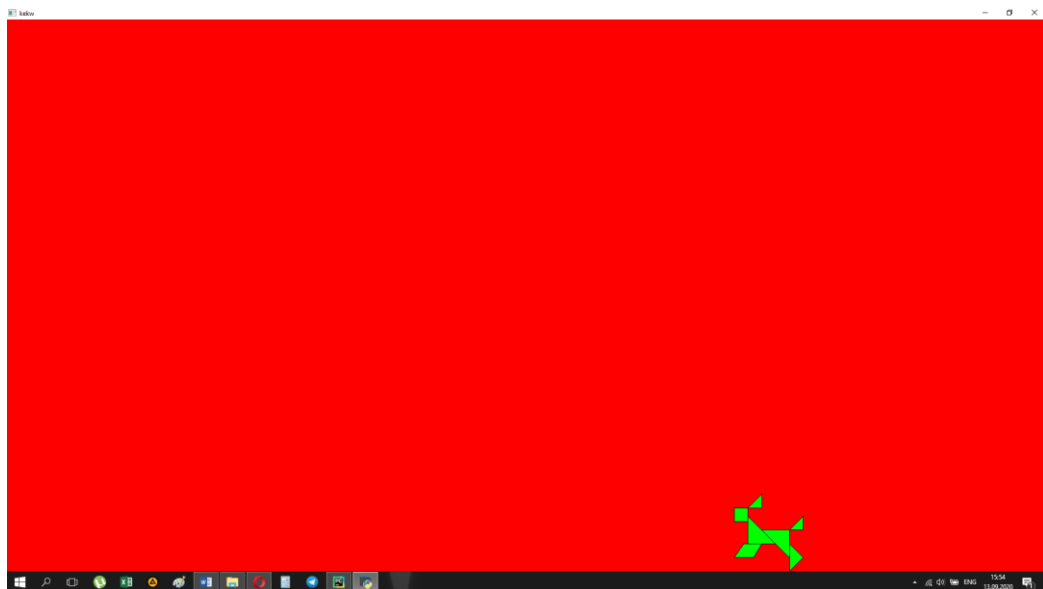
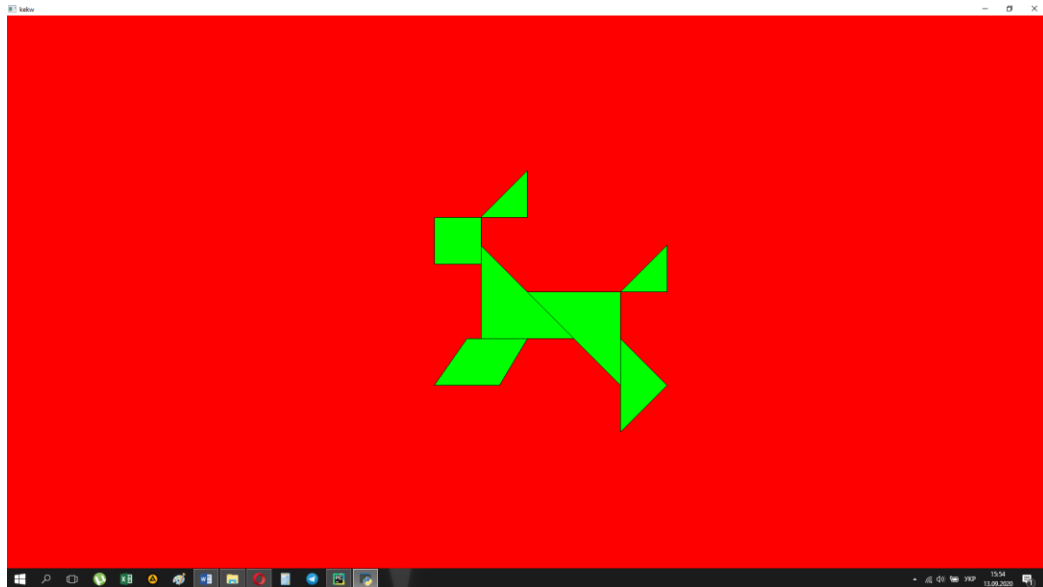
Клас **DisplayMaster** відповідає безпосередньо за створення зображення у вікні. У конструктор приймає наступні параметри: список об'єктів класів геометричних фігур, колір фігури, вектор руху рисунку. Поля *x_center_coordinate* і *y_center_coordinate* потрібні для можливості рухати рисунок, а *scaling_coef* для масштабування. Метод *render_figure()* відображає кожен об'єкт зі списку.

Клас **GeometricPattern** – абстрактний клас. У якому визначений конструктор, що приймає список точок фігури. Метод *draw_me()* – віртуальний.

Класи **Quads** і **Triangles** наслідують від **GeometricPattern** та перевизначають метод *draw_me()* для побудови на екрані чотирикутників і трикутників відповідно.

Я обрав таку структуру програми, оскільки вона надає можливість легко додати новий тип фігур, потрібно лише додати новий клас, що наслідується від **GeometricPattern** та модифікувати метод *get_all_figure_list()* класу **Initializer**.

Результат роботи програми



Висновок

В результаті виконання лабораторної роботи було створено програму, що на основі даних з JSon файлу, будує зображення на основі примітивів. Вивчено основні функції бібліотеки PyOpenGL для побудови двовимірного зображення.

Лістинг програми

GeometricsPatterns.py:

```
from OpenGL.GL import glBegin, glEnd, glVertex2f, glColor3ub, GL_TRIANGLES, GL_LINE_LOOP, GL_QUADS

from DisplayMaster import DisplayMaster

class GeometricPattern:

    def __init__(self, listOfVertex):
        self.listOfVertex = listOfVertex

    def draw_me(self):
        pass

class Triangles(GeometricPattern):

    def draw_me(self):
        #figure color
        glColor3ub(DisplayMaster.figure_color[0], DisplayMaster.figure_color[1], DisplayMaster.figure_color[2])
        #drawing figure
        glBegin(GL_TRIANGLES)
        for coords in self.listOfVertex:
            glVertex2f(DisplayMaster.x_center_coordinate + coords[0] * DisplayMaster.scaling_coef, DisplayMaster.y_center_coordinate + coords[1] * DisplayMaster.scaling_coef)
        glEnd()

        #border color
        glColor3ub(0, 0, 0)
        #drawing border
        glBegin(GL_LINE_LOOP)
        for coords in self.listOfVertex:
            glVertex2f(DisplayMaster.x_center_coordinate + coords[0] * DisplayMaster.scaling_coef, DisplayMaster.y_center_coordinate + coords[1] * DisplayMaster.scaling_coef)
        glEnd()

class Quads(GeometricPattern):

    def draw_me(self):
        #figure color
        glColor3ub(DisplayMaster.figure_color[0], DisplayMaster.figure_color[1], DisplayMaster.figure_color[2])
        #drawing figure
        glBegin(GL_QUADS)
        for coords in self.listOfVertex:
```

```

        glVertex2f(DisplayMaster.x_center_coordinate + coords[0] * DisplayMaster.scaling_coef, DisplayMaster.y_center_coordinate + coords[1] * DisplayMaster.scaling_coef)
    glEnd()

    #border color
    glColor3ub(0, 0, 0)
    #drawing border
    glBegin(GL_LINE_LOOP)
    for coords in self.listOfVertex:
        glVertex2f(DisplayMaster.x_center_coordinate + coords[0] * DisplayMaster.scaling_coef, DisplayMaster.y_center_coordinate + coords[1] * DisplayMaster.scaling_coef)
    glEnd()

```

Initializer.py:

```

import json
from GeometricsPatterns import *

class Initializer:
    """
        A class for reading and storing data from json file.
        ...
        Attributes
        -----
        file_name : str
            name of the source file
        file_data : dict
            key - data type name, value - data stored as list(moving vector, color etc.)
            or as list of list (figure's points)
    """

    def __init__(self, file_name):
        self.file_name = file_name
        self.read_data()

    def read_data(self):
        with open(self.file_name) as jsonFile:
            file_content = jsonFile.read()
            self.file_data = json.loads(file_content)

    def get_background_color(self):
        return self.file_data['backgroundColor']

    def get_figure_color(self):
        return self.file_data['figureColor']

    def get_move_vector(self):
        return self.file_data['vector']

```

```

def get_all_figure_list(self):
    figures_to_draw = []
    all_figure_list = self.file_data['figure']

    for list_of_vertex in all_figure_list['trinagle']:
        figures_to_draw.append(Triangles(list_of_vertex))

    for list_of_vertex in all_figure_list['quad']:
        figures_to_draw.append(Quads(list_of_vertex))

    return figures_to_draw

```

DisplayMaster.py:

```

from OpenGL.GL import glClear, GL_COLOR_BUFFER_BIT, glFlush

class DisplayMaster:
    """
        A class for displaying figure.
        ...
        Attributes
        -----
        x_center_coordinate, y_center_coordinate: int, int
            coordinates of figure center (used for moving). Both are equal zero by default.

        scaling_coef: float
            equal 1 by default, increase/decrease by 50% each time changed

        figure_color: list
            figure color stored as rgb unsigned byte

        move_vector: list
            two coordinates (-1 or 1) that defines figure moving direction

        list_of_figures_to_draw: list
            stores class-object of geometric figures to draw
    """

    x_center_coordinate = 0
    y_center_coordinate = 0

    scaling_coef = 1

    figure_color = []
    move_vector = []

    def __init__(self, list_of_figures_to_draw, figure_color, move_vector):
        self.list_of_figures_to_draw = list_of_figures_to_draw
        DisplayMaster.figure_color = figure_color
        DisplayMaster.move_vector = move_vector

```

```
#
def renderFigure(self):
    glClear(GL_COLOR_BUFFER_BIT)

    for figures in self.list_of_figures_to_draw:
        figures.draw_me()

    glFlush()
```

WindowMaster.py:

```
from OpenGL.GLUT import *
from OpenGL.GL import glClearColor, glViewport, glMatrixMode, glLoadIdentity, GL_
PROJECTION, GL_MODELVIEW
from OpenGL.GLU import gluOrtho2D

from Initializer import Initializer
from DisplayMaster import DisplayMaster

class WindowMaster:
    """
        A class for managing program window. It collects data, initializes window
, draws figure, manages keyboard actions.
        ...
        Attributes
        -----
        background_color: list
            background color stored as rgb unsigned byte

        data_initializer: Initializer
            object of class Initializer used to read and store input data

        displayer: DisplayMaster
            object of class DisplayMaster used to draw main figure
        ...
        Methods
        -----
        create_window():
            Create window, set background color and grid size

        start_drawing():
            Start main loop.

        reshape_window(width, height):
            Manage changing window's size
    """
    def create_window(self):
        glutInit(sys.argv)
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
        glutInitWindowSize(800, 800)
        glutInitWindowPosition(320, 100)
```

```

        glutCreateWindow("lab1")
        glClearColor(self.background_color[0], self.background_color[1], self.background_color[2], 1.0)
        gluOrtho2D(-20.0, 20.0, -20.0, 20.0)

    def __init__(self):
        self.data_initializer = Initializer('sw_templates.json')
        self.background_color = [color/255 for color in self.data_initializer.get_background_color()]
        self.create_window()
        self.displayer = DisplayMaster(self.data_initializer.get_all_figure_list(), self.data_initializer.get_figure_color(), self.data_initializer.get_move_vector())

    def startDrawing(self):
        glutDisplayFunc(self.displayer.renderFigure)
        glutReshapeFunc(self.reshapeWindow)
        glutKeyboardFunc(self.keyboardControl)
        glutMainLoop()

    def reshapeWindow(self, w, h):
        if h == 0:
            h = 1

        koef = float(w) / h
        glViewport(0, 0, w, h)

        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()

        if w > h:
            gluOrtho2D(-20.0 * koef, 20.0 * koef, -20.0, 20.0)
        else:
            gluOrtho2D(-20.0, 20.0, -20.0 / koef, 20.0 / koef)

        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()

    def keyboardControl(self, key, x1, y1):

        key = ord(key)

        if key == 100 or key == 119:
            DisplayMaster.x_center_coordinate = DisplayMaster.x_center_coordinate + DisplayMaster.move_vector[0]
            DisplayMaster.y_center_coordinate = DisplayMaster.y_center_coordinate + DisplayMaster.move_vector[1]
        if key == 97 or key == 115:
            DisplayMaster.x_center_coordinate = DisplayMaster.x_center_coordinate - DisplayMaster.move_vector[0]

```

```
        DisplayMaster.y_center_coordinate = DisplayMaster.y_center_coordinate  
- DisplayMaster.move_vector[1]  
    if key == 43:  
        DisplayMaster.scaling_coef = DisplayMaster.scaling_coef * 1.5  
    if key == 45:  
        DisplayMaster.scaling_coef = DisplayMaster.scaling_coef / 1.5  
    glutPostRedisplay()
```

main.py:

```
from WindowMaster import WindowMaster  
  
if __name__ == "__main__":  
    window = WindowMaster()  
    window.startDrawing()
```

Вміст JSon файлу

```
{ "backgroundColor": [255, 0, 0], "figureColor": [0, 255, 0], "vector": [1, -1], "figure": { "trinagle": [[[0, 0], [2, 0], [2, -2]], [[2, 0], [3, 0], [3, 1]], [[2, -1], [3, -2], [2, -3]], [[-1, 1], [1, -1], [-1, -1]], [[-1, 1.6], [0, 1.6], [0, 2.6]]], "quat": [[[-1.3, -1], [0, -1], [-0.6, -2], [-2, -2]], [[-1, 0.6], [-2, 0.6], [-2, 1.6], [-1, 1.6]]]}}
```