ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ _____

ПРЕПОДАВАТЕЛЬ

канд. техн. наук, доцент		А.В. Аграновский
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №8

Знакомство с OpenGL

по курсу: КОМПЬЮТЕРНАЯ ГРАФИКА

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4329		Д.С. Шаповалова
	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

Содержание

1. Цель работы:

Изучение открытой графической библиотеки OpenGL; построить динамическую 3D-сцену на языке программирования высокого уровня, поддерживающего библиотеку OpenGL.

2. Задание:

С использованием любого языка программирования высокого уровня (из числа изученных в ходе освоения ООП 09.03.02), поддерживающего библиотеку OpenGL, построить динамическую 3D сцену.

3. Теоретические сведения:

Для работы с «непитоновскими» библиотеками (например, OpenGL) необходимы модули, обеспечивающие возможность вызова функций библиотеки непосредственно из программы на языке Python. Библиотека PyOpenGL — модуль, позволяющий в программах на языке Python легко работать с функциями OpenGL, GLU и GLUT, а также с рядом расширений OpenGL.

3.1 OpenGL в общем

OpenGL (Открытая Графическая Библиотека) представляет собой межплатформенный стандарт, который определяет минимальный набор функциональных возможностей для разработки приложений с использованием двумерной и трёхмерной графики. На данный момент эта спецификация поддерживается на таких платформах, как Mac OS, PlayStation 3, Windows и различные версии Unix. Основная цель OpenGL состоит в описании набора функций и их поведения вне зависимости от используемого языка программирования. Ключевым аспектом работы OpenGL является получение множества графических примитивов (таких как точки, линии и другие), а также дальнейшая математическая обработка этих данных для создания растровых изображений.

Преимущества OpenGL включают:

- Повышение производительности в некоторых играх;
- Улучшение работы видеокарт;
- Наличие расширений;
- Доступность дополнительных библиотек;
- Независимость от языка программирования.

Недостатки OpenGL:

- Поскольку это низкоуровневый интерфейс программирования (API), необходимо заранее тщательно продумать последовательность действий;
- Трудности при работе с новыми аппаратными средствами;

– Необходимость установки и использования DirectX.

Для работы с «непитоновскими» библиотеками (например, OpenGL) необходимы модули, обеспечивающие возможность вызова функций библиотеки непосредственно из программы на языке Python. Библиотека PyOpenGL — модуль, позволяющий в программах на языке Python легко работать с функциями OpenGL, GLU и GLUT, а также с рядом расширений OpenGL.

Строго говоря, когда мы пишем программу, используя OpenGL, мы ничего не рисуем. Мы описываем модель сцены, задаем свойства *примитивов,* из которых состоят все остальные объекты, и управляем состояниями OpenGL. Визуализацией этой модели занимается OpenGL, на основе той информации, которую мы сообщили. Современные графические системы позволяют вмешиваться в процесс визуализации, используя **шейдеры**, что позволяет программировать достаточно гибкие и быстрые графические приложения.

Примитивы OpenGL

Посмотрим, из каких элементов (примитивов) состоит наша сцена. В OpenGL есть три типа примитивов: точка, отрезок и многоугольник. Каждый из этих объектов описывается перечислением своих вершин: координаты точки, концов отрезка или вершин многоугольника.

Четырехугольник (Quad)

На любой многоугольник в OpenGL накладываются ограничения. Первое, многоугольник должен быть *простым*, т.е. не иметь самопересечений. Второе, он должен быть *выпуклым*. В выпуклом многоугольнике отрезок, соединяющий две любые две внутренние точки, не пересекает его границы.

Рисунок 1 – Ограничения многоугольника в OpenGL

3.2 Основные функции OpenGl

В данной лабораторной работе используются функции glPushMatrix() и glPopMatrix() в контексте управления состоянием матриц в OpenGL. Эти функции служат

для сохранения и восстановления текущего состояния матриц, что является важным аспектом при работе с 3D-графикой и трансформациями объектов в сцене.

OpenGL использует матричные методы для работы с трансформациями, такими как перемещение, вращение и масштабирование объектов. Управление состоянием матриц осуществляется через стек матриц, что позволяет эффективно организовывать отрисовку сложных сцен.

glPushMatrix()

Описание: Функция glPushMatrix() помещает текущее состояние выбранной матрицы в стек. Это позволяет сохранить текущее положение и ориентацию трехмерной сцены перед применением новых трансформаций.

Использование:

Для применения трансформаций к объекту, не нарушая состояние других объектов, необходимо сначала вызвать glPushMatrix().

После завершения трансформаций и отрисовки объекта, состояние матрицы может быть восстановлено с помощью glPopMatrix().

glPopMatrix()

Описание: Функция glPopMatrix() извлекает верхнее состояние матрицы из стека и делает его текущим. Это восстанавливает матрицу в состояние, в котором она находилась на момент вызова glPushMatrix().

Использование:

После применения необходимых трансформаций и отрисовки целевого объекта, используется glPopMatrix() для возврата к предыдущему состоянию матрицы.

Это позволяет избежать накопления трансформаций, которые могут случайно повлиять на другие объекты в сцене.

glTranslatef(x, y, z)

Описание: Функция glTranslatef(x, y, z) применяет трансляцию к текущей матрице. Здесь (x, y, z) представляют собой величины, на которые будет происходить смещение объекта по осям X, Y и Z соответственно.

Цель: Основной целью трансляции является изменение позиции объектов в трехмерной сцене, что позволяет создавать динамичные и сложные анимации.

Применение функции glTranslatef(x, y, z)

Исходное состояние: Перед вызовом glTranslatef(x, y, z) в OpenGL выбирается определенное состояние матрицы (мировой, модельной или проекционной). Для трансляции объектов обычно используется модельная матрица.

Эффект трансляции: В результате вызова функции объект перемещается на заданные координаты без изменений в своем размере и ориентации. Это позволяет легко организовывать иерархию объектов, например, перемещая группу объектов как единое целое.

Трансляция является ключевым компонентом 3D-графики, поскольку с ней реализуется:

1. Динамика сцены: Возможно перемещать объекты в ответ на взаимодействие пользователя или в процессе анимации, создавая динамичные и живые сцены.
2. Группировка объектов: Легкость сборки несколько объектов в группы и перемещения их как единое целое, что упрощает управление сложными сценами.
3. Контекст восприятия: Смена перспективы и положения наблюдателя относительно объектов, что является важным для создания иллюзии глубины.

glRotatef(angle, x, y, z)

Описание: Функция glRotatef(angle, x, y, z) принимает угол вращения в градусах и трехмерный вектор (x, y, z), который задает ось вращения. Вектор должен быть нормализован, чтобы корректно применить вращение.

Цель: Основной целью вращения является возможность взаимодействовать с объектами, изменяя их ориентацию, что важно для создания реалистичных сцен и анимаций.

Применение функции glRotatef(angle, x, y, z)

Исходное состояние: Обычно перед вызовом glRotatef(angle, x, y, z) в OpenGL выбрана модельная матрица. Это позволяет изменять ориентацию объектов в трехмерном пространстве.

Эффект вращения: При применении функции объект поворачивается вокруг заданной оси на указанный угол. Это позволяет создавать эффекты вращения и анимации.

Использование функции glRotatef(angle, x, y, z) для вращения объектов играет важную роль в 3D-графике, поскольку она предоставляет следующие возможности:

1. Анимации и динамика: Создание динамических сцен, где объекты могут поворачиваться и изменять своё положение в пространстве, делая их более захватывающими.
2. Сложные движения: Комбинирование вращения с другими преобразованиями, такими как смещение и масштабирование, для создания сложных визуальных эффектов.

3. Интерактивность: Обеспечение взаимодействия между объектами в трёхмерном пространстве, например, поворот модели вокруг её оси в ответ на действия пользователя.

gluNewQuadric()

Описание: Функция gluNewQuadric() создает и возвращает указатель на объект типа GLUquadric. Этот объект содержит различные параметры для визуализации квадриков, включая методы отрисовки и обработку нормалей.

Назначение: Основной задачей использования квадриков является облегчение процесса рисования сложных трехмерных фигур. Это позволяет разработчикам быстрее создавать объемные объекты в сцене, не требуя ручного написания всех необходимых инструкций для их отображения.

Применение функции gluNewQuadric()

Инициализация объекта: Прежде чем начать работу с квадриком, нужно вызвать функцию gluNewQuadric(), которая создаст новый объект GLUquadric и вернет его указатель. После этого объект можно настроить и использовать для отрисовки.

Конфигурация параметров: С помощью полученного объекта ваша программа сможет задать различные свойства рендеринга, например, использование нормалей, наложение текстур и стиль отрисовки.

Использование gluNewQuadric() и работа с квадриками имеют следующие преимущества:

1. Уменьшение сложности кода: Квадрики значительно упрощают создание сложных объектов, уменьшая количество необходимого кода для их визуализации.
2. Гибкие возможности отображения: Этот подход предоставляет возможность легко изменять параметры рендеринга, такие как нормали и текстуры, адаптировать примитивы к конкретным задачам.
3. Кросс-платформенная поддержка: Работа с квадриками гарантирует совместимость с разными версиями OpenGL и платформами, делая их универсальными инструментами для различных приложений.

gluCylinder(quadric, baseRadius, topRadius, height, slices, stacks)

Описание:Функция gluCylinder() предназначена для рисования цилиндра на основе объекта типа GLUquadric, который создается с помощью функции gluNewQuadric(). Эта функция предоставляет возможность отобразить цилиндр с заданными параметрами, что включает в себя как обычные цилиндры, так и конусы.

Назначение: Основная цель использования функции gluCylinder() заключается в создании трёхмерных объектов для визуализации разнообразных сцен, например, архитектурных элементов, механических деталей или декоративных объектов.

Параметры:

- quadric: Указатель на объект типа GLUquadric.

Этот параметр представляет собой указатель на структуру данных, которая описывает характеристики поверхности, используемой для построения цилиндра. Объект типа GLUquadric создаётся функцией gluNewQuadric() и содержит информацию о нормалях, текстурах и других свойствах, которые будут применены к создаваемому цилиндру.

- baseRadius: Радиус основания цилиндра.

Это радиус нижней (базовой) части цилиндра. Он определяет размер основания цилиндра, которое будет расположено параллельно плоскости XY.

- topRadius: Радиус верхней части цилиндра.

Данный параметр задаёт радиус верхней части цилиндра. Если этот радиус равен радиусу основания (baseRadius), то будет создан обычный цилиндр. В случае, когда радиус вершины отличается от радиуса основания, результатом станет конусообразная фигура.

- height: Высота цилиндра.

Параметр указывает на расстояние между основанием и вершиной цилиндра вдоль оси Z. Высота определяет длину цилиндра или конуса.

- slices: Количество секторов по окружности цилиндра.

Этот параметр определяет количество вертикальных сегментов, которые используются для построения боковой поверхности цилиндра. Чем больше значение этого параметра, тем более гладким будет отображаться цилиндр. Например, при значении 20 боковые грани цилиндра будут практически незаметны, создавая иллюзию круглой формы.

- stacks: Количество горизонтальных сегментов.

Определяет количество горизонтальных сегментов, разбивающих цилиндр на несколько колец вдоль его высоты. Чем больше это значение, тем точнее будет воспроизведена высота цилиндра.

gluDisk(quadric, innerRadius, outerRadius, slices, loops)

Описание: Функция gluDisk() предназначена для рисования диска с определенными параметрами, основанная на объекте типа GLUquadric, который создается с использованием функции gluNewQuadric(). В зависимости от значений параметров можно создать либо полный круг (сплошной), либо кольцо (с отверстием внутри).

Цель: Основная цель применения функции gluDisk() состоит в создании плоских, круглых объектов, таких как основания колонн, диски, кнопки и прочие подобные элементы.

Параметры:

- quadric: Указатель на объект типа GLUquadric, который был создан ранее с помощью функции gluNewQuadric().
- innerRadius: Внутренний радиус диска. Если этот радиус равен нулю, то диск будет полностью заполненным (т.е., без отверстия посередине). В противном случае он представляет собой внутренний радиус кольца.
- outerRadius: Внешний радиус диска. Он определяет максимальное расстояние от центра диска до его внешнего края.
- slices: Число секторов (или "ломтей"), которые будут использованы при построении окружности диска. Чем выше это число, тем более гладкой будет форма круга. То есть, чем больше ломтей, тем точнее аппроксимируется круговая форма.
- loops: Количество радиальных уровней (слоев) диска. Этот параметр позволяет создавать многослойные объекты, повышая их детализацию.

gluPartialDisk(quadric, innerRadius, outerRadius, slices, loops, startAngle, endAngle)

Описание: Отрисовывает часть диска (дугу), используя заданные радиусы.

Использование: Используется для создания арок или дуг, таких как ручка лейки.

gluSphere(quadric, radius, slices, stacks)

Описание: Отрисовывает сферу с заданным радиусом, количеством сегментов по окружности и вдоль высоты.

Использование: Используется в коде для создания капель воды и других сферических объектов.

glEnable(cap)

Описание: Включает указанную возможность (например, освещение, тест глубины и т.д.).

Использование: Используется для активирования различных аспектов рендеринга в OpenGL.

glDisable(cap)

Описание: Отключает указанную возможность.

Использование: Используется для деактивации определенных функций рендеринга.

glLight(light, pname, param)

Описание: Устанавливает позицию или свойство источника света.

Использование: Позволяет настроить освещение на сцене, задавая свойства таких элементов, как интенсивность и положение.

glLightfv(light, pname, params)

Описание: Устанавливает параметры света, которые имеют массив значений (например, цвет и интенсивность освещения).

Использование: Используется для определения характеристик источников света в 3D-сцене.

glClear(mask)

Описание: Очищает указанные буферы (например, буфер цвета или глубины).

Использование: Используется в начале каждого кадра для удаления предыдущих изображений.

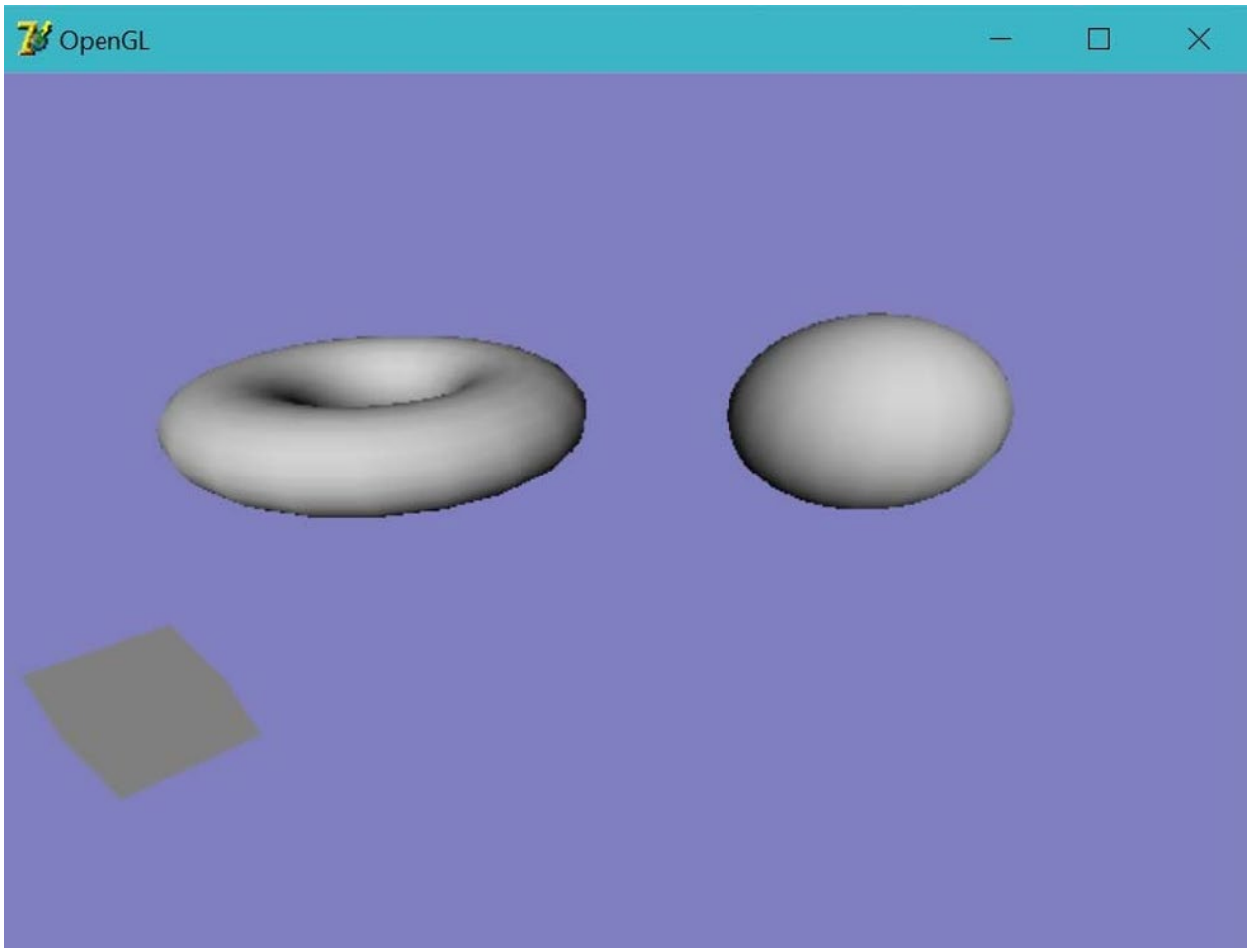

Рисунок 2 – Пример отрисовки объектов (куб, тор и сфера) с использованием функций OpenGL

3.3 Математические формулы расчёта новых координат, позволяющие рассчитать траекторию движения

Параметрическое уравнение окружности: чтобы рассчитать движение объектов по дуге, для заданного радиуса (или расстояния от точки вращения) и угла можно использовать следующие уравнения:

- $y(t)=R*\sin(\theta(t))$;
- $z(t)=R*(1-\cos(\theta(t)))$

Здесь:

- $y(t)$ — вертикальная координата капли в момент времени t;
- $z(t)$ — глубинная координата капли;
- R — радиус дуги;
- $\theta(t)$ — угол, зависящий от времени (или другого параметра), который увеличивается с каждым кадром.

Расчет координат:

y1,y2,y3 (вертикальная координата) для каждой капли изменяется в зависимости от угла через синус:

1) y1 = 0.5 + R * sin(θ(t));
2) y2 = 0.3 + R* sin(θ(t));
3) y3 = 0.37 + R*sin(θ(t));

z1,z2,z3 (глубинная координата) рассчитывается через косинус:

1) z1 = 1.6 + R* (1 -cos(θ(t)));
2) z2 = 1.7 + R* (1 -cos(θ(t)));
3) z3 = 1.75 + radius * (1 -cos(θ(t)));

4. Листинг с кодом программы:

```
import pygame
from pygame.locals import *
from OpenGL.GL import *
from OpenGL.GLU import *
import math

# Функция для отрисовки лейки
def draw_watering_can(height=1, radius=1, handle_radius=0.5,
handle_angle=180, spout_length=1):

    # Отрисовка основной части лейки (цилиндр)
    glPushMatrix()
    glTranslatef(0, height / 2, 0)
    glRotatef(90, 1, 0, 0)
    quadric = gluNewQuadric()
    gluCylinder(quadric, radius, radius, height, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    # Отрисовка дна лейки
    glPushMatrix()
    glTranslatef(0, -height / 2, 0)
    glRotatef(handle_angle / 2, 16, 0, 0)
    quadric = gluNewQuadric()
    gluDisk(quadric, 0, radius, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()
    # второе дно, чтобы закрыть остаток носика лейки
    glPushMatrix()
    glTranslatef(0, 0, 0)
    glRotatef(handle_angle / 2, 16, 0, 0)
    quadric = gluNewQuadric()
    gluDisk(quadric, 0, radius, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    # Отрисовка верха лейки
    glPushMatrix()
    glTranslatef(0, 0.5, -0)
    glRotatef(handle_angle / 2, 10, 0, 0)
    quadric = gluNewQuadric()
    gluPartialDisk(quadric, 0.25, 0.5, 30, 1, 0, 360)
    gluDeleteQuadric(quadric)
    glPopMatrix()
    #Нижний верх лейки
    glPushMatrix()
    glTranslatef(0, 0.4, -0)
    glRotatef(handle_angle / 2, 10, 0, 0)
    quadric = gluNewQuadric()
    gluPartialDisk(quadric, 0.25, 0.5, 30, 1, 0, 360)
    gluDeleteQuadric(quadric)
    glPopMatrix()
    #стенки верха лейки
    glPushMatrix()
    glTranslatef(0, 0.5, 0)
    glRotatef(90, 10, 0, 0)
    quadric = gluNewQuadric()
    gluCylinder(quadric, 0.25, 0.25, 0.1, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()
```

```
    # Отрисовка ручки лейки в виде дуги
    glPushMatrix()
    glTranslatef(0, 0, -0.5)
    glRotatef(handle_angle / 2, 0, 1, 0)
    quadric = gluNewQuadric()
    gluPartialDisk(quadric, 0.3, handle_radius, 30, 1, 0, handle_angle)
    gluDeleteQuadric(quadric)
    glPopMatrix()
# ручка - цилиндром (нет)
#     glPushMatrix()
#     glTranslatef(0, 0, -0.5)
#     glRotatef(handle_angle / 2, 0, 1, 0)
#     quadric = gluNewQuadric()
#     gluCylinder(quadric, 0.25, 0.25, 0.1, 32, 16)
#     gluDeleteQuadric(quadric)
#     glPopMatrix()

    # Отрисовка носика лейки
    glPushMatrix()
    glTranslatef(0, 0, 0.4)
    glRotatef(-30, 1, 0, 0)
    quadric = gluNewQuadric()
    gluCylinder(quadric, 0.07, 0.03, spout_length, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    # Отрисовка душика лейки
    glPushMatrix()
    glTranslatef(0, 0.5, 1.25)
    glScalef(1.0, 1, 0.3)
    quadric = gluNewQuadric()
    gluSphere(quadric, 0.15, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

def draw_water(angle):
    # Считываем радиус дуги
    arc_radius = 0.5  # Радиус дуги

    # Вычисляем новые координаты капель по формуле
    y1 = 0.5 + arc_radius * math.sin(math.radians(angle))
    z1 = 1.6 + arc_radius * (1 - math.cos(math.radians(angle)))

    y2 = 0.3 + arc_radius * math.sin(math.radians(angle))
    z2 = 1.7 + arc_radius * (1 - math.cos(math.radians(angle)))

    y3 = 0.37 + arc_radius * math.sin(math.radians(angle))
    z3 = 1.75 + arc_radius * (1 - math.cos(math.radians(angle)))

    # Отрисовка капель
    glPushMatrix()
    glTranslatef(0, y1, z1)  # Применяем новую координату
    glScalef(1.0, 1, 1)
    quadric = gluNewQuadric()
    gluSphere(quadric, 0.05, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    glPushMatrix()
    glTranslatef(0, y1, z1 - 0.11)  # Применяем новую координату для цилиндра
    quadric = gluNewQuadric()
    gluCylinder(quadric, 0, 0.05, 0.1, 32, 16)
    gluDeleteQuadric(quadric)
```

```
    glPopMatrix()

    # Для капли 2
    glPushMatrix()
    glTranslatef(0.2, y2, z2)
    glScalef(1.0, 1, 1)
    quadric = gluNewQuadric()
    gluSphere(quadric, 0.05, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    glPushMatrix()
    glTranslatef(0.2, y2 + 0.1, z2 - 0.05)
    glRotatef(63, 0.5, 0, 0)
    quadric = gluNewQuadric()
    gluCylinder(quadric, 0, 0.05, 0.1, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    # Для капли 3
    glPushMatrix()
    glTranslatef(-0.13, y3, z3)
    quadric = gluNewQuadric()
    gluSphere(quadric, 0.05, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

    glPushMatrix()
    glTranslatef(-0.15, y3 + 0.1, z3 - 0.08)
    glRotatef(45, 0.5, 0.1, 0)
    quadric = gluNewQuadric()
    gluCylinder(quadric, 0, 0.05, 0.1, 32, 16)
    gluDeleteQuadric(quadric)
    glPopMatrix()

def main():
    pygame.init()
    display = (1000, 800)
    pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
    pygame.display.set_caption('Watering Can')
    gluPerspective(45, (display[0] / display[1]), 0.1, 50.0)
    glTranslatef(0.0, 0.0, -5)

    # Настройка освещения
    glLight(GL_LIGHT0, GL_POSITION, (5, 5, 5, 1))  # Свет исходит слева
сверху спереди
    glLightfv(GL_LIGHT0, GL_AMBIENT, (2, 1, 2, 4))
    glLightfv(GL_LIGHT0, GL_DIFFUSE, (2, 10, 1, 1))

    glEnable(GL_DEPTH_TEST)
    glEnable(GL_LIGHTING)
    glEnable(GL_LIGHT0)
    glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE)

    rotate_vector = [0, 0, 0]
    angle = 0  # Начальный угол

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()
```

```
        # Обработка нажатий клавиш для вращения
        pressed = pygame.key.get_pressed()
        if pressed[pygame.K_UP]:
            rotate_vector = [-1, 0, 0]
        if pressed[pygame.K_DOWN]:
            rotate_vector = [1, 0, 0]
        if pressed[pygame.K_RIGHT]:
            rotate_vector = [0, 1, 0]
        if pressed[pygame.K_LEFT]:
            rotate_vector = [0, -1, 0]
        if pressed[pygame.K_SPACE]:
            rotate_vector = [0, 0, 0]
        # if pressed[pygame.K_LEFT]:
        #     glTranslatef(0.1,0.1,0)
        # if pressed[pygame.K_RIGHT]:
        #     glTranslatef(-0.1,-0.1,0)

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

        # Обновляем угол для движения капель

        angle += 2  # Увеличиваем угол на 2 градуса в каждом кадре
        if angle >= 100:
            angle = 0  # Обнуляем угол для непрерывного движения

        if rotate_vector != [0, 0, 0]:
            glRotatef(1, rotate_vector[0], rotate_vector[1],
rotate_vector[2])

        # Рисуем лейку
        draw_watering_can(height=1, radius=0.5, handle_radius=0.5,
handle_angle=180, spout_length=1)
        draw_water(angle)

        pygame.display.flip()
        pygame.time.wait(10)


if __name__ == '__main__':
    main()
```

5. Экранные формы с результатами работы программы:

Демонстрация движения капель по траектории «дуги»:

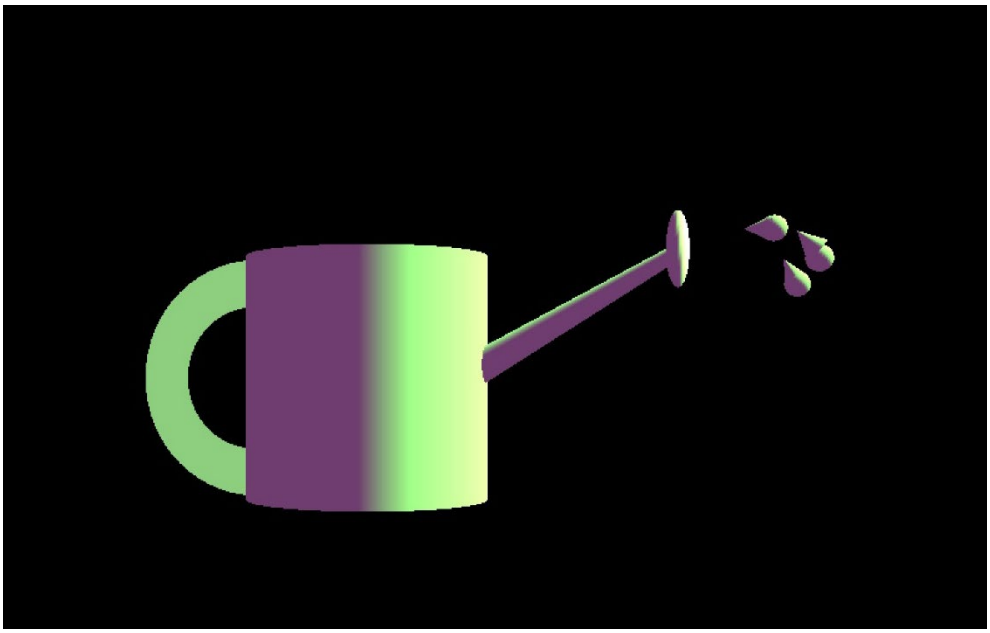

Рисунок 3.1 – Кадр 1, капли в исходном положении

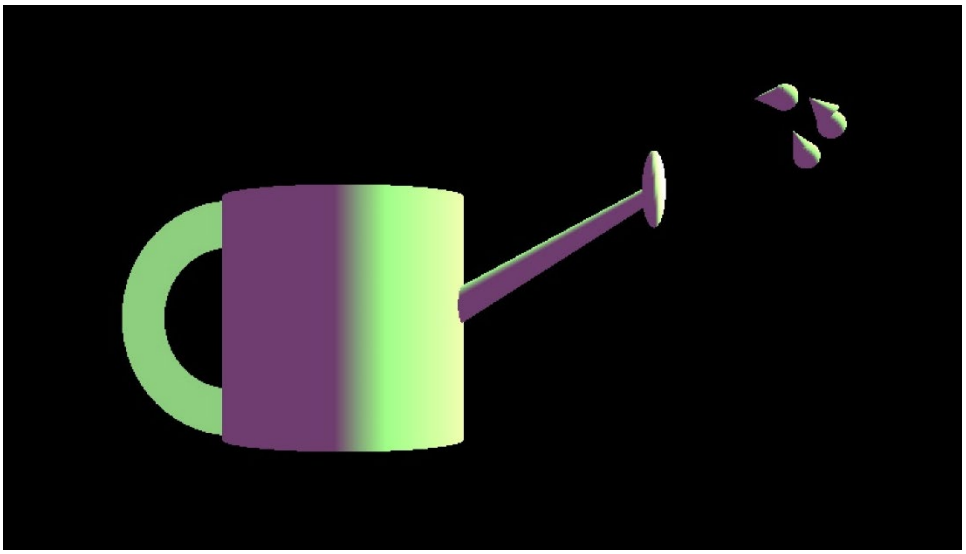

Рисунок 3.2 – Кадр 2, капли сдвинулись дальше от лейки

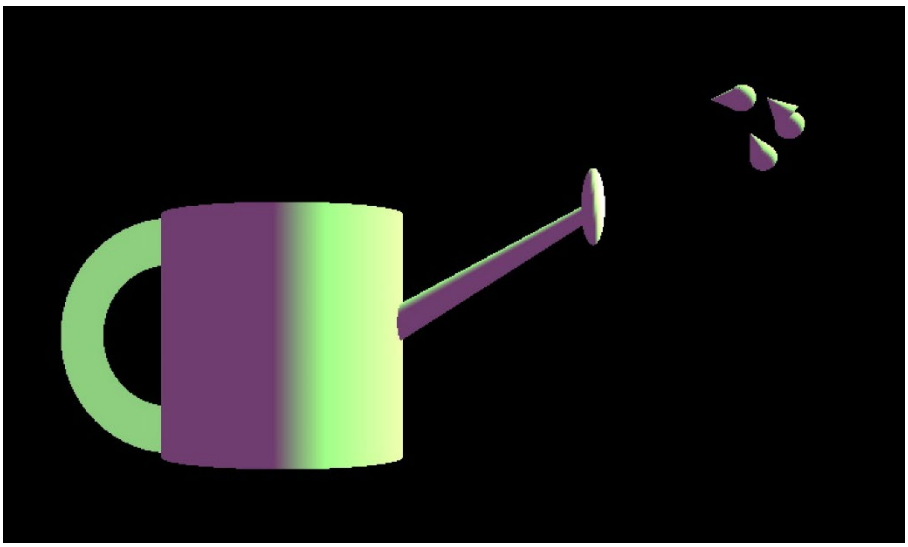

Рисунок 3.3. – Кадр 3, капли сдвинулись ещё дальше

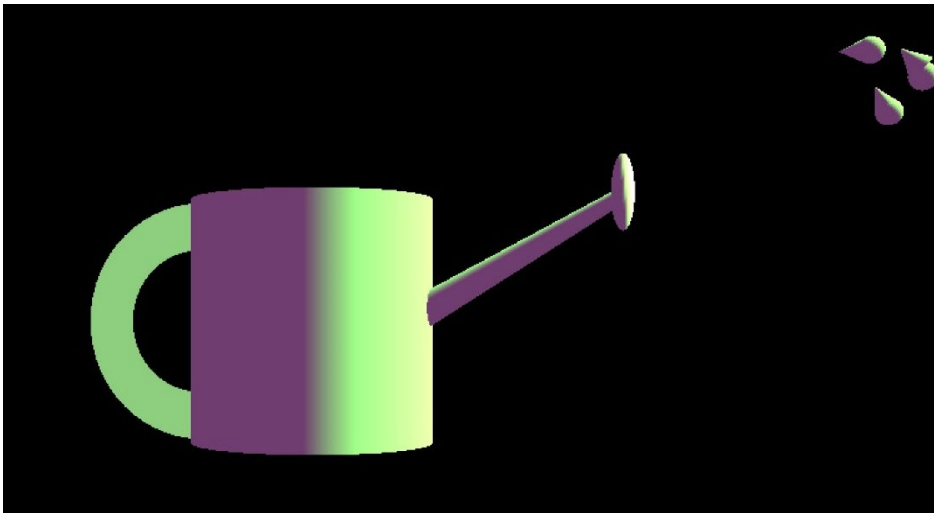

Рисунок 3.4 – Кадр 4, капли достигли края экрана

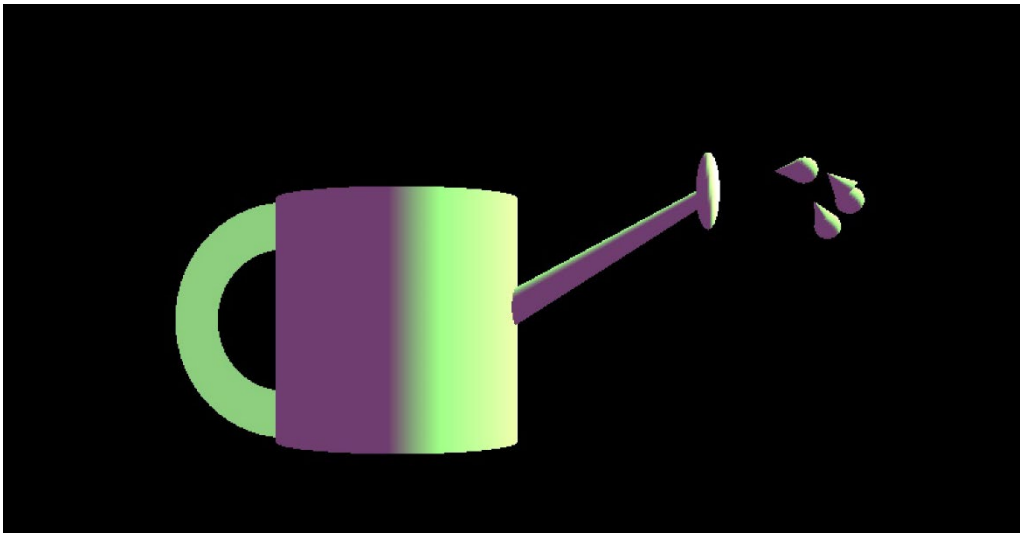

Рисунок 3.5 – Кадр 5, капли вернулись в исходное положение

Демонстрация динамики объекта – лейки. Динамика запускается и управляется клавишами клавиатуры «стрелки», выбирается направления вращения, на «пробел»

движения останавливается. Капли двигаются, по своей траектории, продемонстрировано выше.

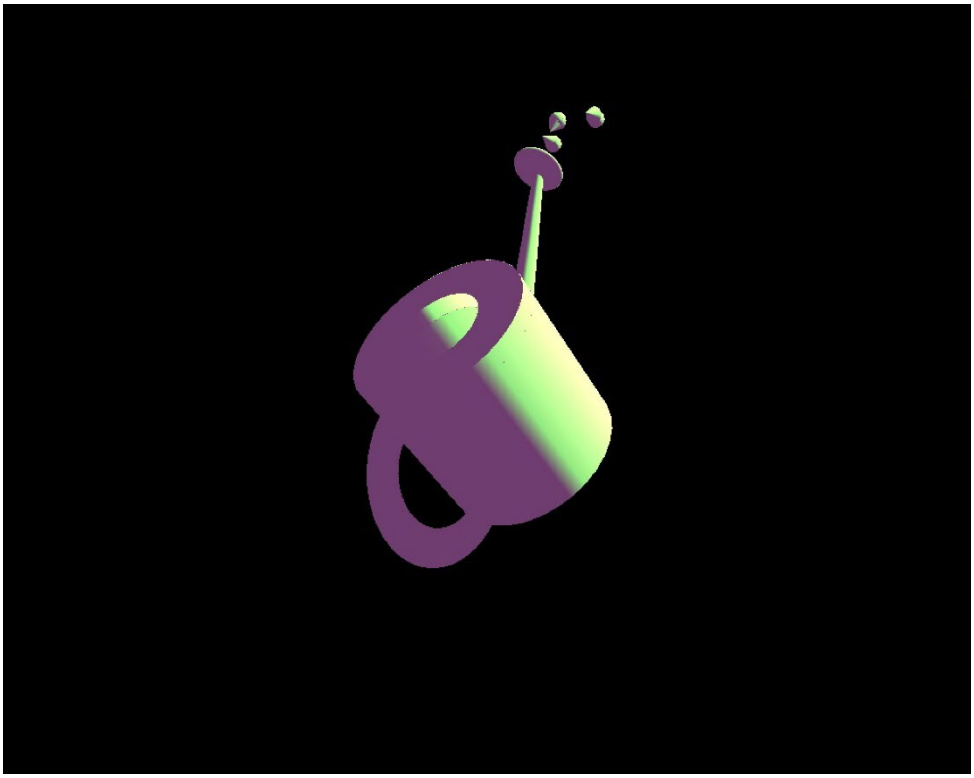

Рисунок 4.1 – Вращение лейки, кадр 1

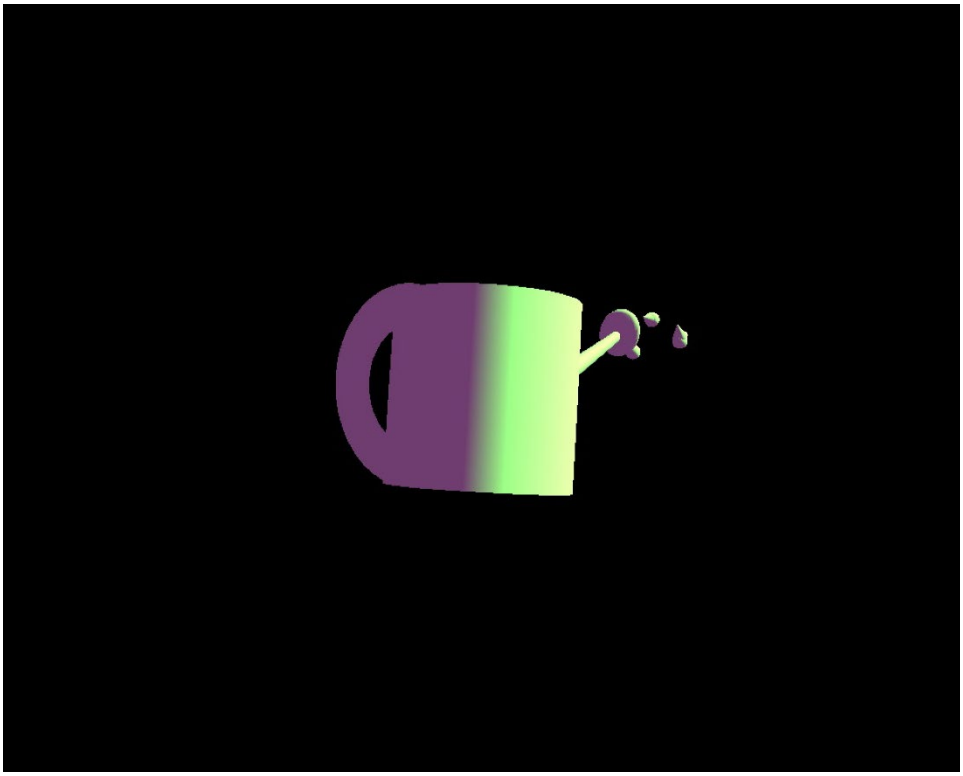

Рисунок 4.2 – Вращение лейки, кадр 2

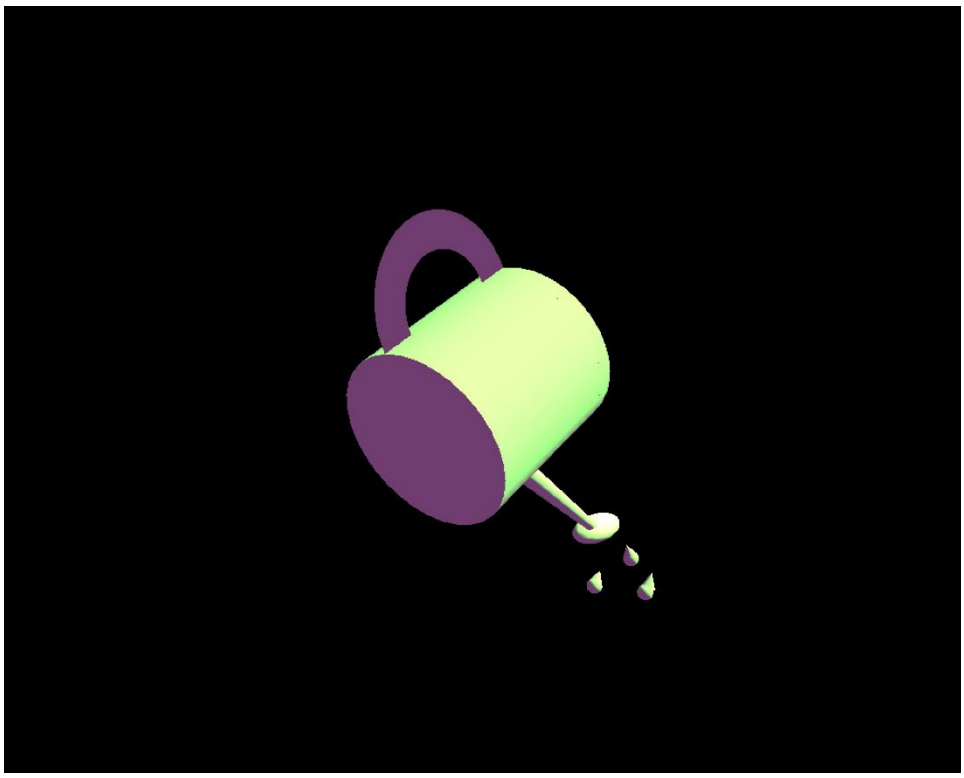

Рисунок 4.3 – Вращение лейки, кадр 3

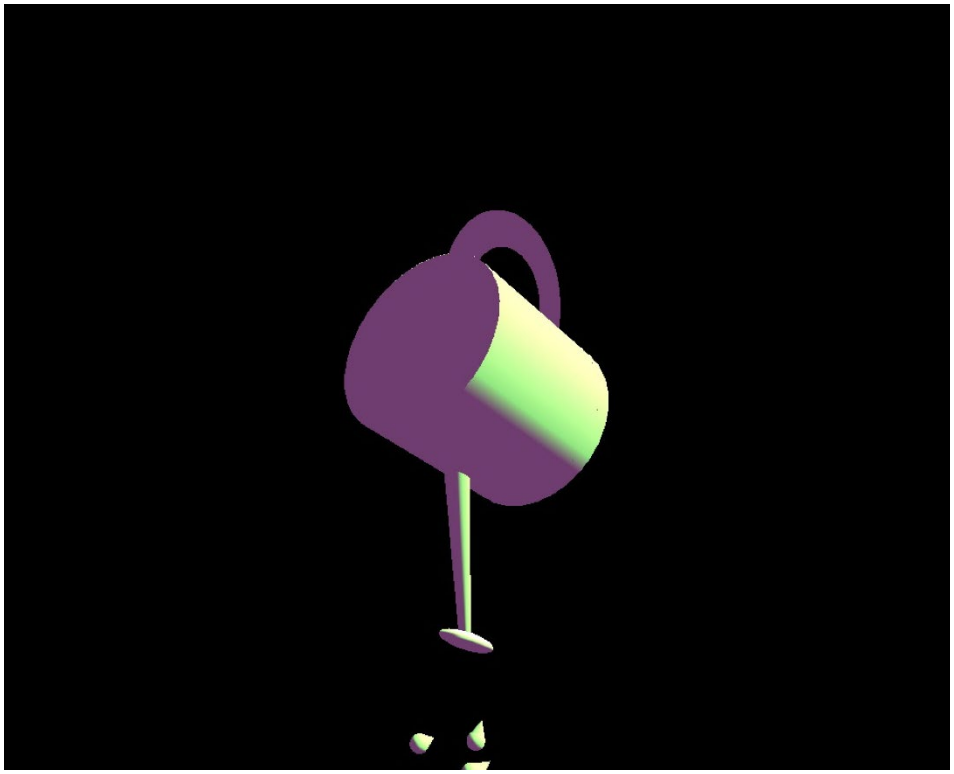

Рисунок 4.4 – Вращение лейки, кадр 4

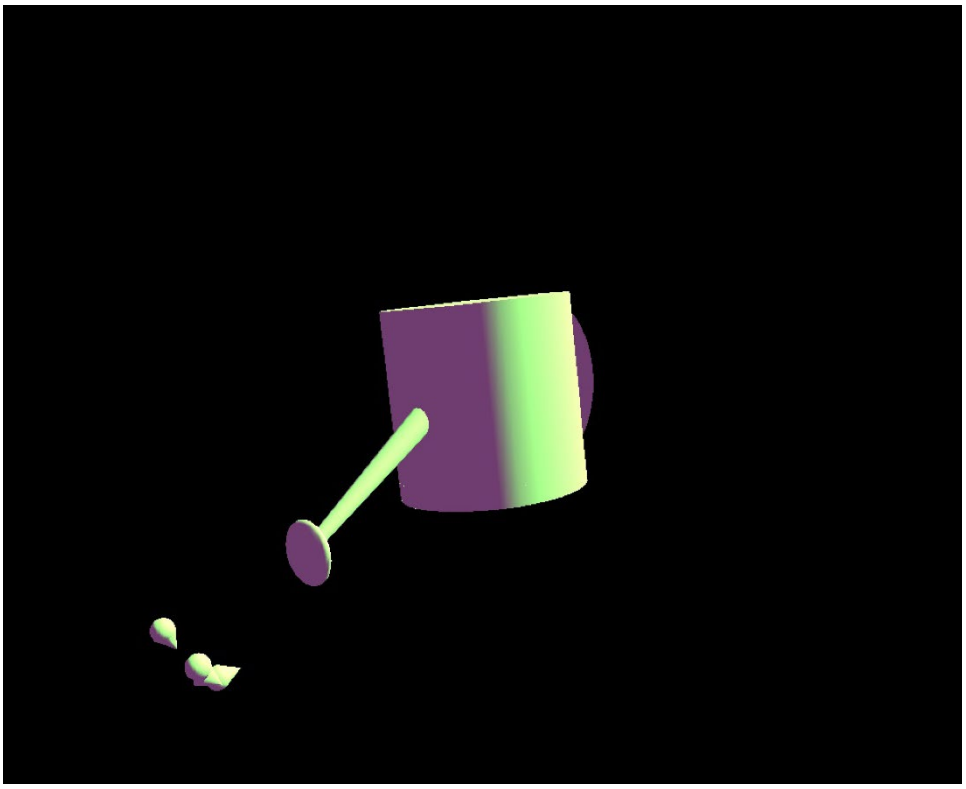

Рисунок 4.5 – Вращение лейки, кадр 5

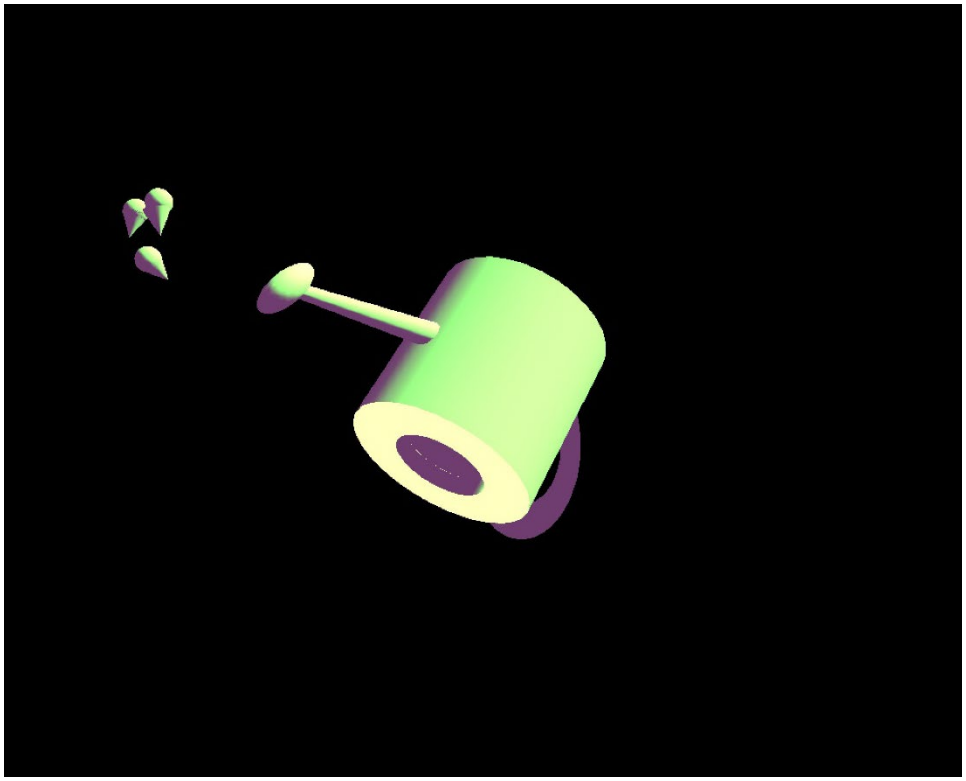

Рисунок 4.5 – Вращение лейки, кадр 6

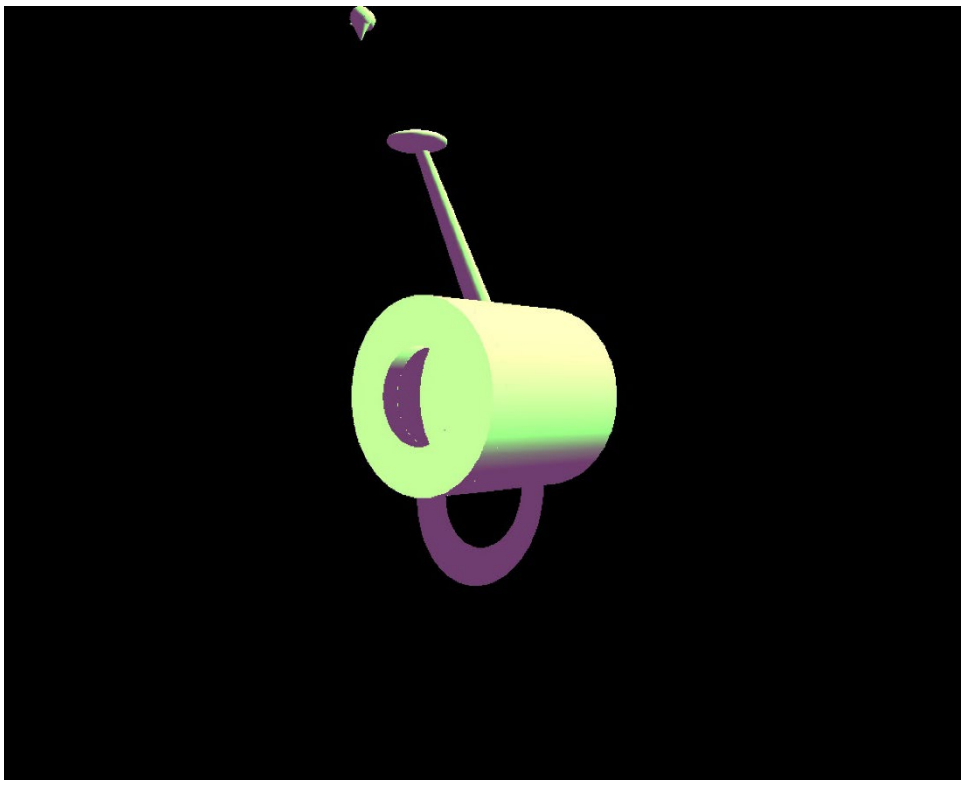

Рисунок 4.7 – Вращение лейки, кадр 7

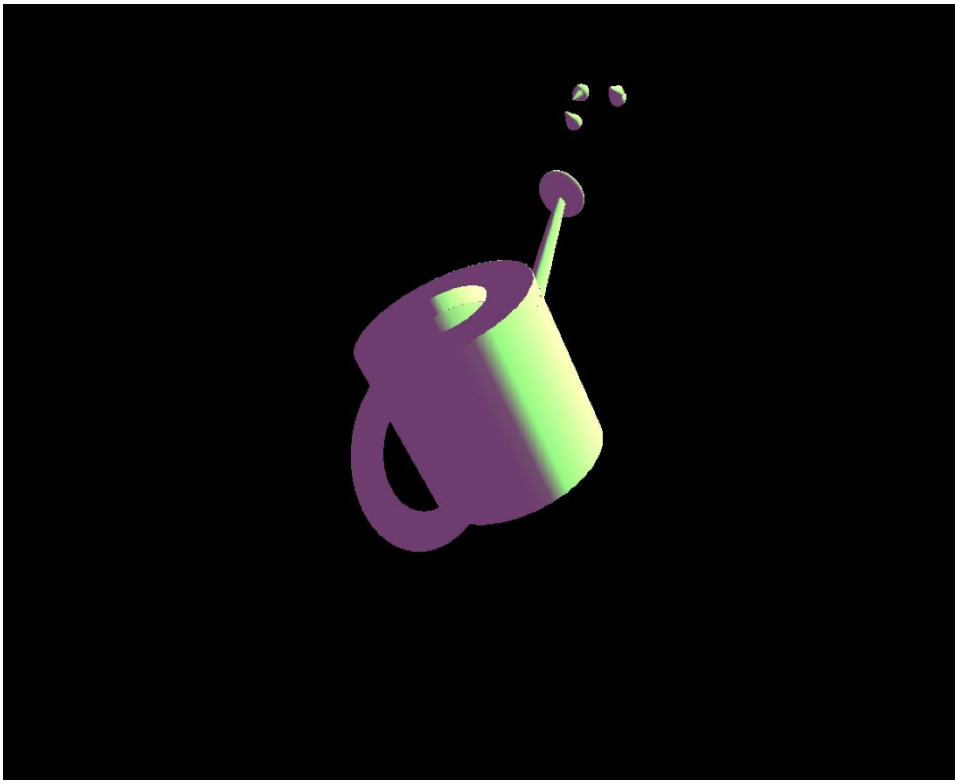

Рисунок 4.8 – Вращение лейки, кадр 8

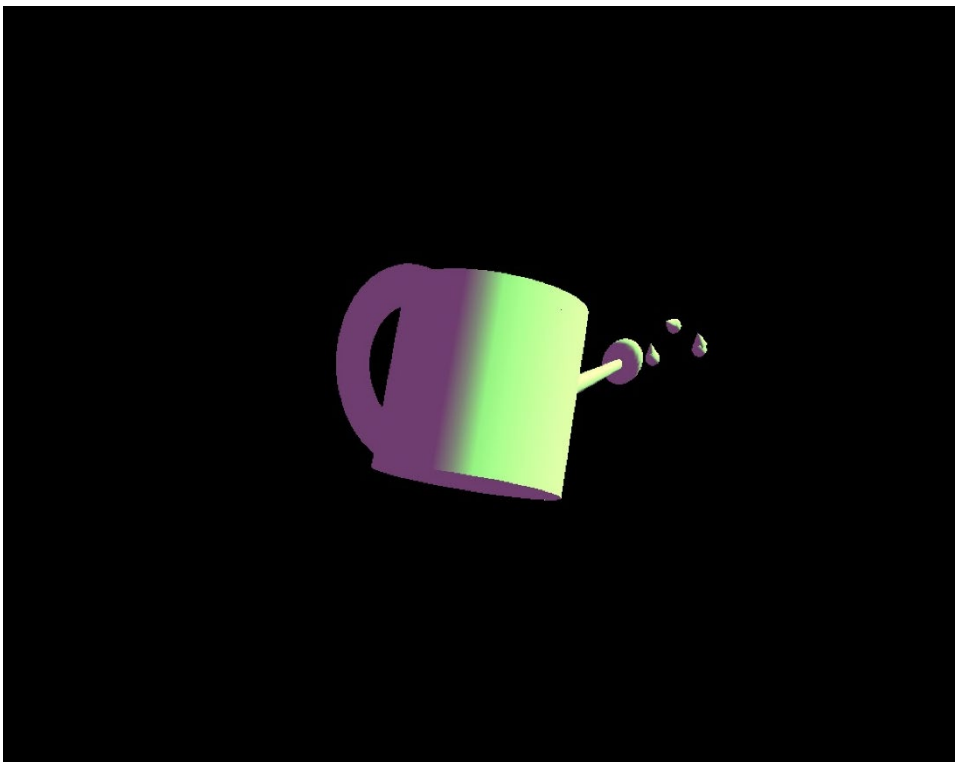

Рисунок 4.9 – Вращение лейки, кадр 9

6. Вывод:

В данной работе я подробно рассмотрела возможности библиотеки OpenGL.

OpenGL является мощным программным обеспечением, используемым для создания разнообразных графических приложений, таких как простые геометрические объекты (примитивы), сложные трёхмерные модели и настройка освещения. Библиотека предоставляет широкий спектр функций, которые помогают разработчикам быстро создавать и отображать различные примитивы и сложные 3D-объекты в виртуальной среде. Она также содержит инструменты для управления освещением сцены, что помогает достичь более реалистичного изображения. Эти возможности делают OpenGL важным инструментом для тех, кто занимается разработкой высококачественных 3D-приложений.

Я разработала динамическую 3D-сцену, используя высокоуровневый язык программирования, совместимый с библиотекой OpenGL. В процессе написания кода для этой сцены были использованы следующие функции библиотеки:

1. **Матричные операции**:

   glTranslatef – функция для перемещения объектов в пространстве;

   gluPerspective – установка перспективного вида камеры;

   glRotatef – вращение объектов относительно выбранной оси;

   glPushMatrix и glPopMatrix – сохранение и восстановление текущих состояний матрицы (что позволяет временно менять матричное состояние без утраты предыдущих настроек).

2. **Освещение**:

   glLight – настройка параметров светового источника;

   glLightfv – определение характеристик света (таких как его цветовые составляющие);

   glColorMaterial – управление цветом материалов объектов под воздействием освещения.

3. **Отображение примитивов**:

   glBegin – начало определения примитивов (точечных, линейных, треугольных элементов и др.);

   glVertex3fv – указание координат вершин примитивов;

   glEnd – окончание описания примитивов.

   glPushMatrix – сохраняет текущую матрицу модели в стек. Позволяет изолировать трансформации в рамках текущей матрицы.

   glPopMatrix – восстанавливает матрицу модели из стека. Возвращает к состоянию сохраненной матрицы.

gluNewQuadric – создает новый объект типа Quadric, который может быть использован для рисования различных гладких форм.

4. **Конфигурация OpenGL**:

glDisable – деактивация определённых функций OpenGL;

glEnable – активация выбранных возможностей OpenGL.

Эти функции помогли мне эффективно контролировать различные аспекты отображения трёхмерных сцен, такие как манипуляции объектами, освещение и конфигурацию графической системы.

Дополнительно была затронута библиотека PyGame, позволяющая работать с графическим интерфейсом, выводить окно с динамической 3D сценой – объектами, отрисованными OpenGL.

Таким образом, в процессе лабораторной работы я ознакомилась с открытой графической библиотекой OpenGL, написав программу, демонстрирующую динамическую сцену, на языке высокого уровня Python. Библиотека может пригодится в дальнейшем для решения других задач в сфере информационных технологий, науке о данных, виртуальной реальности и других областях, где используется моделирование объектов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Краснов М. OpenGL. Графика в проектах Delphi. – СПб.: БХВ Петербург, 2002. - 352 с.
2. Райт, Р.С.-мл., Липчак Б. OpenGL. Суперкнига, 3-е издание. — М.: Издательский дом «Вильямс», 2006. - 1040 с.
3. PyOpenGL для начинающих и немного новогоднего настроения / Хабр – URL: https://habr.com/ru/articles/246625/ (дата обращения 15.12.2024)
4. learnopengl. Урок 1.1 — OpenGL / Хабр – URL: https://habr.com/ru/articles/310790/ (дата обращения 15.12.2024)
5. Компьютерная графика :: OpenGL :: Примитивы и преобразования пространства в OpenGL на примере рисования куба – URL: https://compgraphics.info/OpenGL/draw_a_cube.php (дата обращения 15.12.2024)
6. learnopengl. Урок 1.7 — Трансформации / Хабр – URL: https://habr.com/ru/articles/319144/ (дата обращения 15.12.2024)