**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ**

**Пояснительная записка к расчетно-графической работе**

Группа: АВТ-441 Студент: Баркалов Н. И.

**г. Новосибирск 2024**

**Оглавление**

1. Постановка задачи [3](#_bookmark0)
2. Основная идея [3](#_bookmark1)
3. Основные методы решения [3](#_bookmark2)
4. Принципы работы [5](#_bookmark3)
5. Демонстрация работы и тесты 8
6. Выводы 15
7. Список литературы 16

# Постановка задачи

Задача: смоделировать движение многоугольников по поверхности шара и проекцию этого движения на 2D-плоскость.

# Основная идея

Для решения задачи она разбита на следующие подзадачи:

1. Движение фигур в пространстве по круговой орбите путём поворота векторов.
2. Разбиение фигур на произвольное количество точек.
3. Проекция каждой точки на поверхность шара.
4. Проекция на плоскость.
   1. **Геометрическая модель**

Для решения задачи задействована система координат, в которой шар – единичная сфера с центром в (0, 0, 0). Проекция точек, находящейся на ней, происходит на плоскость z=-2, она же становится тем, что мы увидим на экране.

# Основные методы решения

* 1. **Линейная алгебра**

В решении задачи используется представление точек как векторов, выходящих из начала координат O (0, 0, 0) и операции над ними: поворот, проектирование.

1. **Поворот и движение по эллипсоидной (круговой) орбите.** Поворот или вращение векторов производится путём выполнения собственного ортогонального преобразования в евклидовом пространстве с использованием матриц поворота (матриц направляющих косинусов) в трёхмерном пространстве [2]. Матрицы являются ортогональными и имеют следующий вид:  
   **1. Поворот вокруг оси Ox на угол ϕ [2]:  
     
   2. Поворот вокруг оси Oy на угол ϕ [2]:  
     
   3. Поворот вокруг оси Oz на угол ϕ [2]:**

**4. Матрица поворота последовательности поворотов в некотором оговорённом порядке:**

1. **Разбиение фигур на множество точек.** Для осуществления проекции фигуры на шар, отрезки, из которых состоит фигура, разбиваются на произвольное количество точек, находящихся на одном расстоянии друг от друга, каждая из которых проецируется отдельно. Разбиение отдельно взятого отрезка происходит по следующей формуле [3]:  
     
   где A и B – вершины разбиваемого отрезка, λ – пропорции, в которых точка M разбивает отрезок относительно точки B.
2. **Проекция точек на поверхность шара.** Проекцией точки на шар будет являться точка, в которую будет опущен радиус шара параллельно радиус-вектору проецируемой точки. Так как векторы будут коллинеарны друг другу, то один можно выразить через другой, умножив на некоторый коэффициент k, который можно найти, зная длину исходного векторов [4].
3. **Центральная проекция на плоскость.** Центральная проекция позволяет учесть перспективу. Для её осуществления проводится прямая через проецируемую точку M (xm, ym, zm) и некоторый «центр» - точку C (x0, y0, z0). Проекцией будет являться точка пересечения прямой MC и плоскости экрана P (в рамках данной работы – плоскость z=-2) [5]. Для поиска прямой MC используются канонические уравнения, приводимые к параметрическим.

Подставив значения x, y и z в уравнение плоскости, получим следующую формулу для нахождения параметра t. Координаты найдём подстановкой этого параметра в параметрические уравнения, указанные выше.

* 1. **Методы решения в программировании**

Для решения данной задачи была использована библиотеку Arcade [[1]](#_bookmark5) на языке программирования Python, так как она предоставляет удобные методы для реализации 2D-графики. Выбор же языка обусловлен лёгкой работой с массивами и математическими объектами: кортежами, векторами. Также, организация работы с памятью в Python позволяет сосредоточиться на построении модели, избегая прямой с ней работы. Arcade позволяет строить на экране линии, многоугольники.

1. **Принцип работы**
   1. **Установка компонентов**

Разработка проходит на операционной системе Windows 10, в среде – JetBrains PyCharm, с подключённым менеджером пакетов и библиотек – pip. Установка библиотеки Arcade [[1]](#_bookmark5) произведена с помощью pip, путём ввода следующей команды в консоль Windows:

“pip install arcade”.

* 1. **Разработка**

В начале импортируются необходимые библиотеки и дополнительно написанный модуль «logic.py», содержащий методы для работы с математическими преобразованиями векторов. Здесь же определяются константы: размер окна, название – и глобальный таймер.

import arcade # Импортируем модуль arcade для реализации 2D-графики  
from logic import \* # Импортируем функции, связанные с математическими преобразованиями  
  
# Устанавливаем константы  
  
# Размер экрана  
SCREEN\_WIDTH = 600 # Ширина окна  
SCREEN\_HEIGHT = 600 # Высота окна  
SCREEN\_TITLE = "Sphere projection - rgr" # Имя окна  
  
timer = 0.0 # Глобальная переменная таймера, которая будет повышаться на 0.01 каждый шаг

# Запрашиваем у пользователя номер теста  
number\_of\_test = int(input("Введите номер теста: "))

Объявлена функция для проведения ряда преобразований над векторами в необходимом порядке: поворот, разбиение, проекция на шар, центральная проекция на плоскость. Возвращает массив, содержащий получившиеся в результате точки.

def process(vertices, angles):  
 """Функция проводит ряд преобразований: поворот, разбиение, проектирование на поверхность  
 сферы и центральную проекцию на плоскость. На вход принимает массив исходных вершин  
 фигуры vertices и массив, состоящий из трёх углов поворота a, b и c вокруг осей  
 Ox, Oy и Oz соответственно"""  
 for dot in vertices: # Перебираем все вершины фигуры, поворачиваем их  
 dot[0], dot[1], dot[2] = rotate(dot, angles[0], angles[1], angles[2])  
  
 # Объявляем новый массив, состоящий из вершин многоугольника и точек между ними  
 vertices\_gotten = find\_dots(vertices)  
  
 for dot in vertices\_gotten: # Проецируем каждую точку в новом массиве на сферу  
 dot[0], dot[1], dot[2] = projection\_on\_sphere(dot)  
  
 for dot in vertices\_gotten: # Поиск центральных проекций на плоскость z=-2  
 dot[0], dot[1], dot[2] = central\_projection\_on\_surface(dot)  
  
 dots = vertices\_gotten # Для удобства объявляем новый массив, содержащий точки после всех преобразований  
  
 for dot in vertices\_gotten: # В Arcade отрисовка идёт от левого нижнего края (I часть ПДСК)  
 dot[0] \*= 100 # Чтобы визуализировать более наглядно, увеличиваем координаты  
 dot[0] += 300 # точек в 100 раз и центруем их, прибавляя 300  
 dot[1] \*= 100  
 dot[1] += 300  
  
 return vertices\_gotten

Далее объявляется функция “on\_draw”, внутри которой происходит обработка движения фигуры и непосредственно её отрисовка по точкам:

def on\_draw(delta\_time): # Функция, внутри которой производятся все преобразования фигур и их отрисовка  
 global timer # Добавляем глобальную переменную timer в область видимости on\_draw  
  
 timer += 0.01 # Увеличиваем значение таймера  
  
 if number\_of\_test == 7: # Данное условие реализует выбор номера теста, в каждом из которых  
 vertices = [[0, 2, 2], [2, 2, 0], [0, 2, -2], [-2, 2, 0]] # Объявляются массивы с вершинами фигур  
 vertices2 = [[0, 2, 2], [sqrt(3), 2, -1], [-sqrt(3), 2, -1]]  
 vertices3 = [[1, 1, 3], [-2, 1, 3], [-2, -1, 3]]  
 dots1 = process(vertices, (timer, timer, 0)) # Проводятся их преобразование  
 dots2 = process(vertices2, (timer, timer, timer))  
 dots3 = process(vertices3, (timer, 5, 5))  
 # Объявляется массив, содержащий все обрабатываемые фигуры  
 figures = [dots1, dots2, dots3]  
 elif number\_of\_test == 6:  
 vertices = [[0, 2, 2], [2, 2, 0], [0, 2, -2], [-2, 2, 0]] # Объявляются массивы с вершинами фигур  
 vertices2 = [[0, 2, 2], [sqrt(3), 2, -1], [-sqrt(3), 2, -1]]  
 dots1 = process(vertices, (timer, 0, 0)) # Проводятся их преобразование  
 dots2 = process(vertices2, (timer, timer, 0))  
 figures = [dots1, dots2]  
 elif number\_of\_test == 5:  
 # Объявляются массивы с вершинами фигур  
 vertices = [[3, 6, -3], [4.5, 6, 0], [3, 6, 3], [-3, 6, 3], [-4.5, 6, 0], [-3, 6, -3]]  
 dots = process(vertices, (timer, 0, 3.1415/4)) # Проводятся их преобразование  
 figures = [dots]  
 elif number\_of\_test == 4:  
 vertices = [[3, 0, 3], [-3, 0, 3], [-3, 0, 6], [3, 0, 6]] # Объявляются массивы с вершинами фигур  
 dots = process(vertices, (timer, 0, 0)) # Проводятся их преобразование  
 figures = [dots]  
 elif number\_of\_test == 3:  
 vertices = [[2, 0, 2], [0, 2, 2], [-2, 0, 2], [0, -2, 2]] # Объявляются массивы с вершинами фигур  
 dots = process(vertices, (0, timer, 0)) # Проводятся их преобразование  
 figures = [dots]  
 elif number\_of\_test == 2:  
 vertices = [[0, 2, 2], [sqrt(3), 2, -1], [-sqrt(3), 2, -1]] # Объявляются массивы с вершинами фигур  
 dots = process(vertices, (0, timer, 0)) # Проводятся их преобразование  
 figures = [dots]  
 elif number\_of\_test == 1:  
 vertices = [[0, 2, 2], [2, 2, 0], [0, 2, -2], [-2, 2, 0]] # Объявляются массивы с вершинами фигур  
 dots = process(vertices, (timer, 0, 0)) # Проводятся их преобразование  
 figures = [dots]  
  
 k = 0 # Вспомогательная переменная k, хранящая то, какую сейчас фигуру мы обрабатываем  
 arcade.start\_render()  
 arcade.draw\_circle\_filled(300, 300, 232, [255, 255, 255])  
 for dots in figures:  
 # Массив цветов фигур, 1 будет отрисована красным, 2 - зелёным, 3 - синим  
 c = ([92, 0, 0], [1, 50, 32], [0, 33, 55])[k]  
 k += 1  
 l = len(dots) # Для удобства объявляем переменную l - количество отрисовываемых точек  
 # Начинаем рендер  
 for i in range(l): # Перебираем все точки в dots, рисуя линии от заданной до следующей  
 arcade.draw\_line(dots[i][0], # Ниже с помощью тернарного оператора проверяем является ли точка  
 dots[i][1], # последней в массиве - в таком случае соединяем её с первой  
 dots[i + 1][0] if i != l - 1 else dots[0][0],  
 dots[i + 1][1] if i != l - 1 else dots[0][1],  
 c, # Цвет линии, заданный в RGB  
 2) # Толщина линии

После объявления функции вызываем методы, проводящие настройку и запуск отрисовки.

arcade.open\_window(SCREEN\_WIDTH, SCREEN\_HEIGHT, SCREEN\_TITLE) # Открываем окно, настраиваем его ширину, высоту и название  
arcade.set\_background\_color([73, 66, 63]) # Устанавливаем белый фон  
  
# Периодически вызываем команду отрисовки каждую 1/80 секунды  
arcade.schedule(on\_draw, 1/80)  
  
# Запускаем цикл отрисовки  
arcade.run()

**4.3 Вспомогательные функции для работы с векторами**

Импортируем необходимые для взятия синуса, косинуса и квадратного корня функции [[2]](#_bookmark5)

from math import cos, sin, sqrt # импортируем необходимые функции из math

Функция последовательного поворота вектора на углы a, b и c

def rotate(vector, a=0, b=0, c=0):  
 """Последовательно вращает векторы из массива vector вокруг осей OX(угол a), OY(угол b), OZ(угол c)"""  
 new\_vector = [None, None, None]  
 # Изменение x-координаты  
 new\_vector[0] = vector[0] \* cos(b) \* cos(c) - vector[1] \* sin(c) \* cos(b) + vector[2] \* sin(b)  
 # Изменение y-координаты  
 new\_vector[1] = vector[0] \* (sin(a) \* sin(b) \* cos(c) + sin(c) \* cos(a))  
 new\_vector[1] += vector[1] \* (-sin(a) \* sin(b) \* sin(c) + cos(a) \* cos(c))  
 new\_vector[1] += vector[2] \* -sin(a) \* cos(b)  
 # Изменение Z-координаты  
 new\_vector[2] = vector[0] \* (sin(a) \* sin(c) - sin(b) \* cos(a) \* cos(c))  
 new\_vector[2] += vector[1] \* (sin(a) \* cos(c) + sin(b) \* sin(c) \* cos(a))  
 new\_vector[2] += vector[2] \* cos(a) \* cos(b)  
 # Возвращаем полученный вектор  
 return new\_vector

Функция разбиения отрезков на равноудалённые точек

def find\_dots(vectors, n=15):  
 """возвращает массив из (n+1)\*len(vectors) равноудалённых точек,  
 принадлежащих отрезкам между данными точками"""  
 new\_dots = []  
 # данное выражение помогает создать двумерный массив размера Nx2 вида [(0,1),(1,2)...(n-1,n),(n,0)]  
 for i in [(k, k+1) if k < len(vectors)-1 else (k, 0) for k in range(len(vectors))]:  
 for n in range(16):  
 lm = n / (16-n)  
 new\_dots.append([  
 (vectors[i[0]][j] + lm \* vectors[i[1]][j]) / (lm + 1) for j in range(3)  
 ])  
 return new\_dots

Функция проецирования точки на поверхность сферы. Принимает массив из трёх координат и радиус сферы с центром в (0, 0, 0) (по умолчанию 1)

def projection\_on\_sphere(dot, r=1):  
 """Возвращает вектор длины r, сонаправленный данному"""  
 new\_dot = dot.copy()  
 k = r / sqrt(dot[0] \*\* 2 + dot[1] \*\* 2 + dot[2] \*\* 2)  
 new\_dot[0], new\_dot[1], new\_dot[2] = dot[0] \* k, dot[1] \* k, dot[2] \* k  
 return new\_dot

Функция поиска центральной проекции на плоскость

def central\_projection\_on\_surface(dot, center=[0, 0, 2], A=0, B=0, C=1, D=2):  
 """Возвращает центральную проекцию на плоскость, заданную уравнением  
 Ax+By+Cz+D=0, если не указаны коэффициенты, то на плоскость z=-2, считая  
 центром точку center"""  
 # Рассчитаем координаты направляющего вектора прямой для уравнений  
 # (x-x0)/p=(y-y0)/q=(z-z0)/m  
 p = center[0] - dot[0]  
 q = center[1] - dot[1]  
 m = center[2] - dot[2]  
 # Объявим координаты точки, как x0, y0, z0  
 x0, y0, z0 = dot  
 # Из уравнения плоскости Ax+By+Cz+D=0, куда мы подставим правые части параметрических уравнений  
 # x=x0+pt; y=y0+qt; z=z0+mt, найдём t = (-D-Cz0-By0-Ax0)/(Ap+Bq+Cm)  
 t = (-D-C\*z0-B\*y0-A\*x0)/(A\*p+B\*q+C\*m)  
 return [  
 x0+p\*t,  
 y0+q\*t,  
 z0+m\*t  
 ]

# 5. Демонстрация работы и тесты

Ниже находятся тесты работы программы, демонстрирующие её работу в разных ситуациях.

**Тест 1.** Запущена симуляция движения квадрата с координатами вершин ([0, 2, 2], [2, 2, 0], [0, 2, -2], [-2, 2, 0]) и движение его по круговой орбите вокруг оси Ox. Построен график зависимости координаты (изначально (2, 2, 0)) от угла поворота.

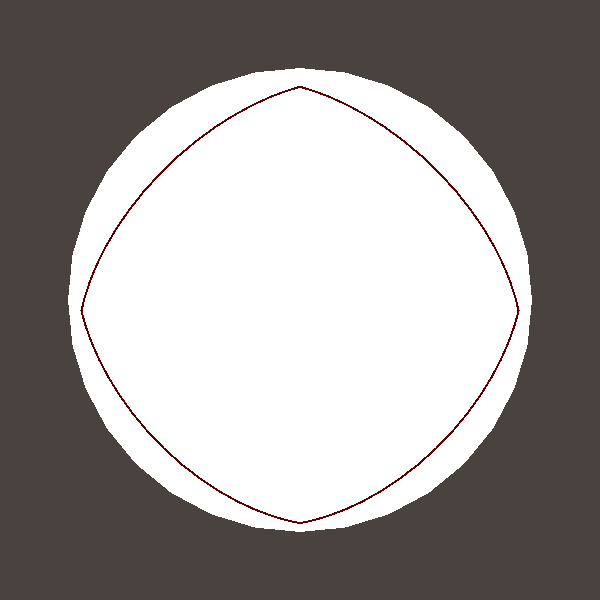
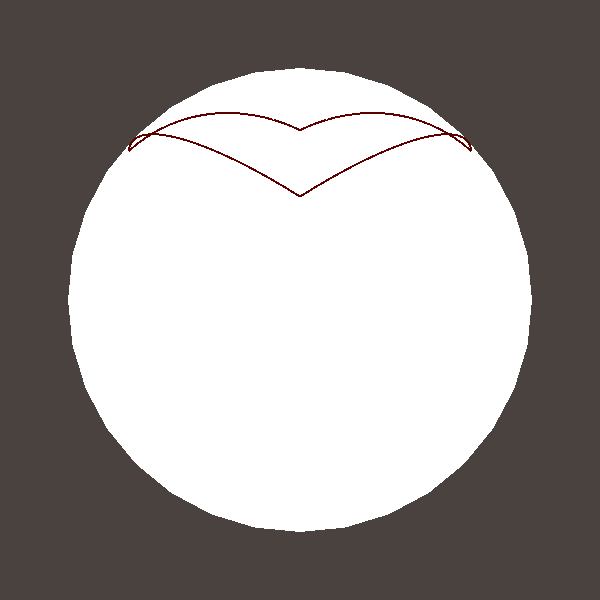
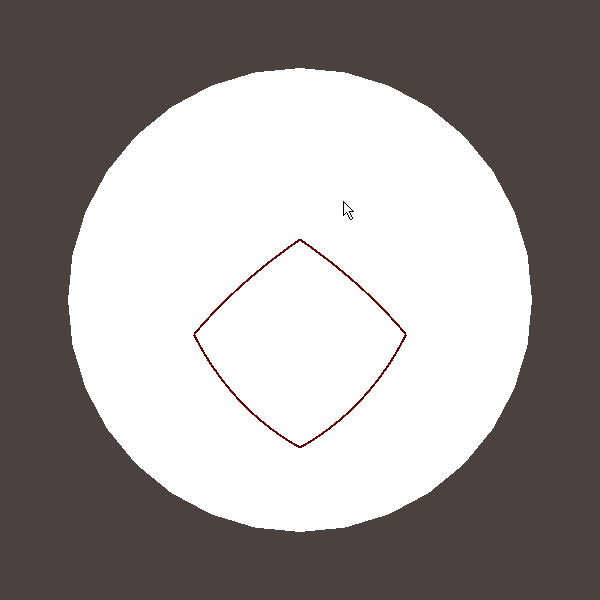


Рисунок 1 – тест 1.

Рисунок 2 – тест 1.

Как видно из графика, координаты меняются по закону синуса (откуда следует круговая орбита движения). При повороте вокруг Ox координата x остаётся неизменной. Тест успешен.

**Тест 2.** Треугольник с координатами ([0, 2, 2], [√3, 2, -1], [-√3, 2, -1]). И его движение по круговой орбите вокруг оси Oy. Построен график зависимости координаты (√3, 2, -1) от угла поворота.

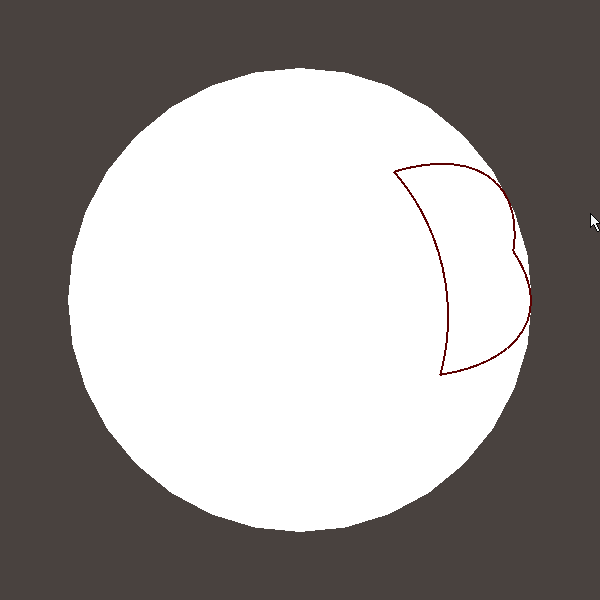
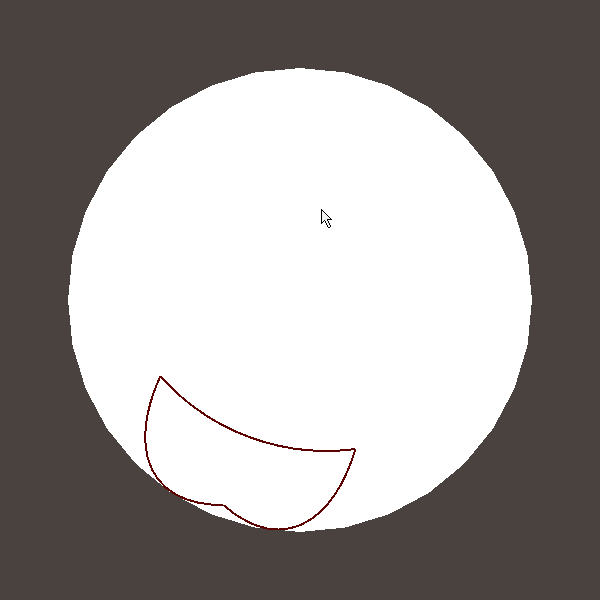


Рисунок 3 – тест 2.

Рисунок 4 – тест 2.

Тест показал, что фигура может двигаться и по периметру видимого круга. Как видно из графика, координаты меняются по закону синуса (откуда следует круговая орбита движения). При повороте вокруг Oy координата y остаётся неизменной. Тест успешен.

**Тест 3.** Движение квадрата с координатами ([2, 0, 2], [0, 2, 2], [-2, 0, 2], [0, -2, 2]) по круговой орбите вокруг оси Oy.

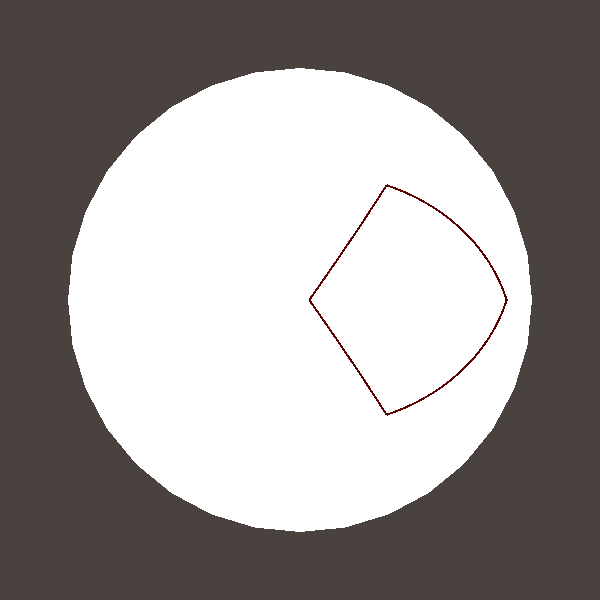
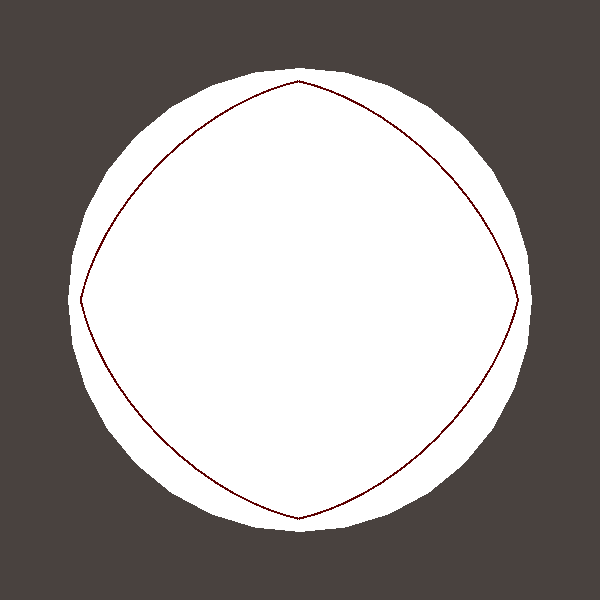
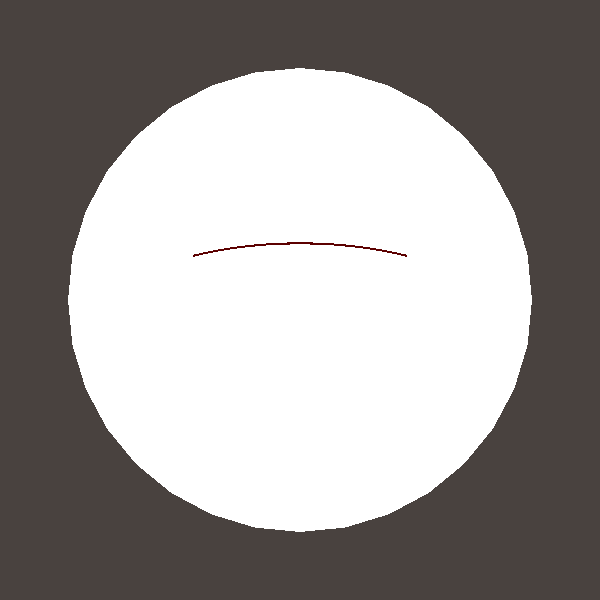
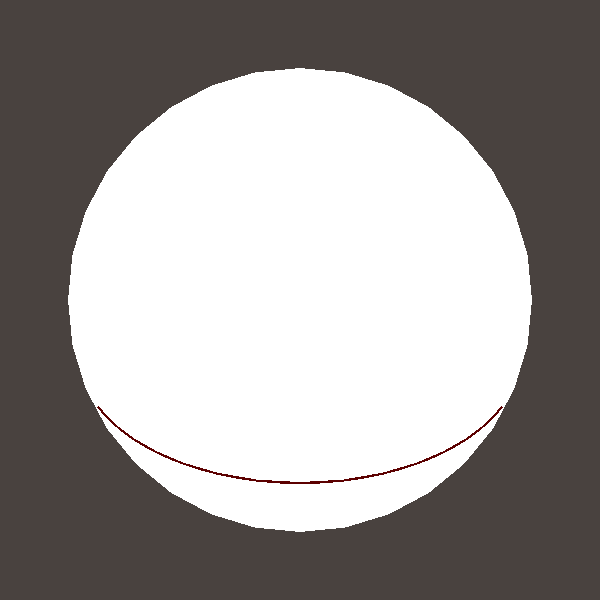


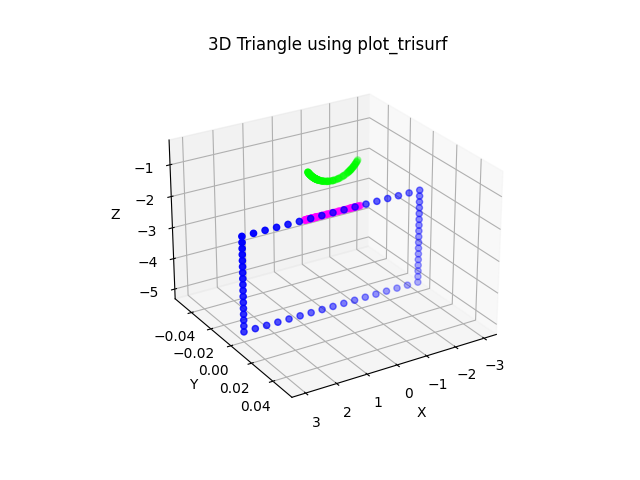
Рисунок 5 – тест 3.

Рисунок 6 – тест 3.

Тест показал, что фигуры могут двигаться вдоль любой оси из любой стартовой координаты (В ПДСК, где Oz направлена вверх, данный квадрат находится выше сферы). График показывает зависимость координат точки (2, 0, 2) от угла поворота (в радианах). Тест успешен.

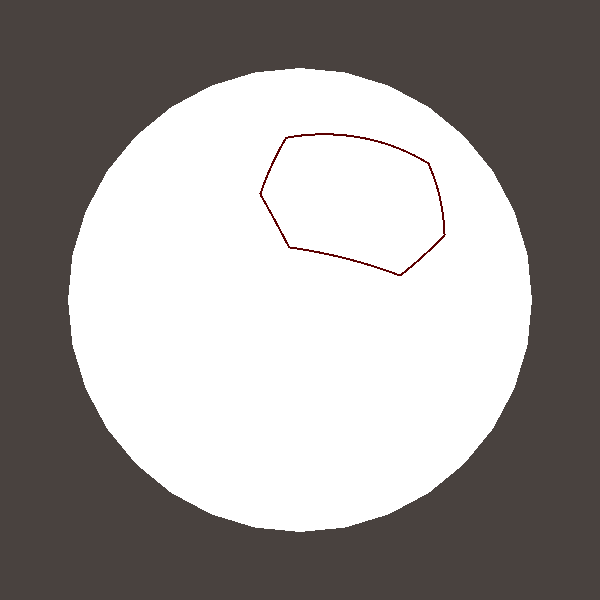
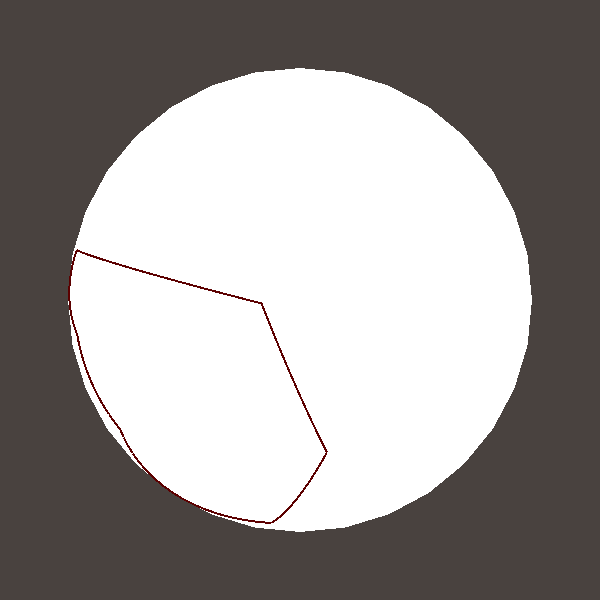
**Тест 4.** Треугольник с координатами ([3, 0, 3], [-3, 0, 3], [-3, 0, 6], [3, 0, 6]) движется по круговой орбите вокруг оси Ox. На трёхмерном графике отображён процесс проецирования точек фигур. Синие точки – исходная фигура, разбитая на точки и повёрнутая на угол в 1800, зелёные – проекции этих точек на поверхность шара, малиновые – центральные проекции на плоскость z=-2.

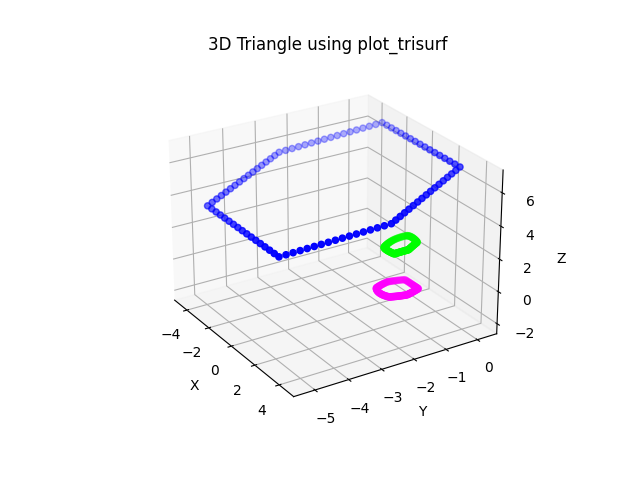
  
Рисунок 7 – тест 4.

Рисунок 8 – тест 4.

Тест показывает возможность сценария, при котором некоторая некоторая фигура проецируется в отрезок на поверхности шара.

**Тест 5.** Движение 6-угольника с координатами ([3, 6, -3], [4.5, 6, 0], [3, 6, 3], [-3, 6, 3], [-4.5, 6, 0], [-3, 6, -3]) вокруг Ox по круговой орбите, отклонённой от оси Oz на 450. На графике изображён процесс проецирования. Синие точки – исходный шестиугольник, повёрнутый на 2 радиана вокруг OX, зелёные – проекции на поверхность шара, малиновые – на плоскость z=-2 относительно центра с координатами (0, 0, 11).

  
Рисунок 9 – тест 5.

Рисунок 10 – тест 5.

Тест показывает, что можно симулировать движение фигур с большим количеством вершин, в то числе многоугольников, ломанных. Если многоугольник правильный и у него большое количество вершин, то его проекция будет стремиться к окружности. Он же показывает, что возможно отклонение орбиты на фиксированный угол. Тест успешен.

**Тест 6.** Движение двух фигур: квадрата с координатами ([0, 2, 2], [2, 2, 0], [0, 2, -2], [-2, 2, 0]) вокруг оси Ox и треугольника с координатами ([0, 2, 2], [√3, 2, -1], [-√3, 2, -1]) вдоль плоскости x+y=0 (задаются два угла поворота: альфа и бета). На графике изображены: исходный второй треугольник (красный), треугольник после поворота на углы в 2 радиана по Ox и Oy и разбитый на точки (синие точки), проекции этих точек на поверхность шара (зелёные) и на плоскость z=-2 относительно центра (0, 0, 11) (малиновые).

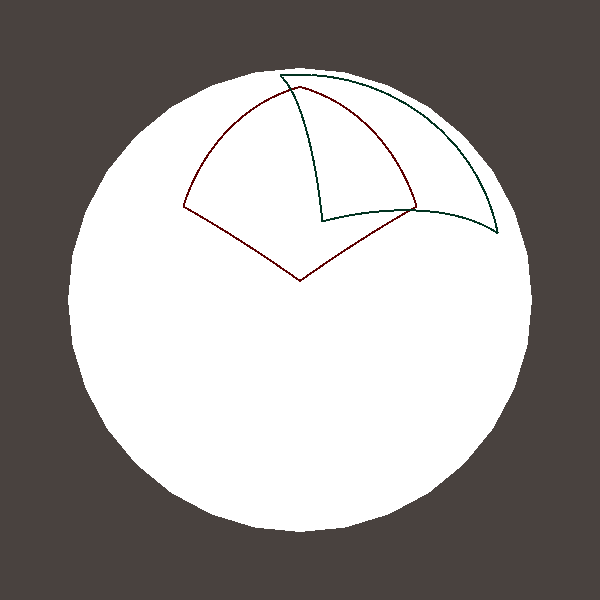
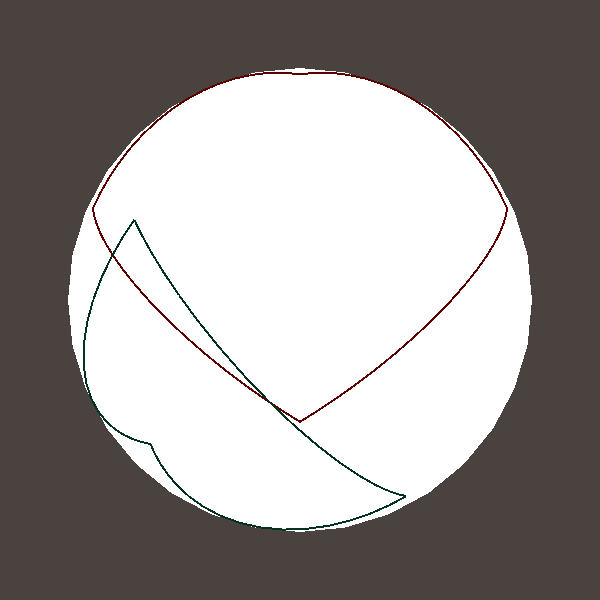
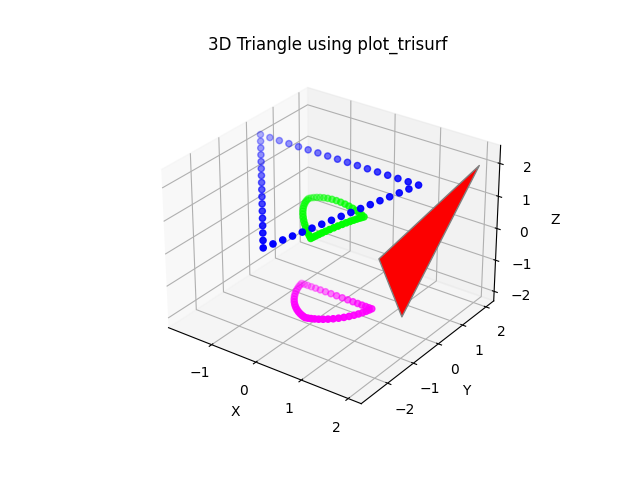
  
Рисунок 11 – тест 6.

Рисунок 12 – тест 6.

Тест показывает возможность запуска двух фигур и задания им разных траекторий движения. Тест успешен.

**Тест 7.** Движение 3-х фигур по различным орбитам. Координаты треугольника и квадрата взяты из прошлого теста. Траектория движения треугольника – окружность в плоскости x+y+z=0 (заданы все три угла вращения), а квадрата – окружность x+y=0 (траектория движения треугольника из прошлого теста). Как третья фигура вводится треугольник с координатами ([1, 1, 3], [-2, 1, 3], [-2, -1, 3]), рисуемый синим цветом. На графике изображено взаимное расположение двух фигур и их проекций: третьего и второго треугольников. Исходные треугольники – красный и голубой, они же, повёрнутые на углы в 2 радиана по своим орбитам – синий и жёлтые, их проекции на поверхность шара – зелёные и чёрные точки, проекции на плоскость z=-2 – малиновые и оранжевые.

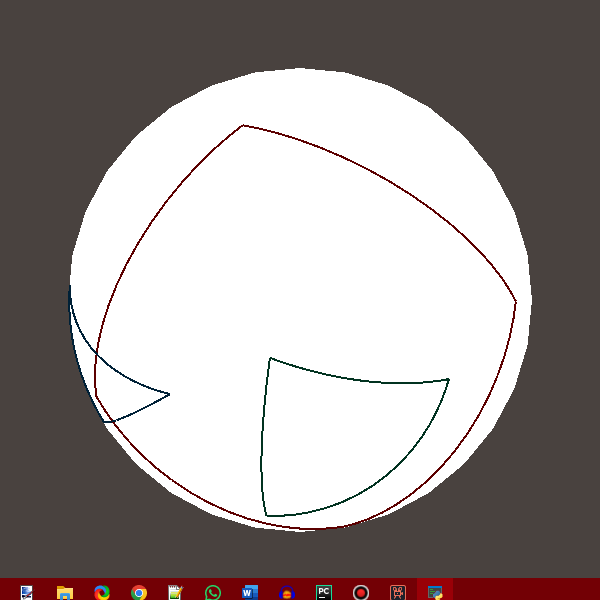
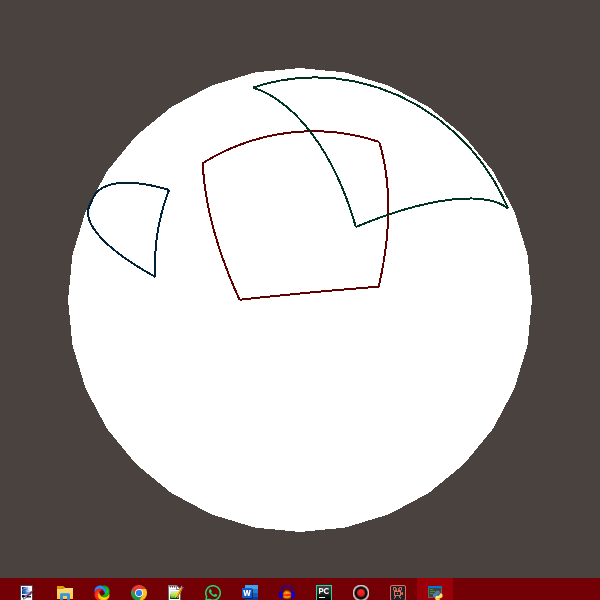


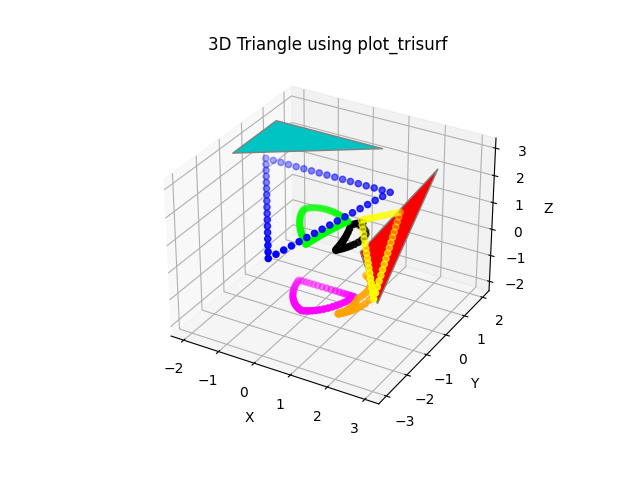
Рисунок 13 – тест 7.

Рисунок 14 – тест 7.

Тест ещё раз доказал возможность запуска неограниченного числа фигур в симуляции. Тест успешен.

# 6. Выводы

Была разработана программа, симулирующая движение многоугольников по поверхности шара.

В ходе тестирования программы не было выявлено ни ошибок, ни сбоев; все тесты завершились успешно. Программа корректно отображает движение объектов по поверхности шара, учитывая перспективу.

# Список литературы

## The Python Arcade Library [<https://api.arcade.academy/en/latest/>]. Дата обращения – 27.11.2024.

* 1. **Матрица поворота**

[<https://ru.wikipedia.org/wiki/Матрица_поворота>]

Дата обращения – 29.11.2024.

* 1. **Деление отрезка в данном отношении** [<http://www.mathprofi.ru/delenie_otrezka_v_dannom_otnoshenii.html>].  
     Дата обращения – 31.11.2024.
  2. **Finding collinear vector given length** [<https://math.stackexchange.com/questions/700681/finding-collinear-vector-given-length>].   
     Дата обращения – 01.12.2024.
  3. **Методы проецирования**[<https://cadinstructor.org/ng/lectures/1-metody-proecirovaniya/>]Дата обращения – 02.12.2024