НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу

«Комп'ютерна графіка»

(назва кредитного модуля)

Звіт до лабораторної роботи 2

Виконав ст. гр ДА81 Переяславський С. К.

Дата 27.10.2020

Керівник Голубова І. А.

Київ – 2020

Зміст

Завдання

Мета роботи: отримати навички з побудови тривимірних зображень з базових примітивів та атомарних елементів.

Завдання:

1) Ознайомитися з принципами побудови тривимірної системи координат.

2) Побудувати сцену, що складається з трьох об'єктів: дві базові фігури і площину. Вважаємо, що центр площини - збігається з центром сцени, а фігури розташовані на рівній відстані по обом сторонам від неї.

2.1) У відповідності до свого варіанту відобразити на екрані базові бібліотечні об’єкти у вигляді каркасу та у вигляді суцільного об’єкту. Задати два різних кольори заповнення.

2.2) Створити поверхню з вершин за допомогою функцій glVertexPointer та glVertex3f, у відповідності до функції вашого варіанту

3) Використовуючи команду gluLookAt та функції обробки клавіш, створити рухому камеру для зображених об’єктів. (Камеру рухаємо навколо центру сцени - повний оборот в горизонтальній і вертикальній площині)

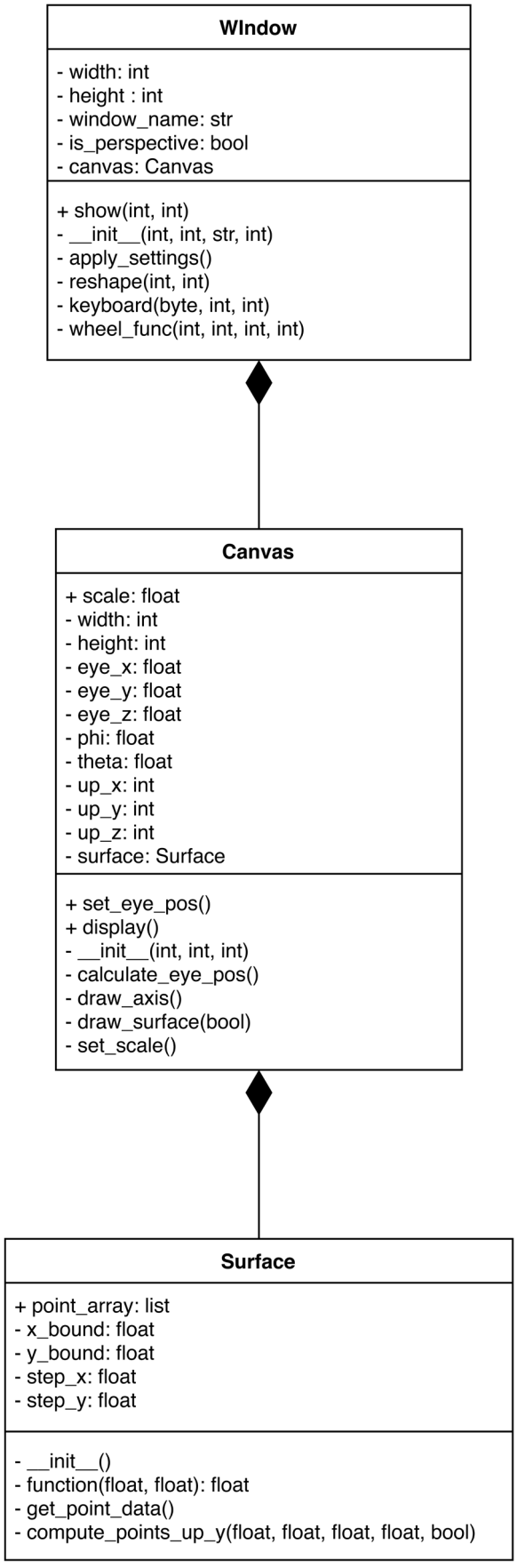
4) Отримати зображення об’єктів в ортогональній та перспективній проекціях використовуючи функції glOrtho, glFrustum, gluLookAt (перемикання між проекціями запрограмувати на натискання клавіш "P" та "O" відповідно). При цьому необхідно враховувати що матриці об’єктів та відображення мають бути обрані безпосередньо командою glMatrixMode, та попередньо нормовані функцією glLoadIdentity.

Індивідуальний варіант завдання:

Варіант	Примітиви контурні/зафарбовані	Функція для зображення
22	2/3	Z=sin(x)+cos(y)

Примітиви	
2	Куб
3	Конус

UML-діаграма класів



Короткий опис

За основу програми була взята ідея з попередньої лабораторної роботи. Така структура змогла забезпечити гнучкість модифікації програми та налаштувати її під задачу відображення 3D – об’єктів, а також зберегти час на етапі проектування.

Клас “Window” відповідає за ініціалізацію вікна та обробляє усі події які його стосуються (дії при: зміні розмірів вікна, згортанні/розгортанні вікна, перекритті його іншим вікном, натисканні клавіш W, S, A, D, O, P, spacebar, масштабування за допомогою колеса прокрутки). Тут визначений метод для первинного налаштування під час запуску програми. Клавіші W, S, A, D відповідають за рух камери навколо вертикальної та горизонтальної осей. Координати позиції камери обраховуються за допомогою сферичних координат. Власне, клавіші W, S, A, D відповідають за зміну (змінюють) значення кутів φ та θ. Клавіша W: зменшити кут θ на 5. Клавіша S: збільшити кут θ на 5. Клавіша A: зменшити кут φ на 5. Клавіша D: збільшити кут φ на 5. Клавіша spacebar дозволяє скинути налаштування (reset) де камера буде знаходитися на кінці осі Z (направлена на спостерігача) і можна буде спостерігати за площиною xOy під прямим кутом. Клавіші P та O відповідають за перехід до перспективної та до ортографічної проекції відповідно. При цьому відбувається перевірка на спробу перейти до ортографічної проекції, якщо така вже налаштована. Так само і для перспективної проекції.

Клас “Canvas” відповідає за відображення в області вікна. Екземпляр класу “Window” містить екземпляр класу “Canvas”, який не може існувати окремо. Тут визначені методи:

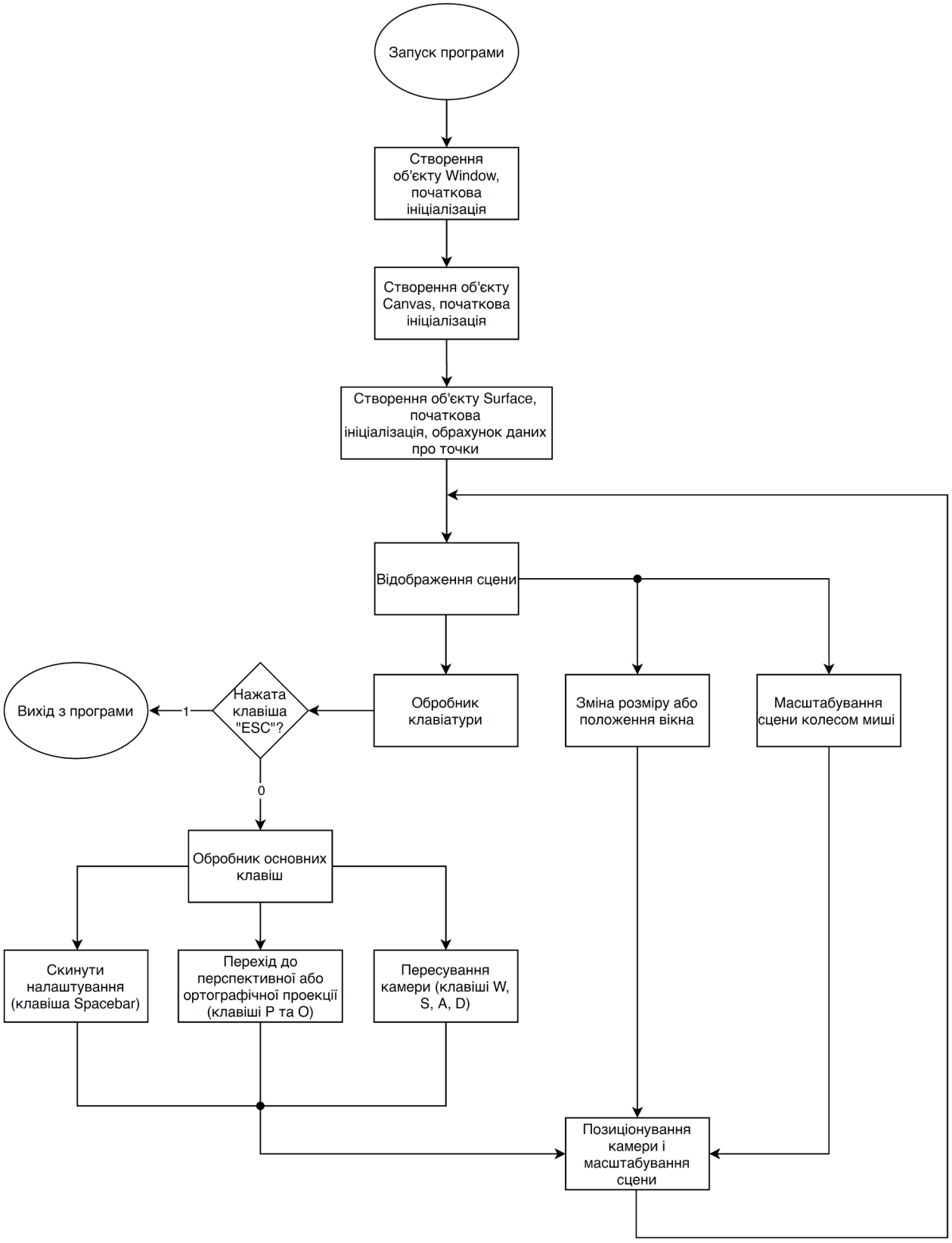
- обрахунку позиції камери;
- позиціонування камери;
- масштабування сцени;
- малювання осей координат;
- малювання поверхні;
- відображення об’єктів на сцені.

Метод draw_surface приймає булеве значення, яке говорить про те, як слід будувати поверхню: за допомогою графічних примітивів, явно прописуючи glBegin та glEnd з вказаним режимом GL_LINES, ітеруючи по всьому списку

вершин або за допомогою масиву вершин, який приймає на вхід кількість координат, які визначають позицію точки (в даному випадку - 3), тип даних, які містяться у списку вершин, власне список вершин. Після цього необхідно викликати процедуру glDrawArrays з режимом малювання (попередньо викликавши glEnableClientState(GL_VERTEX_ARRAY)), початковим індексом та списком вершин, по яким необхідно будувати графічні примітиви.

Клас “Surface” відповідає за надання даних про поверхню, яку необхідно відобразити у тривимірному просторі. Тут визначений методи обрахунку даних про точки, які належать поверхні. На вхід приймаються дані про бажані границі області, в межах якої буде будуватися поверхня, визначена у методі function, а також бажаний крок обрахунку. Варто зазначити, що у даній роботі поверхня будується за допомогою ліній, які в свою чергу утворюють сітку для більш помітного зображення нерівностей та перегинів поверхні. Такий підхід дає змогу будувати будь-яку поверхню: необхідно лише змінити цільову функцію для обрахунку координат та, за наявності обмежень на область визначення, змінити границі області, в межах якої буде будуватися поверхня.

Блок-схема роботи програми



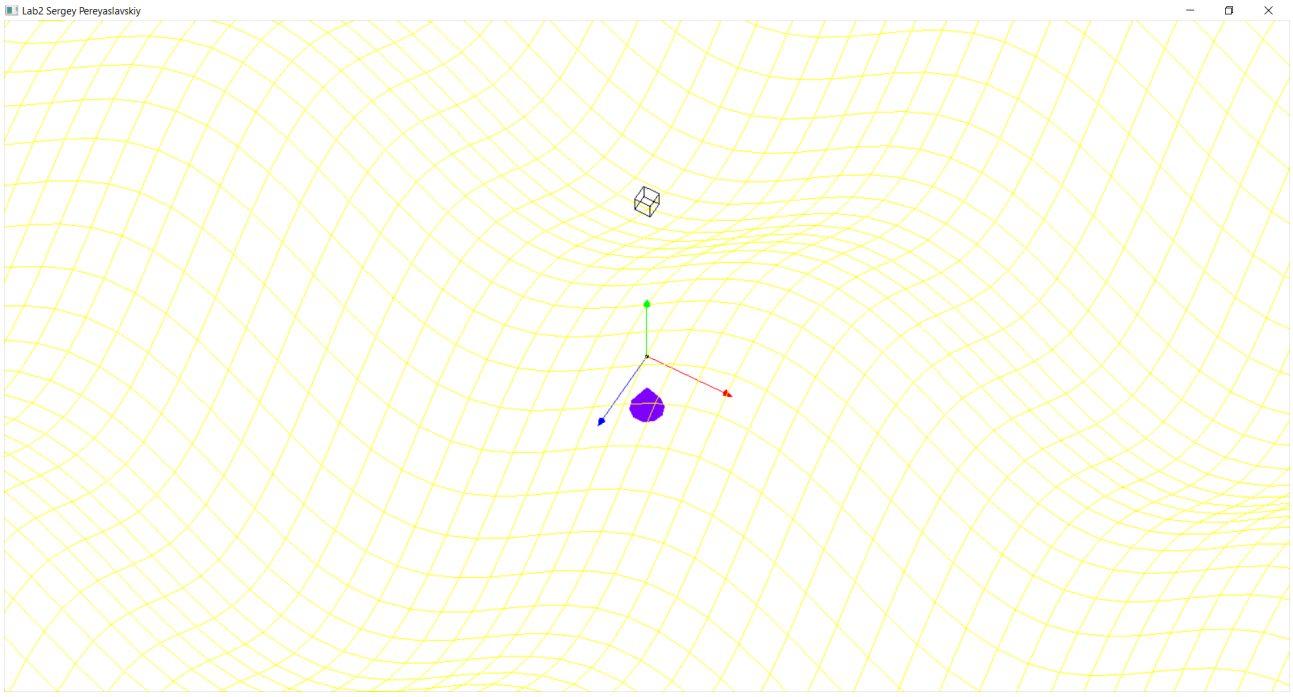
Висновки

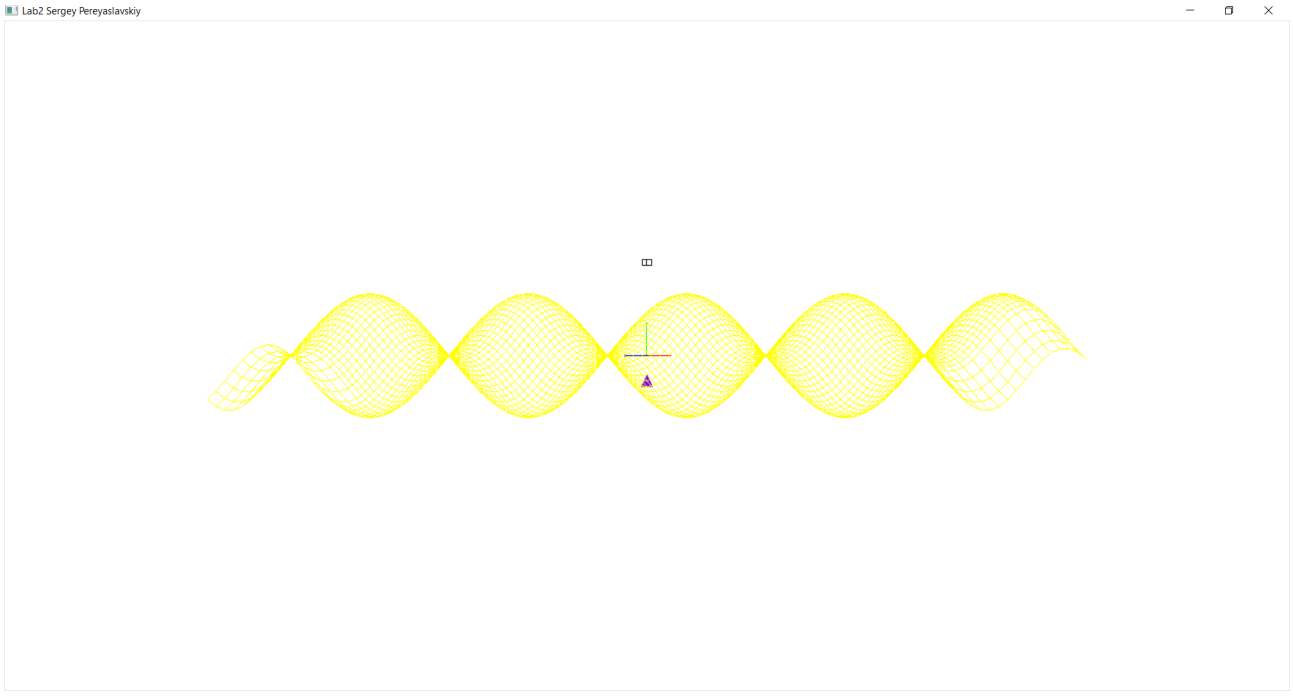
У ході виконання лабораторної роботи була розроблена програма відображення об’ємних об’єктів у тривимірному просторі. За основу була взята програма з попередньої л. р. Така структура змогла забезпечити гнучкість модифікації програми та налаштувати її під задачу відображення 3D – об’єктів, а також зберегти час на етапі проектування.

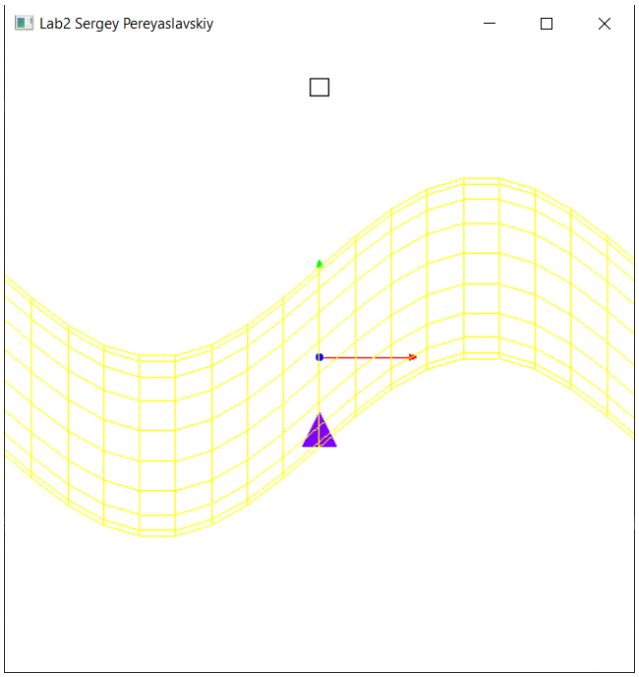
У просторі сцени відображаються фігури згідно індивідуального варіанту завдання, а також поверхня, функція якої також відповідає індивідуальному варіанту завдання. Конус був зображений у вигляді суцільного об’єкту, а куб – у вигляді каркасу. Дані фігури розташовуються на однаковій відстані від поверхні.

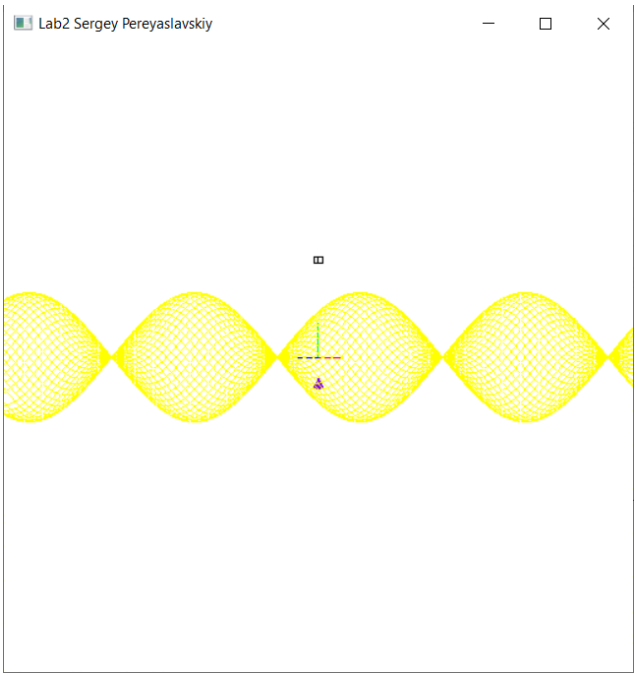
Був реалізований функціонал руху камери навколо горизонтальної та вертикальної осей. Був реалізований функціонал переходу до перспективної або ортографічної проекції та навпаки. Був реалізований скид налаштувань до стандартних (по клавіші пробіл). Був реалізований функціонал масштабування сцени (взагалі кажучи – масштабування відстані від камери до початку координат).

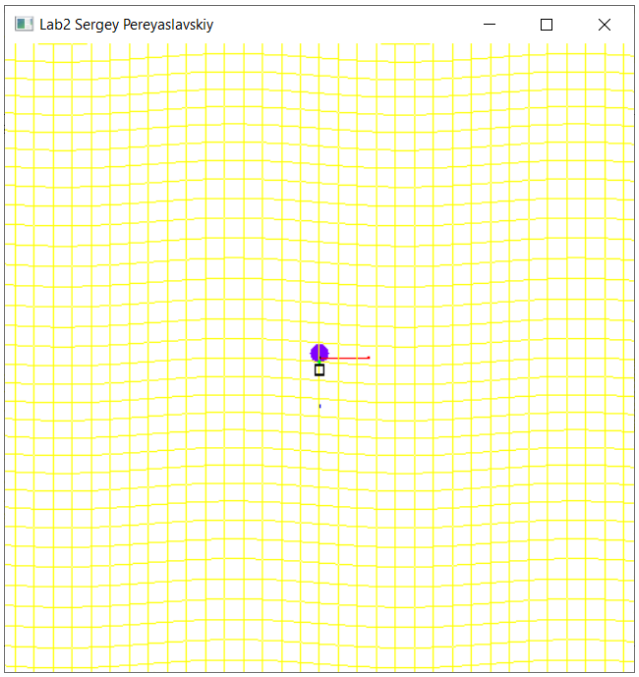
Скріншоти програми

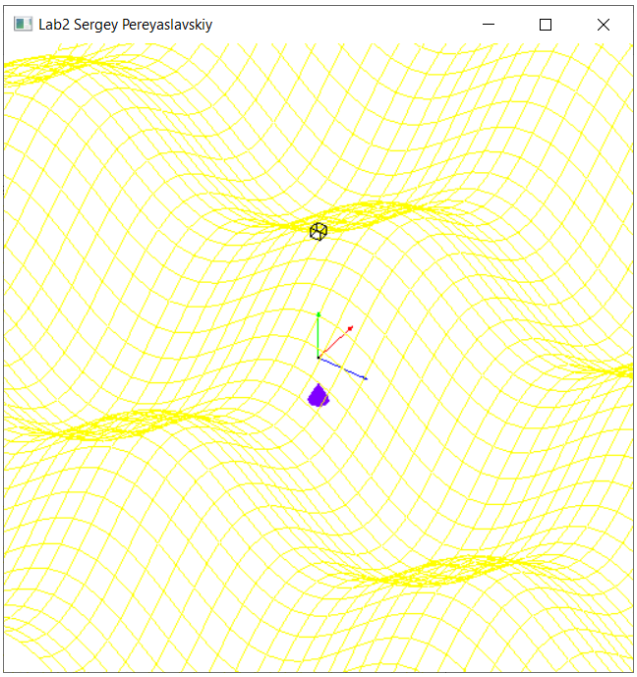





Lab2 Sergey Pereyaslavskiy


Lab2 Sergey Pereyaslavskiy


Lab2 Sergey Pereyaslavskiy


Lab2 Sergey Pereyaslavskiy

Window.py

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
import sys

from Canvas import Canvas


class Window:

    def __init__(self, width, height, window_name):
        self.width = width
        self.height = height
        self.window_name = window_name
        self.is_perspective = True
        self.canvas = Canvas(width, height)

    def apply_settings(self):
        glMatrixMode(GL_PROJECTION)
        gluPerspective(120, 1, 1, 1000)
        glOrtho(-self.width / 2, self.width / 2, -self.height /
2, self.height / 2, -1000, 1000)
        glMatrixMode(GL_MODELVIEW)
        glEnable(GL_DEPTH_TEST)
        glClearColor(1, 1, 1, 1)
        self.canvas.set_eye_pos()
        glViewport(0, 0, self.width, self.height)

    def show(self, pos_x=0, pos_y=0):
        glutInit(sys.argv)
        glutInitDisplayMode(GLUT_RGBA)
        glutInitWindowSize(self.width, self.height)
        glutInitWindowPosition(pos_x, pos_y)
        glutCreateWindow(self.window_name)
        self.apply_settings()
        glutDisplayFunc(self.canvas.display)
        glutKeyboardFunc(self.keyboard)
        glutReshapeFunc(self.reshape)
        glutMouseWheelFunc(self.wheel_func)
        glutMainLoop()

    def reshape(self, w, h):
        self.width = self.canvas.width = w
        self.height = self.canvas.height = h
```

```
        glLoadIdentity()
        glMatrixMode(GL_PROJECTION)
        if self.is_perspective:
            gluPerspective(60, 1, 1, 1000)
        glLoadIdentity()
        glOrtho(-self.width / 2, self.width / 2, -self.height /
2, self.height / 2, -1000, 1000)
        glMatrixMode(GL_MODELVIEW)
        self.canvas.set_eye_pos()
        glViewport(0, 0, self.width, self.height)

    def keyboard(self, key, x, y):
        pressed_key = key.decode("utf-8").lower()
        if pressed_key == chr(27):
            sys.exit(0)
        elif pressed_key == 'p' and self.is_perspective is False:
            self.is_perspective = True
            self.reshape(self.width, self.height)
        elif pressed_key == 'o' and self.is_perspective is True:
            self.is_perspective = False
            self.reshape(self.width, self.height)
        if self.is_perspective:
            glLoadIdentity()
            if pressed_key == ' ':
                self.canvas.phi = 0.0
                self.canvas.theta = 90.0
                self.canvas.up_x = self.canvas.up_z = 0
                self.canvas.up_y = 1
            elif pressed_key == 'w':
                self.canvas.theta = (self.canvas.theta - 5) % 360
                if self.canvas.theta == 0:
                    self.canvas.theta = -5 % 360
            elif pressed_key == 's':
                self.canvas.theta = (self.canvas.theta + 5) % 360
                if self.canvas.theta == 0:
                    self.canvas.theta = 5 % 360
            elif pressed_key == 'a':
                self.canvas.phi = (self.canvas.phi - 5) % 360
            elif pressed_key == 'd':
                self.canvas.phi = (self.canvas.phi + 5) % 360

            if 0 < self.canvas.theta <= 180:
                self.canvas.up_y = 1
            else:
```

Window.py

```
                self.canvas.up_y = -1
            self.canvas.set_eye_pos()
        glutPostRedisplay()

    def wheel_func(self, wheel, direction, x, y):
        if self.is_perspective:
            if direction > 0:
                self.canvas.scale *= 1.1
                glScale(1.1, 1.1, 1.1)
            elif direction < 0:
                self.canvas.scale *= 0.9
                glScale(0.9, 0.9, 0.9)
            glutPostRedisplay()
```

Canvas.py

```
from OpenGL.GL import *
from OpenGL.GLU import *
from OpenGL.GLUT import *
from math import sin, cos, pi

from Surface import Surface


class Canvas:

    def __init__(self, width, height):
        self.width = width
        self.height = height
        self.eye_x = self.eye_y = self.eye_z = 0
        self.phi = 0.0
        self.theta = 90.0
        self.up_x = 0
        self.up_y = 1
        self.up_z = 0
        self.scale = 1
        self.surface = Surface()

    def calculate_eye_pos(self):
        self.eye_z = sin(self.theta * pi / 180) * cos(self.phi *
pi / 180)
        self.eye_x = sin(self.theta * pi / 180) * sin(self.phi *
pi / 180)
        self.eye_y = cos(self.theta * pi / 180)

    def set_eye_pos(self):
        self.calculate_eye_pos()
        gluLookAt(self.eye_x, self.eye_y, self.eye_z, 0, 0, 0,
self.up_x, self.up_y, self.up_z)
        self.set_scale()

    def draw_axis(self):
        # x-axis and the arrow
        glPushMatrix()
        glRotate(90, 0, 1, 0)
        glTranslated(0, 0, 50)
        glColor3f(1, 0, 0)
        glutSolidCone(2, 5, 100, 100)
        glPopMatrix()
        glBegin(GL_LINE_LOOP)
```

```
        glVertex3f(0, 0, 0)
        glVertex3f(50, 0, 0)
        glEnd()

        # y-axis and the arrow
        glPushMatrix()
        glRotate(-90, 1, 0, 0)
        glTranslated(0, 0, 50)
        glColor3f(0, 1, 0)
        glutSolidCone(2, 5, 100, 100)
        glPopMatrix()
        glBegin(GL_LINE_LOOP)
        glVertex3f(0, 0, 0)
        glVertex3f(0, 50, 0)
        glEnd()

        # z-axis and the arrow
        glPushMatrix()
        glTranslated(0, 0, 50)
        glColor3f(0, 0, 1)
        glutSolidCone(2, 5, 100, 100)
        glPopMatrix()

        glBegin(GL_LINE_LOOP)
        glVertex3f(0, 0, 0)
        glVertex3f(0, 0, 50)
        glEnd()

    def draw_surface(self, vertex_array=True):
        if vertex_array:
            glEnableClientState(GL_VERTEX_ARRAY)
            glVertexPointer(3, GL_FLOAT, 0, self.surface.
point_array)
            glDrawArrays(GL_LINES, 0, len(self.surface.
point_array))
            glDisableClientState(GL_VERTEX_ARRAY)
        else:
            glBegin(GL_LINES)
            for x, y, z in self.surface.point_array:
                glVertex3f(x, y, z)
            glEnd()

    def display(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
```

```
        glColor3f(0, 0, 0)
        glutSolidSphere(1, 10, 10)

        glPushMatrix()
        glTranslated(0, 150, 0)
        glColor3f(0, 0, 0)
        glutWireCube(10)
        glPopMatrix()

        glPushMatrix()
        glTranslated(0, -50, 0)
        glRotate(-90, 1, 0, 0)
        glColor3f(0.5, 0, 1)
        glutSolidCone(10, 20, 10, 10)
        glPopMatrix()

        self.draw_axis()
        glColor3f(1, 1, 0)
        self.draw_surface(vertex_array=True)
        glFlush()

    def set_scale(self):
        glScale(self.scale, self.scale, self.scale)
```

Surface.py

```
from math import sin, cos, pi


class Surface:

    def __init__(self):
        self.x_bound = 500.0
        self.y_bound = 500.0
        self.step_x = 20.0
        self.step_y = 20.0
        self.point_array = []
        self.get_point_data()

    def function(self, x, y):
        return sin(x * pi / 180) + cos(y * pi / 180)

    def get_point_data(self):
        self.compute_points_up_y(self.x_bound, self.y_bound, self
.step_x, self.step_y, swap=False)
        self.compute_points_up_y(self.y_bound, self.x_bound, self
.step_y, self.step_x, swap=True)

    def compute_points_up_y(self, x_bound, z_bound, step_x,
step_z, swap=False):
        x = -x_bound
        while x != x_bound + step_x:
            z = -z_bound
            while z != z_bound:
                z += step_z
                if not swap:
                    y = self.function(x, z)
                    self.point_array.append([x, y, z])
                else:
                    y = self.function(z, x)
                    self.point_array.append([z, y, x])
            x += step_x
        x = -x_bound
        while x != x_bound + step_x:
            z = z_bound
            while z != -z_bound:
                z -= step_z
                if not swap:
                    y = self.function(x, z)
                    self.point_array.append([x, y, z])
```

```
                else:
                    y = self.function(z, x)
                    self.point_array.append([z, y, x])
            x += step_x
```