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

ОТЧЕТ

по лабораторной работе №6

по дисциплине «Компьютерная графика»

Тема: Реализация трехмерного объекта с использованием библиотеки OpenGL

Студент гр. 0382 Корсунов А.А.

Преподаватель Герасимова Т.В.

Санкт-Петербург

2023

Цель работы.

Разработать программу, реализующую представление разработанного вами трехмерного рисунка, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL. Разработанная программа должна быть пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя, замена типа проекции, управление преобразованиями, как с помощью мыши, так и с помощью диалоговых элементов.

Теоретические положения.

Виды геометрических моделей

Геометрическая модель объекта является машинным представлением его формы и размеров, получаемым прежде всего в результате вычислений, часто связанных на начальном этапе с интерактивными действиями пользователя.

Двумерные модели, которые позволяют формировать и изменять чертежи, были первыми моделями, нашедшими применение. Такое моделирование широко используется до сих пор и вполне устраивает промышленные организации при решении разнообразных задач.
Однако двумерное представление объекта даже с достаточным числом проекций, разрезов и сечений не совсем удобно для сложных объектов.

Трехмерная модель служит основой для создания виртуального представления объекта в трех измерениях. Выделяют три вида трехмерных моделей:

- каркасные (wire, «проволочные») модели;
- поверхностные (surface) модели;

- модели сплошных тел (solid, твердотельные).

+Каркасные модели полностью описываются в терминах точек и линий. Каркасное моделирование представляет собой моделирование самого низкого уровня и имеет ряд серьезных ограничений, большинство из которых возникает из-за недостатка информации о гранях, заключенных между линиями, и невозможности выделить внешнюю и внутреннюю область изображения твердого объемного тела. Однако каркасная модель требует гораздо меньше компьютерной памяти, чем две другие модели, и может оказаться вполне пригодной для решения некоторых задач, относящихся к простым формам.

Каркасная 3D-модель используется для следующих целей.

- Создание базовых 3D-проектов для проверки и быстрого внесения корректировок
- Обзор модели со всех сторон
- Анализ пространственных взаимосвязей, включая расстояния между углами и ребрами, а также визуальная проверка на предмет возможных пересечений
- Создание видов в перспективе
- Создание ортогональных и дополнительных видов

Задание.

Написать программу, рисующую сцену из трехмерных каркасных объектов.

Требования

1 Грани объектов рисуются с помощью доступных функций рисования отрезка в координатах окна. При этом использовать шейдеры GLSL и

OpenGL

2 Вывод объектов с прорисовкой невидимых граней;

3 перемещения, повороты и масштабирование объектов по каждой из осей независимо от остальных.

4 Генерация объектов с заданной мелкостью разбиения.

5 При запуске программы объекты сразу должно быть хорошо видны.

6 Пользователь имеет возможность вращать фигуры (2 степени свободы) и изменять параметры фигур.

7 Возможно изменять положение наблюдателя.

8 Нарисовать оси системы координат.

Все варианты требований могут быть выбраны интерактивно.

Ход работы.

Лабораторная работа выполнялась на языке программирования «Python» с использованием модулей «tkinter» и «OpenGL».

1. Классы вершин и ребер для рисования каркасных фигур

а) Класс вершин

```
class Vertex:  # класс под зарисовки вершин (объект класса принимает 3 координаты и через метод draw через glVertex
    # красит), метод draw необходимо вызывать в конструкции glBegin - glEnd
    def __init__(self, x, y, z):
        self._x = x
        self._y = y
        self._z = z

    def draw(self):
        if self._z is None:
            glVertex2fv((self._x, self._y))
        else:
            glVertex3fv((self._x, self._y, self._z))
```

В конструктор класса следует передавать координаты вершин, через метод draw

происходит отрисовка вершины (следует использовать в конструкции glBegin – glEnd)

б) Класс ребер

```
class Edge:  # класс под зарисовку ребер
    def __init__(self, vertex1, vertex2):  # конструктор принимает 2 вершины для ребра
        self._vertex1 = vertex1
        self._vertex2 = vertex2

    def draw_edge(self):  # рисует сплошнуб линию через класс vertex
        glLineWidth(2)
        glBegin(GL_LINES)
        self._vertex1.draw()
        self._vertex2.draw()
        glEnd()

    def draw_dotted_edge(self):  # рисует точечную линию через класс vertex
        glPushAttrib(GL_ENABLE_BIT)
        glLineStipple(1, 0x1111)
        glEnable(GL_LINE_STIPPLE)
        self.draw_edge()
        glPopAttrib()

    def get_vertexes(self):  # возвращает вершины ребра
        return [self._vertex1, self._vertex2]
```

Все фигуры будут отрисовываться через этот класс, в конструкторе он принимает две вершины, между которыми необходимо ребро. В классе реализовано прорисовка как сплошной линией, так и точечной линией, точечная линия будет использоваться для прорисовки отрицательных полуосей.

2. Отрисовка координатных осей

метод прорисовки класса Axis():

```
def draw(self):

    glMatrixMode(GL_MODELVIEW)  # Эта функция сообщает OpenGL, что следующие операции над матрицами будут
    # применяться к матрице моделирования-вида, которая определяет положение и ориентацию камеры, а также положение
    # и ориентацию всех объектов, отображаемых на экране.
    glPushMatrix()  # Вызов glPushMatrix() позволяет сохранить текущее состояние матрицы в стеке матриц OpenGL,
    # чтобы его можно было восстановить позже с помощью функции glPopMatrix()
    gluLookAt(-2, 2, -6, 0, 0, 0, 0, 1, 0)  # позволяет создать 3D-вид, который можно
```

```
использовать для рендеринга сцены с заданной камерой.

    # glMultMatrixf(self._identity_mat)
    color = 0
    colors = [(1, 0, 0),  # цвета осей
              (1, 1, 0),
              (0, 1, 1),
              (1, 0, 0),
              (1, 1, 0),
              (0, 1, 1)
              ]

    for edge in self._edges:
        glColor3f(colors[color][0], colors[color][1], colors[color][2])
        if color > 2:  # отрицательные полуоси красятся точечными линиями
            Edge.draw_dotted_edge(Edge(self._vertices[edge[0]], self._vertices[edge[1]]))
        else:  # положительные полуоси красятся сплошными линиями
            Edge.draw_edge(Edge(self._vertices[edge[0]], self._vertices[edge[1]]))
        color += 1

    glPopMatrix()

def render(self):
    self.draw()
```

Для отрисовки осей был написан класс Axis, в нем задаются вершины для осей в R^3. При создании объекта класса в конструктор ничего передавать не надо, все данные для отрисовки уже хранятся в полях класса. После создания объекта нужно будет только вызывать метод render(), который прорисовывает оси.

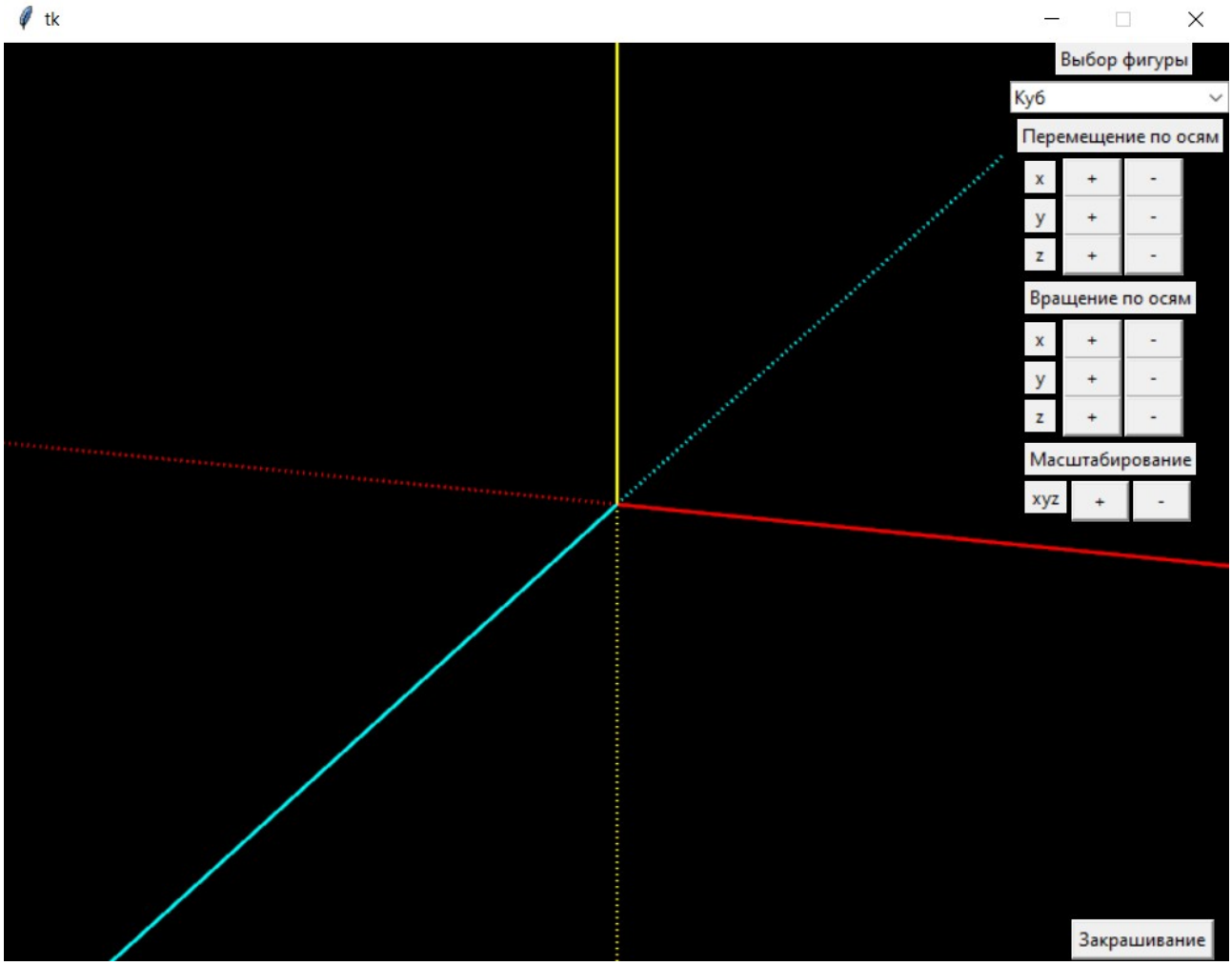

Рисунок 1 — Координатные оси, красная ось — x, желтая — y, голубая - z

2. Класс камеры

Главные методы класса:

```
def rotate_x(self):  # поворот фигуры по оси x
    glLoadIdentity()  # загрузка единичной матрицы
    glRotatef(self.rx, 1, 0, 0)  # поворот вокруг оси X на угол rx
    glMultMatrixf(self._identity_mat)  # перемножает текущую матрицу с единичной матрицей,
    # чтобы сохранить текущее положение объекта.
    self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую матрицу моделирования и просмотра,
    # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную self._identity_mat

def rotate_y(self):  # то же самое, но по оси y
    glLoadIdentity()
    glRotatef(self.ry, 0, 1, 0)
    glMultMatrixf(self._identity_mat)
    self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
```

```
def rotate_z(self):  # то же самое, но по оси z
    glLoadIdentity()
    glRotatef(self.rz, 0, 0, 1)
    glMultMatrixf(self._identity_mat)
    self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

def translate(self):  # перемещение фигуры
    glLoadIdentity()  # загрузка единичной матрицы
    glTranslatef(self.tx, self.ty, self.tz)  # перемещение на tx, ty, tz
    glMultMatrixf(self._identity_mat)  # перемножает текущую матрицу с единичной матрицей,
    # чтобы сохранить текущее положение объекта.
    self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую матрицу моделирования и просмотра,
    # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную self._identity_mat

def render(self):  # рендер измененного объекта
    self.translate()
    self.rotate_y()
    self.rotate_x()
    self.rotate_z()
```

Перемещение и повороты камеры реализованы через функции glTranslate и glRotate соответственно. Перемещение камеры назначены на следующие клавиши клавиатуры: «a» и «d» (по оси x), «w» и «s» (по оси z), «q» и «e» (по оси y), и для поворота камеры «x» «z» (по z). Клавиши были привязаны к событиям нажатия через библиотеку tkinter. На рисунке 2 можно увидеть пример изменения положения камеры наблюдателя.

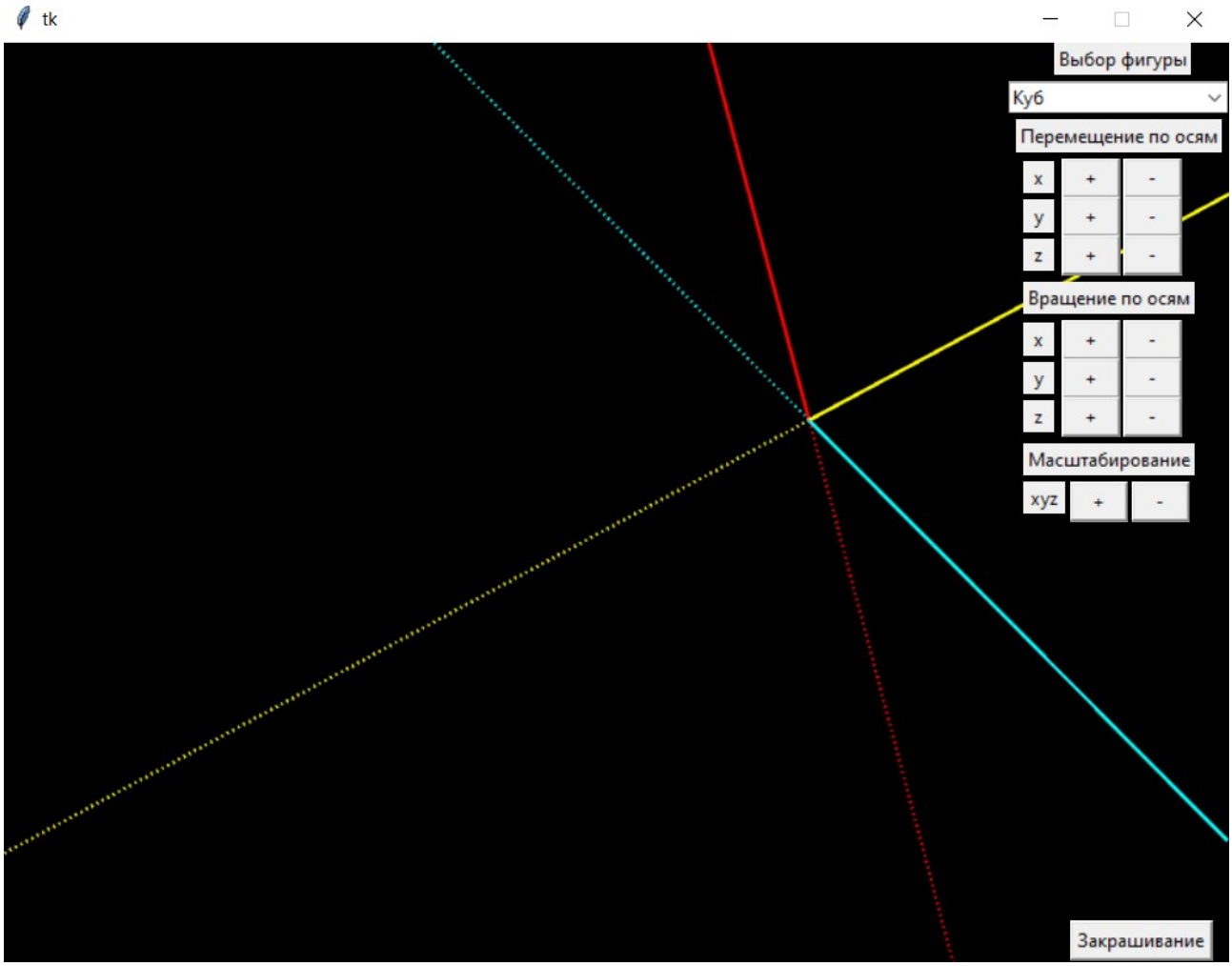

Рисунок 2 — Изменение положения камеры

3. Универсальный класс под фигуры (класс Figure)

Основные методы класса:

```
def render(self):
    glMatrixMode(GL_MODELVIEW)
    glPushMatrix()
    gluLookAt(self.coordinates[0], self.coordinates[1], self.coordinates[2], self.coordinates[3],
              self.coordinates[4], self.coordinates[5], self.coordinates[6], self.coordinates[7],
              self.coordinates[8])  # перемещение куба
    # в начало координат (его центра)
    glMultMatrixf(self._trans_mat)  # перемножение матриц, чтобы сохранить
преобразования
    glMultMatrixf(self._rotation_mat)
    glMultMatrixf(self._scale_mat)

    glPushMatrix()
    self.move_x()  # перемещение
    self.move_y()
    self.move_z()
```

```
    glPopMatrix()

    glPushMatrix()
    glLoadIdentity()
    self.rotate_global()  # поворот
    self._rotation_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
    glPopMatrix()

    glPushMatrix()
    glLoadIdentity()
    self.scale()  # масштабирование
    self._scale_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
    glPopMatrix()

    self.draw()
    glPopMatrix()

def move_x(self):  # перемещение по x
    glLoadMatrixf(self._trans_mat)  # загрузка единичной матрицы
    glTranslatef(self.tx, 0, 0)  # перемещение на tx, 0, 0
    self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую матрицу
моделирования и просмотра,
    # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную
self._trans_mat

def move_y(self):  # то же самое по y
    glLoadMatrixf(self._trans_mat)
    glTranslatef(0, self.ty, 0)
    self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

def move_z(self):  # то же самое по z
    glLoadMatrixf(self._trans_mat)
    glTranslatef(0, 0, self.tz)
    self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

def rotate_global(self):  # сначала поворачивает объект вокруг глобальной системы
координат, а затем применяет
    # локальную матрицу преобразования. Это означает, что объект вращается вокруг
глобальной оси, а затем его
    # новая ориентация преобразуется обратно в локальную систему координат.
    glRotatef(self.rx, 1, 0, 0)
    glRotatef(self.ry, 0, 1, 0)
    glRotatef(self.rz, 0, 0, 1)
    glMultMatrixf(self._rotation_mat)

def scale(self):
    glScale(self.sxyz, self.sxyz, self.sxyz)
    glMultMatrixf(self._scale_mat)
```

Данный класс на основе переданного в конструктор списка ребер отображает получившуюся фигуру. В нем реализованы методы перемещения, поворота и масштабирования фигуры. Демонстрация будет осуществляться на кубе:

```
vertex0_cube = Vertex(1, -1, -1)
vertex1_cube = Vertex(1, 1, -1)
vertex2_cube = Vertex(-1, 1, -1)
vertex3_cube = Vertex(-1, -1, -1)
vertex4_cube = Vertex(1, -1, 1)
vertex5_cube = Vertex(1, 1, 1)
vertex6_cube = Vertex(-1, -1, 1)
vertex7_cube = Vertex(-1, 1, 1)

edges_cube = (Edge(vertex0_cube, vertex1_cube),
        Edge(vertex0_cube, vertex3_cube),
        Edge(vertex0_cube, vertex4_cube),
        Edge(vertex2_cube, vertex1_cube),
        Edge(vertex2_cube, vertex3_cube),
        Edge(vertex2_cube, vertex7_cube),
        Edge(vertex6_cube, vertex3_cube),
        Edge(vertex6_cube, vertex4_cube),
        Edge(vertex6_cube, vertex7_cube),
        Edge(vertex5_cube, vertex1_cube),
        Edge(vertex5_cube, vertex4_cube),
        Edge(vertex5_cube, vertex7_cube),
        )
```

выше приведен код для создания ребер под куб. Преобразования с фигурами реализовано через виджеты tkinter, которые располагаются справа-сверху окна (рис. 2) .Демонстрация работы для куба приведена на следующих рисунках:

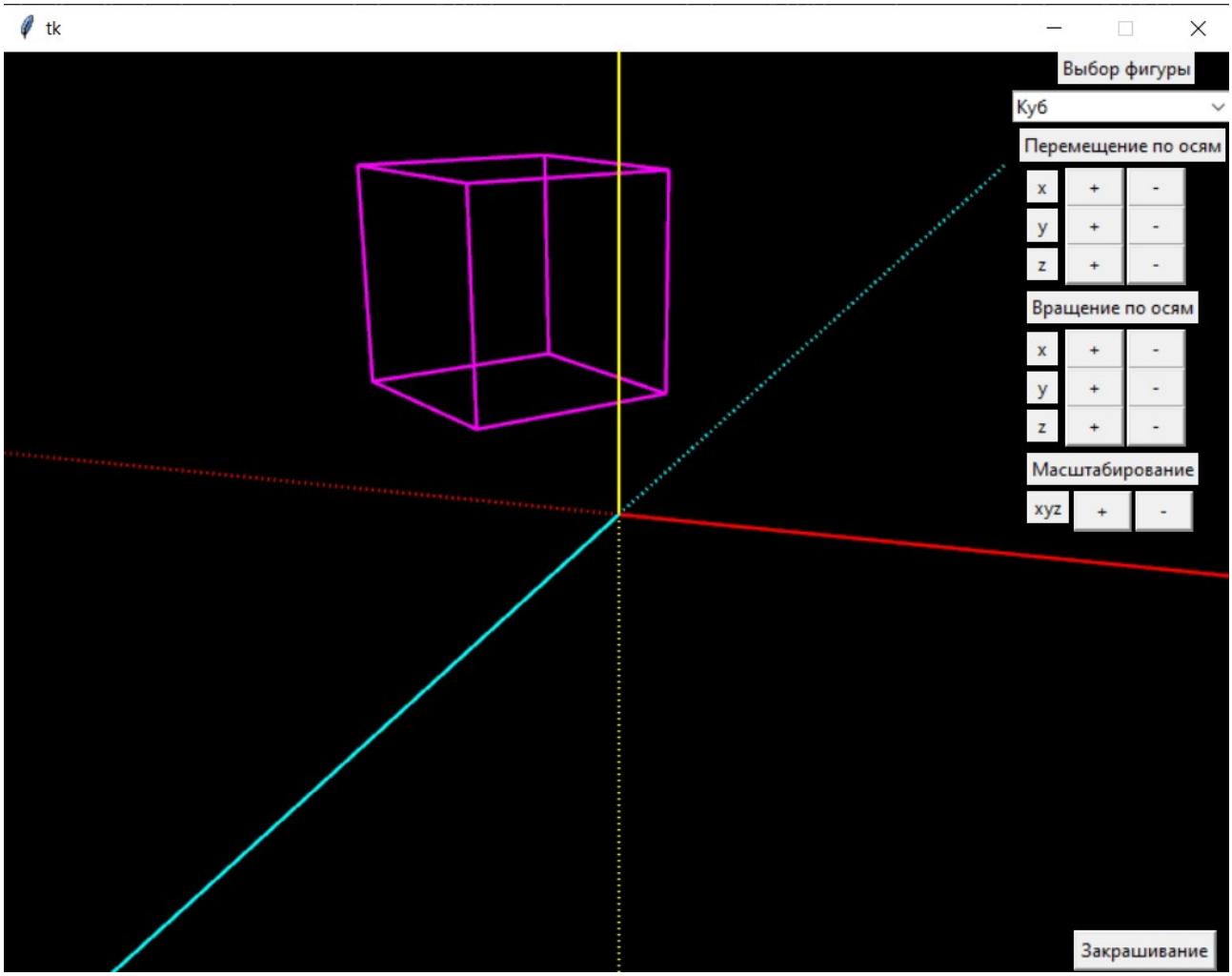

Рисунок 3 — начальное положение куба

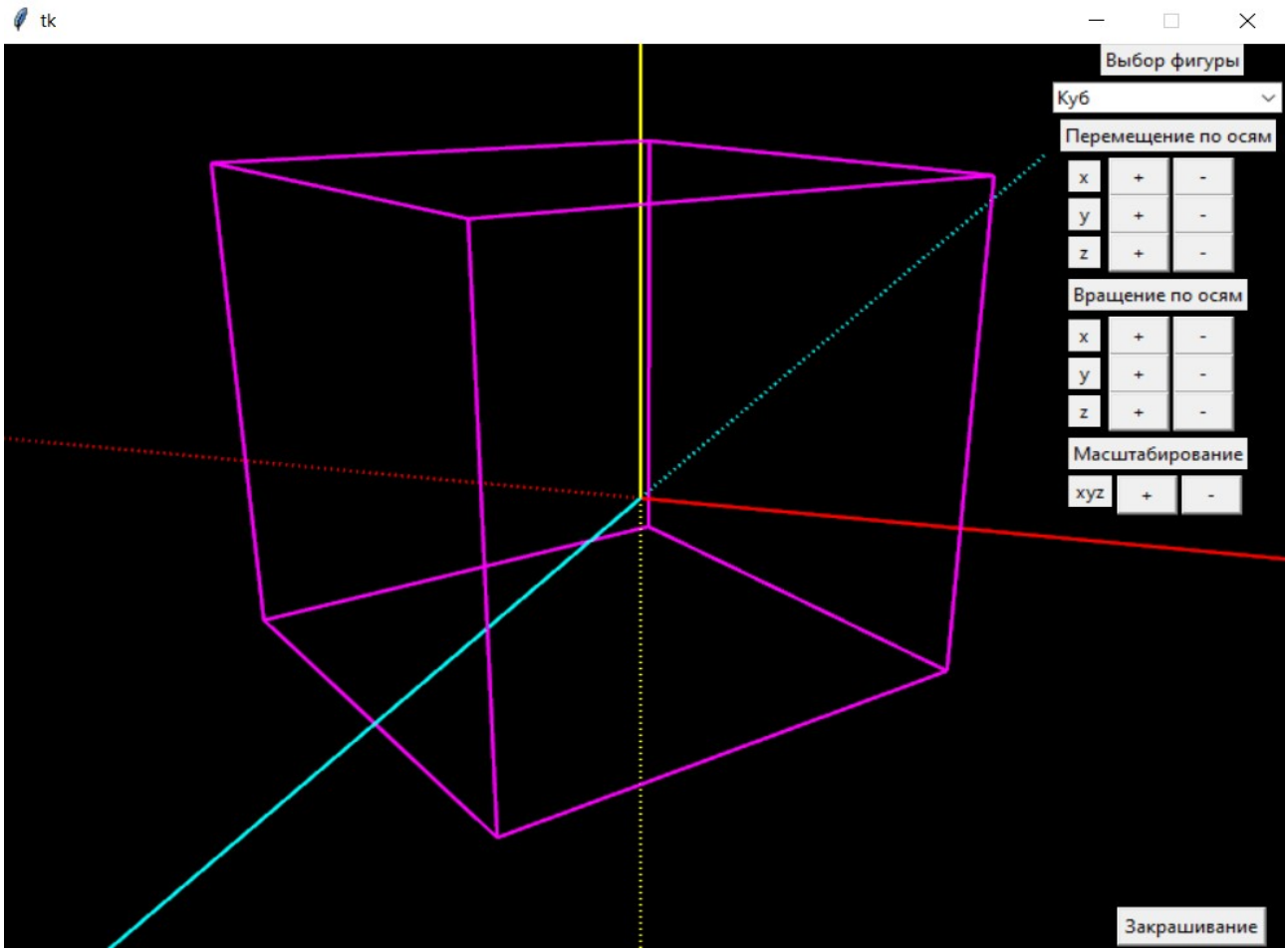

Рисунок 4 — Перемещение куба

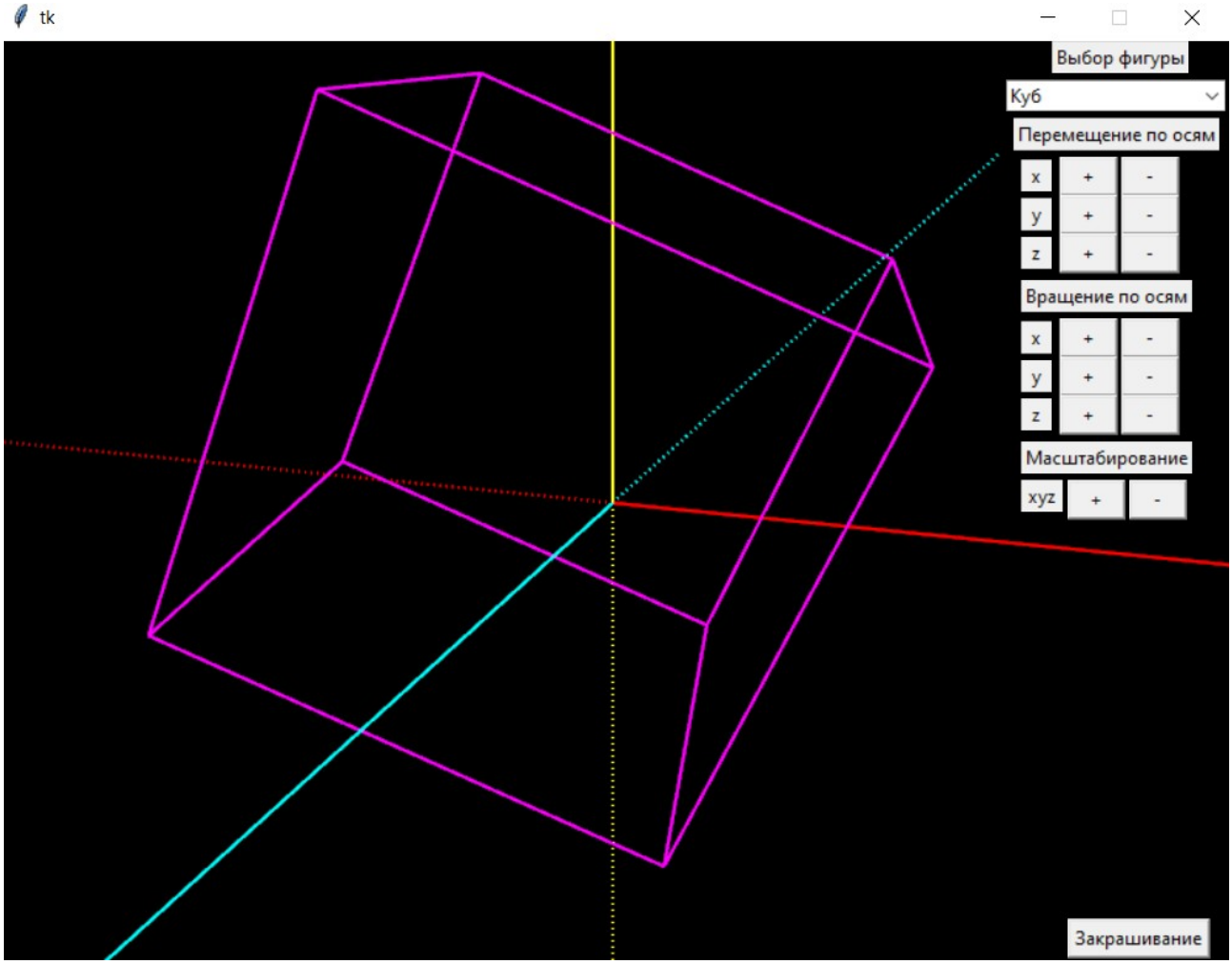

Рисунок 5 — Вращение куба

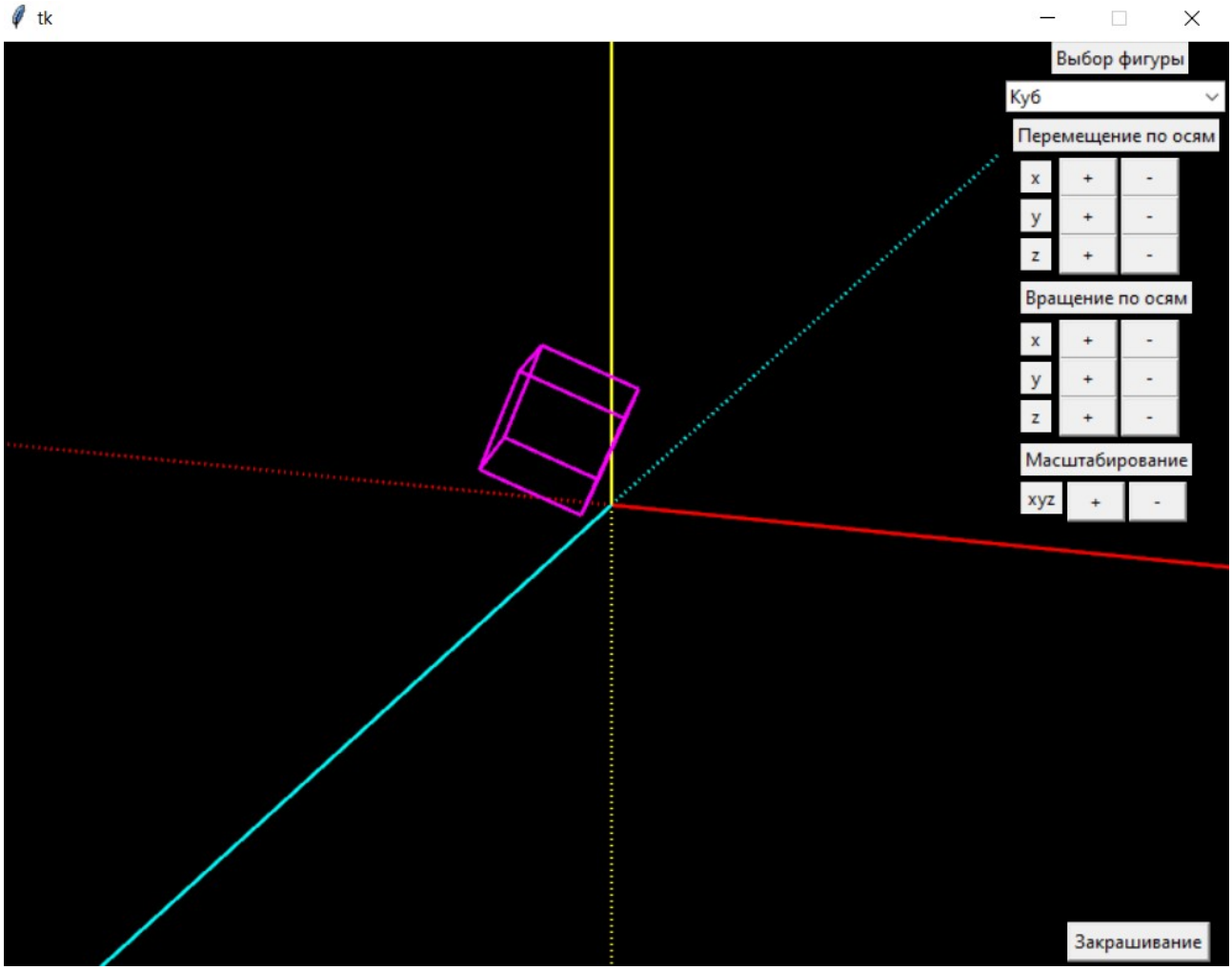

Рисунок 6 — Масштабирование куба

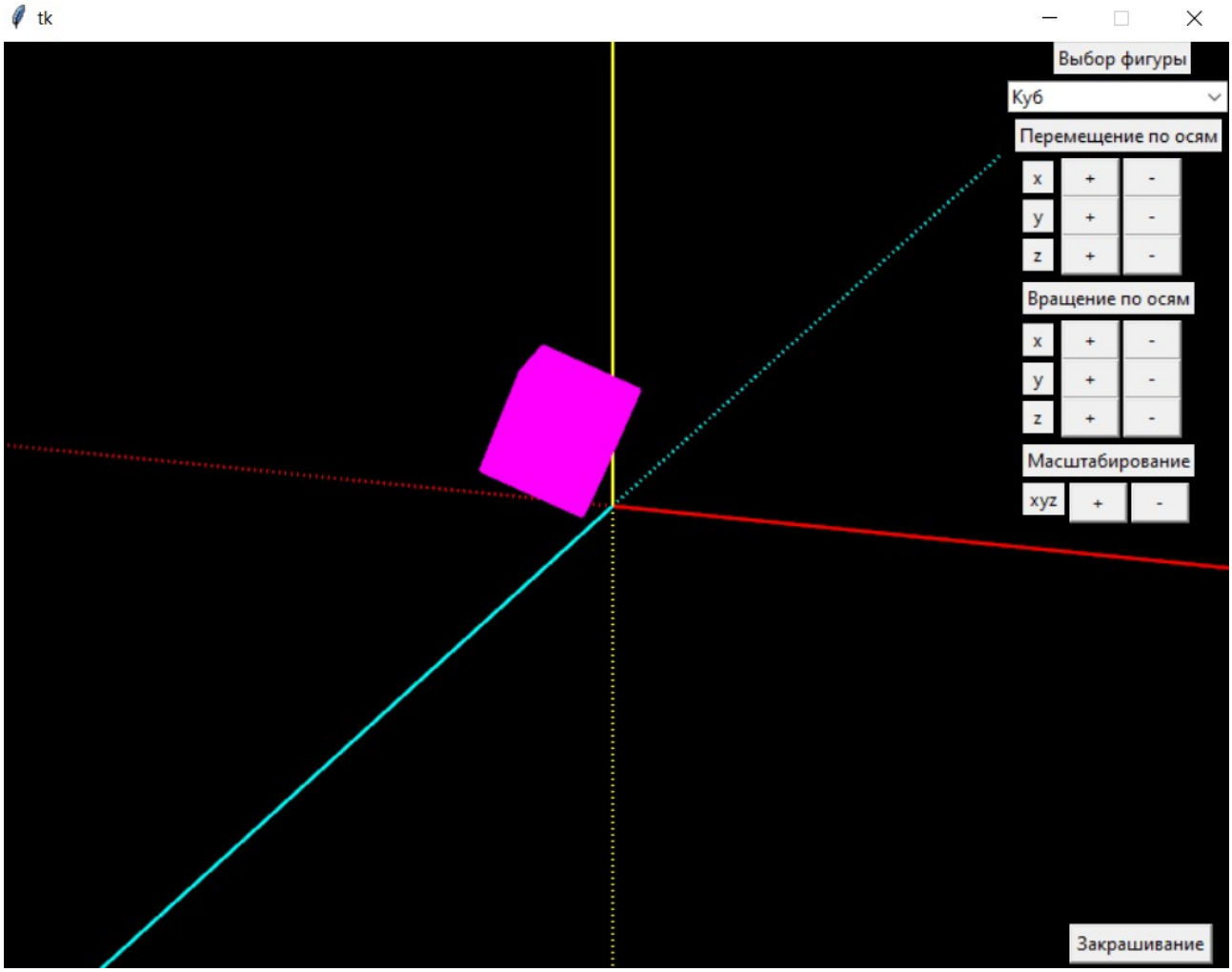

Рисунок 8 — Заливка куба

4. Добавление других фигур

а) Пирамида

Ребра для пирамиды:

```
vertex1_pyramid = Vertex(-1, 0, 0)
vertex2_pyramid = Vertex(0, 0, 1)
vertex3_pyramid = Vertex(1, 0, 0)
vertex4_pyramid = Vertex(0, 0, -1)
vertex5_pyramid = Vertex(0, 1, 0)

edges_pyramid = (Edge(vertex1_pyramid, vertex5_pyramid),
                 Edge(vertex2_pyramid, vertex5_pyramid),
                 Edge(vertex3_pyramid, vertex5_pyramid),
                 Edge(vertex4_pyramid, vertex5_pyramid),
                 Edge(vertex1_pyramid, vertex4_pyramid),
                 Edge(vertex3_pyramid, vertex4_pyramid),
                 Edge(vertex2_pyramid, vertex3_pyramid),
                 Edge(vertex1_pyramid, vertex2_pyramid))
```

б) Конус

Ребра для конуса:

```
vertexes_circle = []
createCircle(0, 0, 0, 1)
edge_cone = []
vertex_cone = Vertex(0, 1, 0)

for i in range(len(vertexes_circle) - 1):
    temp_vertex1 = Vertex(vertexes_circle[i][0], vertexes_circle[i][2], vertexes_circle[i][1])
    temp_vertex2 = Vertex(vertexes_circle[i + 1][0], vertexes_circle[i + 1][2], vertexes_circle[i + 1][1])
    edge_cone.append(Edge(temp_vertex1, temp_vertex2))
    if i % 5 == 0:
        edge_cone.append(Edge(temp_vertex1, vertex_cone))
```

здесь createCircle – функция для создания массива под вершины круга:

```
def createCircle(shift_x, shift_y, shift_z, R):
    global vertexes_circle
    steps = 100
    angle = math.pi * 2 / steps

    for i in range(steps):
        newX = R * math.sin(angle * i) + shift_x
        newY = -R * math.cos(angle * i) + shift_y
        vertexes_circle.append([newX, newY, shift_z])

    newX = R * math.sin(angle * 1000) + shift_x
    newY = -R * math.cos(angle * 1000) + shift_y
    vertexes_circle.append([newX, newY, shift_z])
```

в) Цилиндр:

Ребра под цилиндр:

```
edge_cylinder = []
temp_circle = vertexes_circle.copy()
vertexes_circle.clear()
createCircle(0, 0, 1, 1)

for i in range(len(temp_circle) - 1):
    temp_vertex1 = Vertex(temp_circle[i][0], temp_circle[i][2], temp_circle[i][1])
    temp_vertex2 = Vertex(temp_circle[i + 1][0], temp_circle[i + 1][2], temp_circle[i + 1][1])

    temp1_vertex1 = Vertex(vertexes_circle[i][0], vertexes_circle[i][2], vertexes_circle[i][1])
    temp1_vertex2 = Vertex(vertexes_circle[i + 1][0], vertexes_circle[i + 1][2], vertexes_circle[i + 1]
```

[1])

```
edge_cylinder.append(Edge(temp_vertex1, temp_vertex2))
edge_cylinder.append(Edge(temp_vertex1, temp1_vertex1))
edge_cylinder.append(Edge(temp1_vertex1, temp1_vertex2))
```

Далее пойдут примеры демонстрации работы всех фигур вместе. Важно отметить, что преобразования фигур не влияют друг на друга (т. е. фигуры можно преобразовывать независимо от других). Выбор текущей фигуры осуществляется через виджет справа-сверху окна (по умолчанию выставлено преобразование куба).

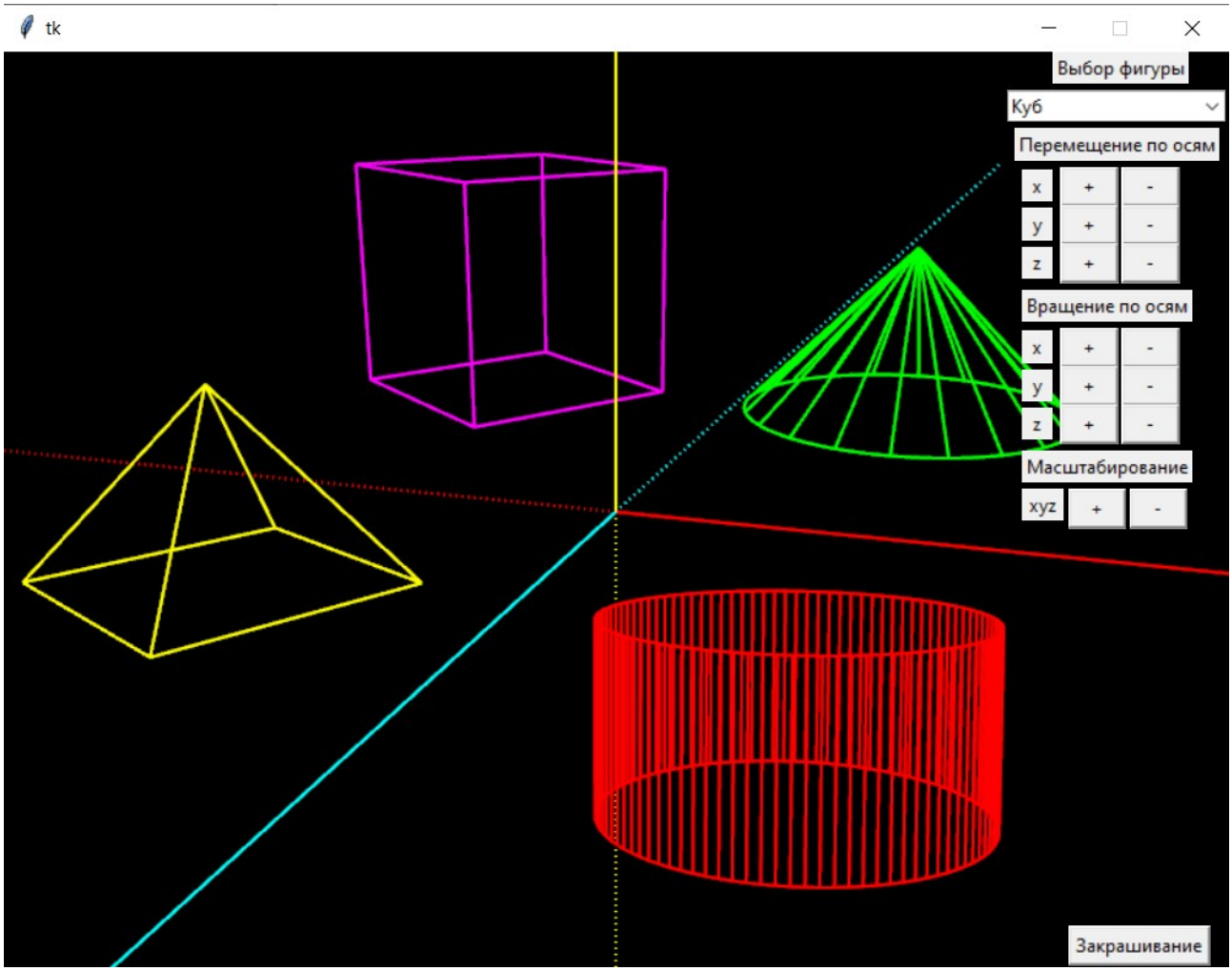

Рисунок 9 — Начальное положение всех фигур

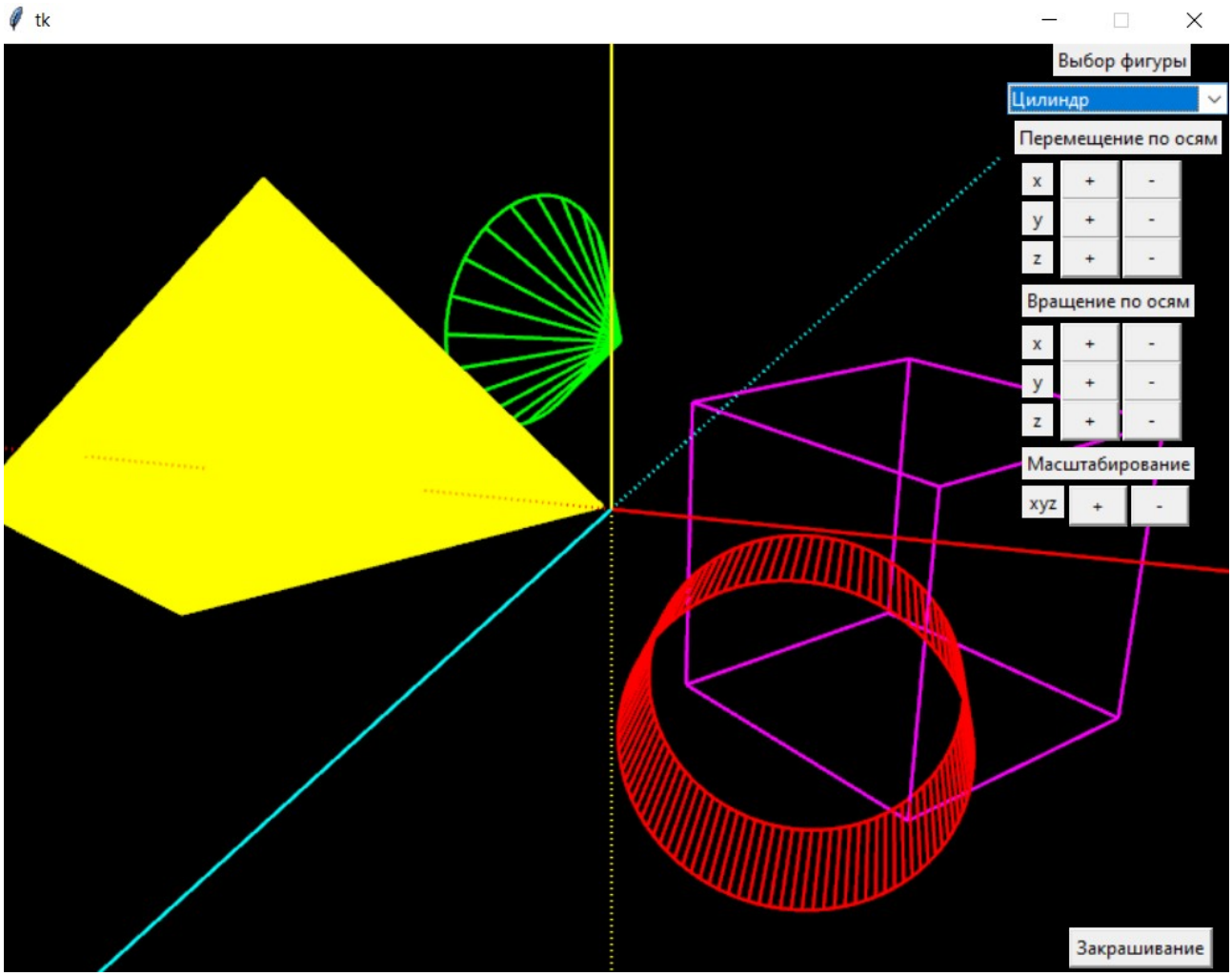

Рисунок 10 — Некоторое преобразование всех фигур

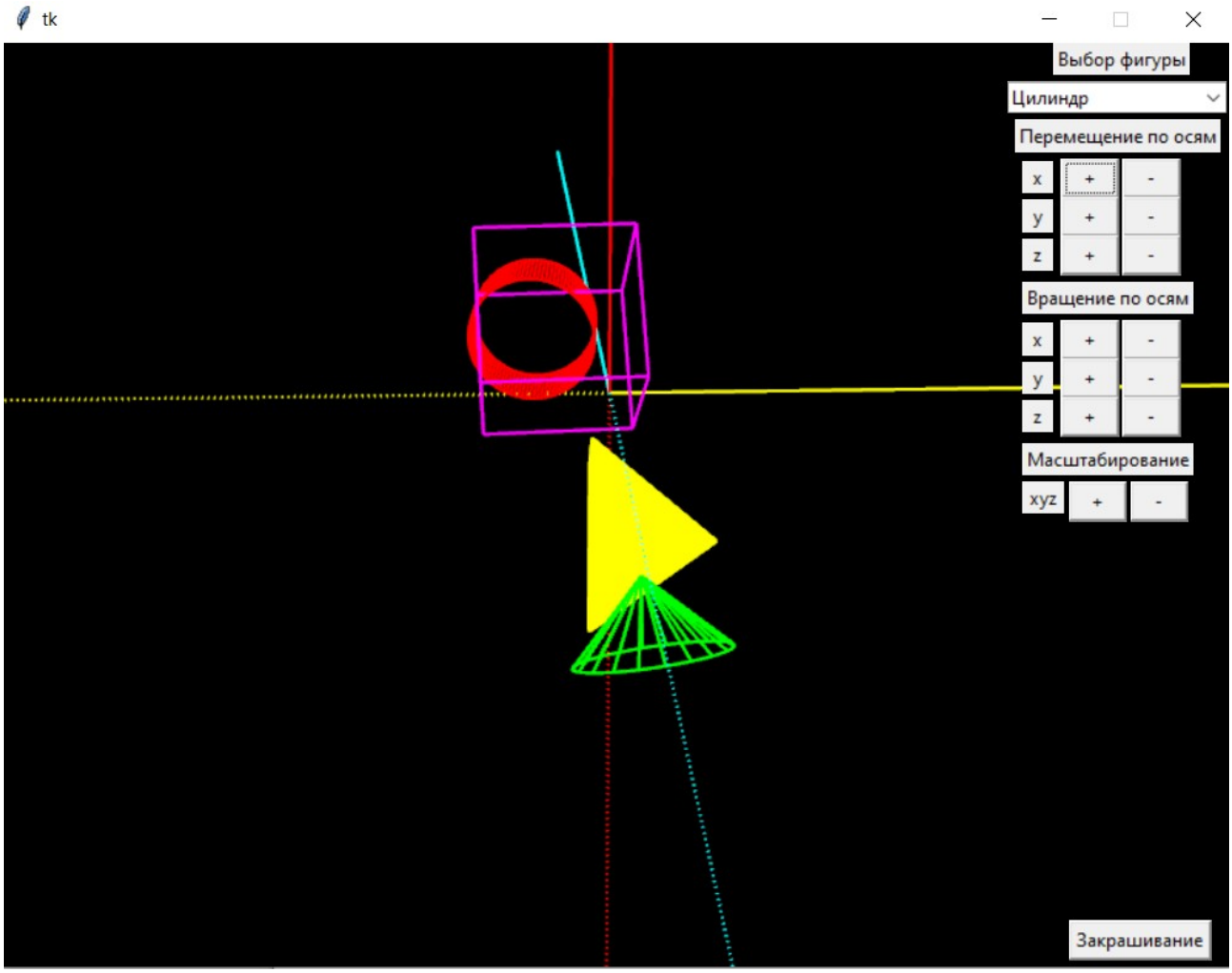

Рисунок 11 — То же преобразование, но с измененным положением камеры

Вывод.

Была разработана программа, реализующая представление трехмерного рисунка, используя предложенные функции библиотеки OpenGL (матрицы видового преобразования, проецирование) и язык GLSL. Разработанная программа была пополнена возможностями остановки интерактивно различных атрибутов через вызов соответствующих элементов интерфейса пользователя, замена типа проекции, управление преобразованиями, как с помощью мыши, так и с помощью диалоговых элементов.

Приложение А. Исходный код.

Файл model.axis.py

```
from model.edge import Edge
from OpenGL.GL import glPushMatrix, glMatrixMode, glColor3f, glPopMatrix, GL_MODELVIEW
from OpenGL.GLU import gluLookAt
from model.identity_mat import identity_mat44
from model.vertex import Vertex


class Axis:
    def __init__(self):  # класс под зарисовку осей
        self._identity_mat = identity_mat44()  # единичная матрица 4x4
        self._vertices = [Vertex(0, 0, 0),  # вершины под оси
                          Vertex(-100, 0, 0),
                          Vertex(0, 100, 0),
                          Vertex(0, 0, -100),
                          Vertex(100, 0, 0),
                          Vertex(0, -100, 0),
                          Vertex(0, 0, 100),
                          ]
        self._edges = [(0, 1),  # для отрисовки ребер
                       (0, 2),
                       (0, 3),
                       (0, 4),
                       (0, 5),
                       (0, 6),
                       ]

    def draw(self):

        glMatrixMode(GL_MODELVIEW)  # Эта функция сообщает OpenGL, что следующие операции над матрицами будут
        # применяться к матрице моделирования-вида, которая определяет положение и ориентацию камеры, а также положение
        # и ориентацию всех объектов, отображаемых на экране.
        glPushMatrix()  # Вызов glPushMatrix() позволяет сохранить текущее состояние матрицы в стеке матриц OpenGL,
        # чтобы его можно было восстановить позже с помощью функции glPopMatrix()
        gluLookAt(-2, 2, -6, 0, 0, 0, 0, 1, 0)  # Таким образом, данная функция устанавливает камеру в точке (-2, 2, -6)
        # с направлением взгляда на точку (0, 0, 0) и считает, что направление "вверх" камеры определяется вектором
        # (0, 1, 0). Это позволяет создать 3D-вид, который можно использовать для рендеринга сцены с заданной камерой.

        # glMultMatrixf(self._identity_mat)
        color = 0
```

```
        colors = [(1, 0, 0),  # цвета осей
                  (1, 1, 0),
                  (0, 1, 1),
                  (1, 0, 0),
                  (1, 1, 0),
                  (0, 1, 1)
                  ]

        for edge in self._edges:
            glColor3f(colors[color][0], colors[color][1], colors[color][2])
            if color > 2:  # отрицательные полуоси красятся точечными линиями
                Edge.draw_dotted_edge(Edge(self._vertices[edge[0]], self._vertices[edge[1]]))
            else:  # положительные полуоси красятся сплошными линиями
                Edge.draw_edge(Edge(self._vertices[edge[0]], self._vertices[edge[1]]))
            color += 1

        glPopMatrix()

    def render(self):
        self.draw()
```

файл model.camera.py:

```
from OpenGL.GL import *

from model.identity_mat import identity_mat44


class Camera:
    def __init__(self):
        self._identity_mat = identity_mat44()  # матрица преобразований (изначально единичная матрица 4x4)
        self.tx = 0  # координаты по перемещению
        self.ty = 0
        self.tz = 0
        self.ry = 0  # координаты по вращению
        self.rx = 0
        self.rz = 0

    def rotate_x(self):  # поворот фигуры по оси x
        glLoadIdentity()  # загрузка единичной матрицы
        glRotatef(self.rx, 1, 0, 0)  # поворот вокруг оси X на угол rx
        glMultMatrixf(self._identity_mat)  # перемножает текущую матрицу с единичной матрицей,
        # чтобы сохранить текущее положение объекта.
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую матрицу моделирования и просмотра,
        # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную self._identity_mat
```

```
    def rotate_y(self):  # то же самое, но по оси y
        glLoadIdentity()
        glRotatef(self.ry, 0, 1, 0)
        glMultMatrixf(self._identity_mat)
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def rotate_z(self):  # то же самое, но по оси z
        glLoadIdentity()
        glRotatef(self.rz, 0, 0, 1)
        glMultMatrixf(self._identity_mat)
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def translate(self):  # перемещение фигуры
        glLoadIdentity()  # загрузка единичной матрицы
        glTranslatef(self.tx, self.ty, self.tz)  # перемещение на tx, ty, tz
        glMultMatrixf(self._identity_mat)  # перемножает текущую матрицу с единичной
матрицей,
        # чтобы сохранить текущее положение объекта.
        self._identity_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую
матрицу моделирования и просмотра,
        # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную
self._identity_mat

    def render(self):  # рендер измененного объекта
        self.translate()
        self.rotate_y()
        self.rotate_x()
        self.rotate_z()
```

файл model.edge.py:

```
from OpenGL.GL import *


class Edge:  # класс под зарисовку ребер
    def __init__(self, vertex1, vertex2):  # конструктор принимает 2 вершины для ребра
        self._vertex1 = vertex1
        self._vertex2 = vertex2

    def draw_edge(self):  # рисует сплошнуб линию через класс vertex
        glLineWidth(2)
        glBegin(GL_LINES)
        self._vertex1.draw()
        self._vertex2.draw()
        glEnd()

    def draw_dotted_edge(self):  # рисует точечную линию через класс vertex
        glPushAttrib(GL_ENABLE_BIT)
```

```
        glLineStipple(1, 0x1111)
        glEnable(GL_LINE_STIPPLE)
        self.draw_edge()
        glPopAttrib()

    def get_vertexes(self):  # возвращает вершины ребра
        return [self._vertex1, self._vertex2]
```

файл model.figure.py:

```
from OpenGL.GL import *
from OpenGL.GLU import gluLookAt
from model.identity_mat import identity_mat44


class Figure:
    def __init__(self, edges, color, coordinates):  # конструктору нужно передать ребра куба
        self._trans_mat = identity_mat44()  # матрица перемещения
        self._rotation_mat = identity_mat44()  # матрица поворота
        self._scale_mat = identity_mat44()  # матрица масштабирования
        self._edges = edges
        self.tx = 0  # координаты перемещения
        self.ty = 0
        self.tz = 0
        self.rx = 0  # координаты поворота
        self.ry = 0
        self.rz = 0
        self.sxyz = 1
        # self.sx = 0  # координаты масштабирования
        # self.sy = 0
        # self.sz = 0
        self.fill = False
        self.color = color
        self.coordinates = coordinates

    def change_fill(self):  # для закрашывания фигуры
        self.fill = not self.fill

    def draw(self):
        glPushMatrix()

        glColor3f(self.color[0], self.color[1], self.color[2])
        temp_vertexes = []

        for edge in self._edges:  # отрисовка ребер
            edge.draw_edge()
            temp_vertexes.append(edge.get_vertexes()[0])
            temp_vertexes.append(edge.get_vertexes()[1])
```

```
        if self.fill:  # закрашивание граней
            glBegin(GL_POLYGON)
            for i in range(len(temp_vertexes)):
                temp_vertexes[i].draw()
            glEnd()

        glPopMatrix()

    def render(self):
        glMatrixMode(GL_MODELVIEW)
        glPushMatrix()
        gluLookAt(self.coordinates[0], self.coordinates[1], self.coordinates[2], self.coordinates[3],
                  self.coordinates[4], self.coordinates[5], self.coordinates[6], self.coordinates[7],
                  self.coordinates[8])  # перемещение куба
        # в начало координат (его центра)
        glMultMatrixf(self._trans_mat)  # перемножение матриц, чтобы сохранить
преобразования
        glMultMatrixf(self._rotation_mat)
        glMultMatrixf(self._scale_mat)

        glPushMatrix()
        self.move_x()  # перемещение
        self.move_y()
        self.move_z()
        glPopMatrix()

        glPushMatrix()
        glLoadIdentity()
        # if local_rot:
        # self.rotate_local()
        # else:
        self.rotate_global()  # поворот
        self._rotation_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
        glPopMatrix()

        glPushMatrix()
        glLoadIdentity()
        self.scale()  # масштабирование
        self._scale_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
        glPopMatrix()

        self.draw()
        glPopMatrix()

    # def rotate(self):
    #   glPushMatrix()
    #  glLoadMatrixf(self._trans_mat)
    # glRotatef(0.5, 0, 1, 0)
```

```
    # self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)
    # glPopMatrix()

    def move_x(self):  # перемещение по x
        glLoadMatrixf(self._trans_mat)  # загрузка единичной матрицы
        glTranslatef(self.tx, 0, 0)  # перемещение на tx, 0, 0
        self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)  # чтобы получить новую
матрицу моделирования и просмотра,
        # которая включает в себя поворот вокруг оси X, и сохраняет ее в переменную
self._trans_mat

    def move_y(self):  # то же самое по y
        glLoadMatrixf(self._trans_mat)
        glTranslatef(0, self.ty, 0)
        self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    def move_z(self):  # то же самое по z
        glLoadMatrixf(self._trans_mat)
        glTranslatef(0, 0, self.tz)
        self._trans_mat = glGetFloatv(GL_MODELVIEW_MATRIX)

    # def rotate_local(self):  # вращает объект сначала вокруг его собственной системы
координат, а затем применяет
    # глобальную матрицу преобразования. Это означает, что объект вращается сначала
вокруг своей собственной оси,
    # а затем его новая ориентация преобразуется в глобальную систему координат.
    #    glMultMatrixf(self._rotation_mat)
    #   glRotatef(self.rx, 1, 0, 0)
    #  glRotatef(self.ry, 0, 1, 0)
    # glRotatef(self.rz, 0, 0, 1)

    def rotate_global(self):  # сначала поворачивает объект вокруг глобальной системы
координат, а затем применяет
        # локальную матрицу преобразования. Это означает, что объект вращается вокруг
глобальной оси, а затем его
        # новая ориентация преобразуется обратно в локальную систему координат.
        glRotatef(self.rx, 1, 0, 0)
        glRotatef(self.ry, 0, 1, 0)
        glRotatef(self.rz, 0, 0, 1)
        glMultMatrixf(self._rotation_mat)

    def scale(self):
        glScale(self.sxyz, self.sxyz, self.sxyz)
        glMultMatrixf(self._scale_mat)

    def stop(self):  # остановить все преобразования
        self.tx = 0
        self.ty = 0
        self.tz = 0
```

```
        self.ry = 0
        self.rx = 0
        self.rz = 0
        self.sxyz = 1
```

файл model.identity_mat.py:

```
import numpy


def identity_mat44():  # возвращает единичную матрицу 4x4
    return numpy.matrix(numpy.i
```

файл model.vertex.py:

```
from OpenGL.GL import glVertex3fv, glVertex2fv


class Vertex:  # класс под зарисовки вершин (объект класса принимает 3 координаты и
через метод draw через glVertex
    # красит), метод draw необходимо вызывать в конструкции glBegin - glEnd
    def __init__(self, x, y, z):
        self._x = x
        self._y = y
        self._z = z

    def draw(self):
        if self._z is None:
            glVertex2fv((self._x, self._y))
        else:
            glVertex3fv((self._x, self._y, self._z))
```

файл view.main.py

```
from tkinter import *
from pyopengltk import OpenGLFrame
from OpenGL.GL import *
from OpenGL.GLU import *
from model.axis import Axis
from model.camera import Camera
from model.vertex import Vertex
from model.edge import Edge
from model.figure import Figure
from tkinter.ttk import Combobox
import math


def resize(width, height):
```

```
    glViewport(0, 0, width, height)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(40.0, float(width / height), 0.1, 50.0)
    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()


def click_fill_figure():
    figures[figure_id].change_fill()


def get_figure():
    temp_figure = combo_select_figure.get()
    global figure_id
    if temp_figure == "Куб":
        figure_id = 0
    elif temp_figure == "Пирамида":
        figure_id = 1
    elif temp_figure == "Конус":
        figure_id = 2
    elif temp_figure == "Цилиндр":
        figure_id = 3


def camera_plus_tx(event):
    camera.tx = 0.1


def camera_minus_tx(event):
    camera.tx = -0.1


def camera_plus_tz(event):
    camera.tz = 0.1


def camera_minus_tz(event):
    camera.tz = -0.1


def camera_plus_ty(event):
    camera.ty = 0.1


def camera_minus_ty(event):
    camera.ty = -0.1
```

```
def camera_plus_ry(event):
    camera.ry = 1.0


def camera_minus_ry(event):
    camera.ry = -1.0


def camera_plus_rx(event):
    camera.rx = 1.0


def camera_minus_rx(event):
    camera.rx = -1.0


def camera_plus_rz(event):
    camera.rz = 1.0


def camera_minus_rz(event):
    camera.rz = -1.0


def plus_axes_x(event):
    figures[figure_id].tx = 0.01


def minus_axes_x(event):
    figures[figure_id].tx = -0.01


def plus_axes_y(event):
    figures[figure_id].ty = 0.01


def minus_axes_y(event):
    figures[figure_id].ty = -0.01


def plus_axes_z(event):
    figures[figure_id].tz = 0.01


def minus_axes_z(event):
    figures[figure_id].tz = -0.01


def plus_axes_rx(event):
```

```
    figures[figure_id].rx = 0.5


def minus_axes_rx(event):
    figures[figure_id].rx = -0.5


def plus_axes_ry(event):
    figures[figure_id].ry = 0.5


def minus_axes_ry(event):
    figures[figure_id].ry = -0.5


def plus_axes_rz(event):
    figures[figure_id].rz = 0.5


def minus_axes_rz(event):
    figures[figure_id].rz = -0.5


def plus_axes_sxyz(event):
    figures[figure_id].sxyz = 1.01


def minus_axes_sxyz(event):
    figures[figure_id].sxyz = 1 / 1.01


def camera_stop(event):
    if camera.tx > 0:
        camera.tx = 0
    elif camera.tx < 0:
        camera.tx = 0
    elif camera.tz > 0:
        camera.tz = 0
    elif camera.tz < 0:
        camera.tz = 0
    elif camera.ty > 0:
        camera.ty = 0.0
    elif camera.ty < 0:
        camera.ty = 0.0
    elif camera.ry > 0:
        camera.ry = 0.0
    elif camera.ry < 0:
        camera.ry = 0.0
    elif camera.rx < 0:
```

```
        camera.rx = 0.0
    elif camera.rx > 0:
        camera.rx = 0.0
    elif camera.rz < 0:
        camera.rz = 0.0
    elif camera.rz > 0:
        camera.rz = 0.0


def figure_stop(event):
    figures[figure_id].stop()


def createCircle(shift_x, shift_y, shift_z, R):
    global vertexes_circle
    steps = 100
    angle = math.pi * 2 / steps

    for i in range(steps):
        newX = R * math.sin(angle * i) + shift_x
        newY = -R * math.cos(angle * i) + shift_y
        vertexes_circle.append([newX, newY, shift_z])

    newX = R * math.sin(angle * 1000) + shift_x
    newY = -R * math.cos(angle * 1000) + shift_y
    vertexes_circle.append([newX, newY, shift_z])


class DrawingWindow(OpenGLFrame):  # создание класса на основе пакета pyopengltk

    def initgl(self):  # инициализация
        resize(*SCREEN_SIZE)

    def redraw(self):  # перерисовка
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glPushMatrix()
        glEnable(GL_DEPTH_TEST)

        camera.render()
        axis.render()

        get_figure()
        for i in range(len(figures)):
            figures[i].render()

        glPopMatrix()

        glMatrixMode(GL_PROJECTION)  # устанавливается текущая матрица
        glPushMatrix()  # сохраняется текущая матрица проекции на стек матриц
```

```
    glLoadIdentity()  # загружается единичная матрица проекции
    glOrtho(0.0, SCREEN_SIZE[0], SCREEN_SIZE[1], 0.0, 0.0, 1.0)  # устанавливатся ортографическая матрица проекции
    # Ортографическая матрица проекции позволяет рисовать объекты на экране без перспективной деформации.
    glMatrixMode(GL_MODELVIEW)

    glLoadIdentity()
    glDisable(GL_CULL_FACE)  # отключение отсечение граней

    glClear(GL_DEPTH_BUFFER_BIT)

    glMatrixMode(GL_PROJECTION)
    glPopMatrix()
    glMatrixMode(GL_MODELVIEW)


root = Tk()  # главное окно
root.resizable(False, False)

axis = Axis()
camera = Camera()

vertex0_cube = Vertex(1, -1, -1)
vertex1_cube = Vertex(1, 1, -1)
vertex2_cube = Vertex(-1, 1, -1)
vertex3_cube = Vertex(-1, -1, -1)
vertex4_cube = Vertex(1, -1, 1)
vertex5_cube = Vertex(1, 1, 1)
vertex6_cube = Vertex(-1, -1, 1)
vertex7_cube = Vertex(-1, 1, 1)

edges_cube = (Edge(vertex0_cube, vertex1_cube),
        Edge(vertex0_cube, vertex3_cube),
        Edge(vertex0_cube, vertex4_cube),
        Edge(vertex2_cube, vertex1_cube),
        Edge(vertex2_cube, vertex3_cube),
        Edge(vertex2_cube, vertex7_cube),
        Edge(vertex6_cube, vertex3_cube),
        Edge(vertex6_cube, vertex4_cube),
        Edge(vertex6_cube, vertex7_cube),
        Edge(vertex5_cube, vertex1_cube),
        Edge(vertex5_cube, vertex4_cube),
        Edge(vertex5_cube, vertex7_cube),
        )

vertex1_pyramid = Vertex(-1, 0, 0)
vertex2_pyramid = Vertex(0, 0, 1)
vertex3_pyramid = Vertex(1, 0, 0)
```

```
vertex4_pyramid = Vertex(0, 0, -1)
vertex5_pyramid = Vertex(0, 1, 0)

edges_pyramid = (Edge(vertex1_pyramid, vertex5_pyramid),
            Edge(vertex2_pyramid, vertex5_pyramid),
            Edge(vertex3_pyramid, vertex5_pyramid),
            Edge(vertex4_pyramid, vertex5_pyramid),
            Edge(vertex1_pyramid, vertex4_pyramid),
            Edge(vertex3_pyramid, vertex4_pyramid),
            Edge(vertex2_pyramid, vertex3_pyramid),
            Edge(vertex1_pyramid, vertex2_pyramid))

vertexes_circle = []
createCircle(0, 0, 0, 1)
edge_cone = []
vertex_cone = Vertex(0, 1, 0)

for i in range(len(vertexes_circle) - 1):
   temp_vertex1 = Vertex(vertexes_circle[i][0], vertexes_circle[i][2], vertexes_circle[i][1])
   temp_vertex2 = Vertex(vertexes_circle[i + 1][0], vertexes_circle[i + 1][2], vertexes_circle[i + 1][1])
   edge_cone.append(Edge(temp_vertex1, temp_vertex2))
   if i % 5 == 0:
      edge_cone.append(Edge(temp_vertex1, vertex_cone))

edge_cylinder = []
temp_circle = vertexes_circle.copy()
vertexes_circle.clear()
createCircle(0, 0, 1, 1)

for i in range(len(temp_circle) - 1):
   temp_vertex1 = Vertex(temp_circle[i][0], temp_circle[i][2], temp_circle[i][1])
   temp_vertex2 = Vertex(temp_circle[i + 1][0], temp_circle[i + 1][2], temp_circle[i + 1][1])

   temp1_vertex1 = Vertex(vertexes_circle[i][0], vertexes_circle[i][2], vertexes_circle[i][1])
   temp1_vertex2 = Vertex(vertexes_circle[i + 1][0], vertexes_circle[i + 1][2], vertexes_circle[i + 1]
[1])

   edge_cylinder.append(Edge(temp_vertex1, temp_vertex2))
   edge_cylinder.append(Edge(temp_vertex1, temp1_vertex1))
   edge_cylinder.append(Edge(temp1_vertex1, temp1_vertex2))

cylinder = Figure(edge_cylinder, [1, 0, 0], [-2, 2, -6, 1, 1.5, 0, 0, 1, 0])
cone = Figure(edge_cone, [0, 100, 0], [-5, 2, -6, 3, -1.5, 0, 0, 1, 0])
cube = Figure(edges_cube, [1, 0, 1], [-10, 2, -6, -1.6499988, -1.77999955, 0, 0, 1, 0])
pyramid = Figure(edges_pyramid, [1, 25, 0], [-2, 2, -6, -1.9699985, 0.36999992, 0, 0, 1, 0])

figures = [cube, pyramid, cone, cylinder]
#figures = []
figure_id = 0
```

```
window_width = 800  # размеры окна (ширина и высота)
window_height = 600
SCREEN_SIZE = (window_width, window_height)

app = DrawingWindow(root, width=window_width, height=window_height)  # создание окна для отрисови
app.pack(fill=BOTH, expand=YES)  # отобразить

root.bind('a', camera_plus_tx)
root.bind('d', camera_minus_tx)
root.bind('w', camera_plus_tz)
root.bind('s', camera_minus_tz)
root.bind('q', camera_plus_ty)
root.bind('e', camera_minus_ty)
root.bind('<Right>', camera_plus_ry)
root.bind('<Left>', camera_minus_ry)
root.bind('<Up>', camera_minus_rx)
root.bind('<Down>', camera_plus_rx)
root.bind('x', camera_minus_rz)
root.bind('z', camera_plus_rz)
root.bind('<KeyRelease>', camera_stop)

label_select_figure = Label(text="Выбор фигуры")
label_select_figure.place(x=685, y=0)
combo_select_figure = Combobox(app)
combo_select_figure['values'] = ("Куб", "Пирамида", "Конус", "Цилиндр")
combo_select_figure['state'] = 'readonly'
combo_select_figure.current(0)
combo_select_figure.place(x=655, y=25)

label_select_axis = Label(text="Перемещение по осям")
label_select_axis.place(x=660, y=50)

label_select_x = Label(text="x", width=2, height=1)
label_select_x.place(x=665, y=77)
button_plus_axes_x = Button(app, text="+", width=4)
button_plus_axes_x.place(x=690, y=75)
button_minus_axes_x = Button(app, text="-", width=4)
button_minus_axes_x.place(x=730, y=75)
button_plus_axes_x.bind('<Button-1>', plus_axes_x)
button_plus_axes_x.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_x.bind('<Button-1>', minus_axes_x)
button_minus_axes_x.bind('<ButtonRelease-1>', figure_stop)

label_select_y = Label(text="y", width=2, height=1)
label_select_y.place(x=665, y=102)
button_plus_axes_y = Button(app, text="+", width=4)
button_plus_axes_y.place(x=690, y=100)
```

```
button_minus_axes_y = Button(app, text="-", width=4)
button_minus_axes_y.place(x=730, y=100)
button_plus_axes_y.bind('<Button-1>', plus_axes_y)
button_plus_axes_y.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_y.bind('<Button-1>', minus_axes_y)
button_minus_axes_y.bind('<ButtonRelease-1>', figure_stop)

label_select_z = Label(text="z", width=2, height=1)
label_select_z.place(x=665, y=127)
button_plus_axes_z = Button(app, text="+", width=4)
button_plus_axes_z.place(x=690, y=125)
button_minus_axes_z = Button(app, text="-", width=4)
button_minus_axes_z.place(x=730, y=125)
button_plus_axes_z.bind('<Button-1>', plus_axes_z)
button_plus_axes_z.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_z.bind('<Button-1>', minus_axes_z)
button_minus_axes_z.bind('<ButtonRelease-1>', figure_stop)

label_select_axis = Label(text="Вращение по осям")
label_select_axis.place(x=665, y=155)

label_select_rx = Label(text="x", width=2, height=1)
label_select_rx.place(x=665, y=182)
button_plus_axes_rx = Button(app, text="+", width=4)
button_plus_axes_rx.place(x=690, y=180)
button_minus_axes_rx = Button(app, text="-", width=4)
button_minus_axes_rx.place(x=730, y=180)
button_plus_axes_rx.bind('<Button-1>', plus_axes_rx)
button_plus_axes_rx.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_rx.bind('<Button-1>', minus_axes_rx)
button_minus_axes_rx.bind('<ButtonRelease-1>', figure_stop)

label_select_ry = Label(text="y", width=2, height=1)
label_select_ry.place(x=665, y=207)
button_plus_axes_ry = Button(app, text="+", width=4)
button_plus_axes_ry.place(x=690, y=205)
button_minus_axes_ry = Button(app, text="-", width=4)
button_minus_axes_ry.place(x=730, y=205)
button_plus_axes_ry.bind('<Button-1>', plus_axes_ry)
button_plus_axes_ry.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_ry.bind('<Button-1>', minus_axes_ry)
button_minus_axes_ry.bind('<ButtonRelease-1>', figure_stop)

label_select_rz = Label(text="z", width=2, height=1)
label_select_rz.place(x=665, y=232)
button_plus_axes_rz = Button(app, text="+", width=4)
button_plus_axes_rz.place(x=690, y=230)
button_minus_axes_rz = Button(app, text="-", width=4)
button_minus_axes_rz.place(x=730, y=230)
```

```
button_plus_axes_rz.bind('<Button-1>', plus_axes_rz)
button_plus_axes_rz.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_rz.bind('<Button-1>', minus_axes_rz)
button_minus_axes_rz.bind('<ButtonRelease-1>', figure_stop)

label_select_axis = Label(text="Масштабирование")
label_select_axis.place(x=665, y=260)
label_select_sxyz = Label(text="xyz", width=3, height=1)
label_select_sxyz.place(x=665, y=285)
button_plus_axes_sxyz = Button(app, text="+", width=4)
button_plus_axes_sxyz.place(x=695, y=285)
button_minus_axes_sxyz = Button(app, text="-", width=4)
button_minus_axes_sxyz.place(x=735, y=285)
button_plus_axes_sxyz.bind('<Button-1>', plus_axes_sxyz)
button_plus_axes_sxyz.bind('<ButtonRelease-1>', figure_stop)
button_minus_axes_sxyz.bind('<Button-1>', minus_axes_sxyz)
button_minus_axes_sxyz.bind('<ButtonRelease-1>', figure_stop)

fill_figure = Button(app, text="Закрашивание", command=click_fill_figure)
fill_figure.place(x=695, y=570)

app.animate = 1
app.mainloop()
```