МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине Компьютерная графика

Тема: «Формирование различных поверхностей с использованием ее пространственного разворота и ортогонального проецирования на плоскость при ее визуализации (выводе на экран дисплея)»

Студенты гр. 1307 Грунская Н.Д.
Тростин М.Ю.
Голубев М.А.

Преподаватель _____ Матвеева И.В.

Санкт-Петербург
2024

Цель работы:
Практическое закрепление теоретических знаний о формировании различных поверхностей с использованием ее пространственного разворота и ортогонального проецирования на плоскость

Постановка задачи:
Сформировать билинейную поверхность на основе произвольного задания ее четырех угловых точек. Обеспечить ее поворот относительно осей X и Y

Краткая теоретическая информация:
Билинейные поверхности - трёхмерные поверхности, задающиеся в пространстве 4-х угловых точек поверхности
Тогда уравнение билинейной поверзности представляется как:
$\overline{Q}(u,w) = \overline{P_{00}}(1-u)(1-w) + \overline{P_{01}}(1-u)w + \overline{P_{10}}u(1-w) + \overline{P_{11}}uw$

Для поворота вокруг осей X и Y необходимо применить поворот на один и тот же угол для каждой из четырёх задающих точек

Реализация алгоритма:
Функция поворота точки вокруг оси X
```
def rotate_x(self, num, angle):
    x = self.points[num].coordinates[0]
    y = self.points[num].coordinates[1] * math.cos(angle)
- self.points[num].coordinates[2] * math.sin(angle)
    z = self.points[num].coordinates[1] * math.sin(angle)
+ self.points[num].coordinates[2] * math.cos(angle)
    return [x, y, z]
```

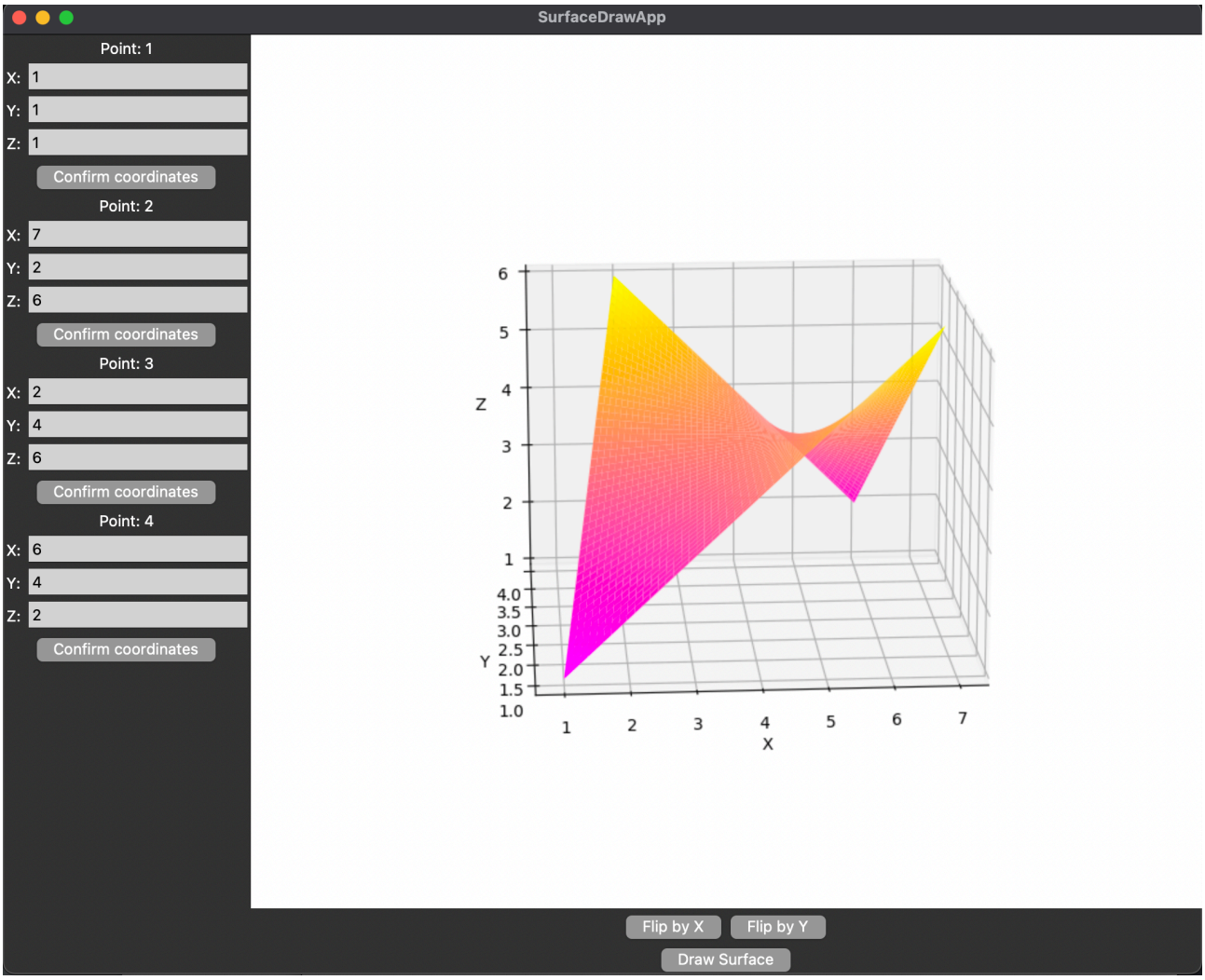
Функция разворота фигуры вокруг оси X
```
def flip_by_x(self):
    angle = math.pi
    for i in range(4):
        self.points[i].coordinates = self.rotate_x(i,
angle)
    self.draw_curve()
```

Функция поворота точки вокруг оси Y
```
def rotate_y(self, num, angle):
    x = self.points[num].coordinates[0] * math.cos(angle)
+ self.points[num].coordinates[2] * math.sin(angle)
    y = self.points[num].coordinates[1]
    z = -self.points[num].coordinates[0] *
math.sin(angle) + self.points[num].coordinates[2] *
math.cos(angle)
    return [x, y, z]
```

Функция разворота фигуры вокруг оси Y

```
def flip_by_y(self):
    angle = math.pi
    for i in range(4):
        self.points[i].coordinates = self.rotate_y(i,
angle)
    self.draw_curve()
```

Пример работы приложения:



Выводы:

В ходе выполнения работы были практически закреплены теоретические знания о формировании различных поверхностей с использованием ее пространственного разворота и ортогонального проецирования на плоскость

Приложение
Ссылка на видео:

Исходный код программы:

```
import math
import tkinter as tk
from tkinter import messagebox
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg

import math
import tkinter as tk
from tkinter import messagebox
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import
FigureCanvasTkAgg

global frequency
frequency = 100


class Point(tk.Frame):
    def __init__(self, parent, number):
        super().__init__(parent)

        self.coordinates = [0, 0, 0]
        self.number = number
        self.confirm_button = tk.Button
        self.point_frame = tk.Frame(self, width=100,
height=50)
        self.point_frame.pack()
        new_point_label = tk.Label(self.point_frame,
text="Point: " + self.number)
        new_point_label.pack()

        # literally small x with his little labels and
entry's in his little frame
        self.x_frame = tk.Frame(self, width=50,
height=25)
        x_label = tk.Label(self.x_frame, text="X:")
        x_label.pack(side="left")
        self.x_entry = tk.Entry(self.x_frame)
```

```
        self.x_entry.pack(side='right')
        self.x_frame.pack()

        # stupid y (I hate him)
        self.y_frame = tk.Frame(self, width=50,
height=25)
        y_label = tk.Label(self.y_frame, text="Y:")
        y_label.pack(side="left")
        self.y_entry = tk.Entry(self.y_frame)
        self.y_entry.pack(side='right')
        self.y_frame.pack()

        # ZZZZZZZZZ do I even have to sa anything LOL
        self.z_frame = tk.Frame(self, width=50,
height=25)
        z_label = tk.Label(self.z_frame, text="Z:")
        z_label.pack(side="left")
        self.z_entry = tk.Entry(self.z_frame)
        self.z_entry.pack(side='right')
        self.z_frame.pack()

        self.confirm_button = tk.Button(self,
text="Confirm coordinates",
command=self.confirm_coordinates)
        self.confirm_button.pack()

    def confirm_coordinates(self):
        self.coordinates[0] = int(self.x_entry.get())
        self.coordinates[1] = int(self.y_entry.get())
        self.coordinates[2] = int(self.z_entry.get())
        self.x_entry.config(bg="lightgrey", fg="black")
        self.y_entry.config(bg="lightgrey", fg="black")
        self.z_entry.config(bg="lightgrey", fg="black")


class SurfaceApp(tk.Tk):
    def __init__(self):
        super().__init__()

        self.fig = None
        self.ax = None
        self.canvas = None

        self.title("SurfaceDrawApp")
        self.geometry("1020x800")
```

```
        self.points = []

        self.menu_frame = tk.Frame(self, width=200,
height=600)
        self.menu_frame.pack(side="left", fill="y")

        self.draw_surface_button = tk.Button(self,
text="Draw Surface", command=self.draw_curve)
        self.draw_surface_button.pack(side="bottom")

        for i in range(4):
            point = Point(self.menu_frame, str(i + 1))
            self.points.append(point)
            point.pack(side='top', fill="both",
expand=False)

        self.buttons_frame = tk.Frame(self, width=200,
height=100)

        self.flip_by_x_button =
tk.Button(self.buttons_frame, text="Flip by X",
command=self.flip_by_x)
        self.flip_by_x_button.pack(side="left")
        self.flip_by_y_button =
tk.Button(self.buttons_frame, text="Flip by Y",
command=self.flip_by_y)
        self.flip_by_y_button.pack(side="right")

        self.buttons_frame.pack(side="bottom")

    def make_coordinates(self):
        points_correct = []
        for i in range(len(self.points)):
            a = [self.points[i].coordinates[0],
self.points[i].coordinates[1],
self.points[i].coordinates[2]]
            points_correct.append(a)
        print(points_correct)
        return points_correct

    def set_coordinates(self):

        coordinates = self.make_coordinates()

        u = np.linspace(0, 1, frequency)  # параметр u
```

```
        v = np.linspace(0, 1, frequency)  # параметр v
        x_values = []

        x1 = coordinates[0][0]
        x2 = coordinates[1][0]
        x3 = coordinates[2][0]
        x4 = coordinates[3][0]
        for i in range(len(u)):
            for j in range(len(v)):
                x = x1 * (1 - u[i]) * (1 - v[j]) + x2 *
v[j] * (1 - u[i]) + x3 * (1 - v[j]) * u[i] + x4 * u[i] *
v[j]
                x_values.append(x)

        y_values = []

        y1 = coordinates[0][1]
        y2 = coordinates[1][1]
        y3 = coordinates[2][1]
        y4 = coordinates[3][1]
        for i in range(len(u)):
            for j in range(len(v)):
                y = y1 * (1 - u[i]) * (1 - v[j]) + y2 *
v[j] * (1 - u[i]) + y3 * (1 - v[j]) * u[i] + y4 * u[i] *
v[j]
                y_values.append(y)

        z_values = []

        z1 = coordinates[0][2]
        z2 = coordinates[1][2]
        z3 = coordinates[2][2]
        z4 = coordinates[3][2]
        for i in range(len(u)):
            for j in range(len(v)):
                z = z1 * (1 - u[i]) * (1 - v[j]) + z2 *
v[j] * (1 - u[i]) + z3 * (1 - v[j]) * u[i] + z4 * u[i] *
v[j]
                z_values.append(z)

        x_mesh = np.array(x_values).reshape((frequency,
frequency))  # assuming you have 100 points in u and v
        y_mesh = np.array(y_values).reshape((frequency,
frequency))  # assuming you have 100 points in u and v
```

```
        z_mesh = np.array(z_values).reshape((frequency,
frequency))

        return [x_mesh, y_mesh, z_mesh]

    def rotate_x(self, num, angle):
        x = self.points[num].coordinates[0]
        y = self.points[num].coordinates[1] *
math.cos(angle) - self.points[num].coordinates[2] *
math.sin(angle)
        z = self.points[num].coordinates[1] *
math.sin(angle) + self.points[num].coordinates[2] *
math.cos(angle)
        return [x, y, z]

    def rotate_y(self, num, angle):
        x = self.points[num].coordinates[0] *
math.cos(angle) + self.points[num].coordinates[2] *
math.sin(angle)
        y = self.points[num].coordinates[1]
        z = -self.points[num].coordinates[0] *
math.sin(angle) + self.points[num].coordinates[2] *
math.cos(angle)
        return [x, y, z]

    def flip_by_x(self):
        angle = math.pi
        for i in range(4):
            self.points[i].coordinates = self.rotate_x(i,
angle)
        self.draw_curve()

    def flip_by_y(self):
        angle = math.pi
        for i in range(4):
            self.points[i].coordinates = self.rotate_y(i,
angle)
        self.draw_curve()

    def draw_curve(self):
        if len(self.points) < 4:
            messagebox.showerror("Error", "At least 4
points are required to draw the surface")
            return
```

```
        if self.canvas is not None:
            self.fig.clf()
            self.canvas.get_tk_widget().pack_forget()
            self.canvas.get_tk_widget().destroy()

        # assuming you have 100 points in u and v
        self.fig = plt.figure()
        self.ax = self.fig.add_subplot(111,
projection='3d')
        settled_coordinates = self.set_coordinates()
        surf =
self.ax.plot_surface(settled_coordinates[0],
settled_coordinates[1], settled_coordinates[2],
                                    cmap='spring')

        self.ax.set_xlabel("X", loc='right')
        self.ax.set_ylabel("Y")
        self.ax.set_zlabel("Z")

        self.canvas = FigureCanvasTkAgg(self.fig,
master=self)
        self.canvas.draw()
        self.canvas.get_tk_widget().pack(side="right",
fill="both", expand=True)


if __name__ == "__main__":
    app = SurfaceApp()
    app.mainloop()
```

Приложение:

Ссылка на видео:
https://www.youtube.com/watch?v=QfSVIGJbpH8