Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Компьютерная графика" №1

Студент: Андрюшин Л. Д.

Группа: М8О-312Б-22

Москва, 2024

Основы 2D-графики и трансформаций

Цель лабораторной работы:
В данной лабораторной работе вам предстоит научиться работать с графическим API для отрисовки 2D-примитивов, освоить основные 2D-трансформации (перемещение, масштабирование, поворот) и изучить алгоритмы построения 2D-кривых.

Вариант 7. Отрисовка многоугольника с интерполяцией
Постройте многоугольник с 6-ю вершинами.
Реализуйте перемещение всех вершин одновременно с помощью матричных операций.
Добавьте интерполяцию между начальной и конечной позицией многоугольника для плавного движения.
Дополнительно: Реализуйте циклическую анимацию, при которой многоугольник меняет свою форму и возвращается в исходное состояние.

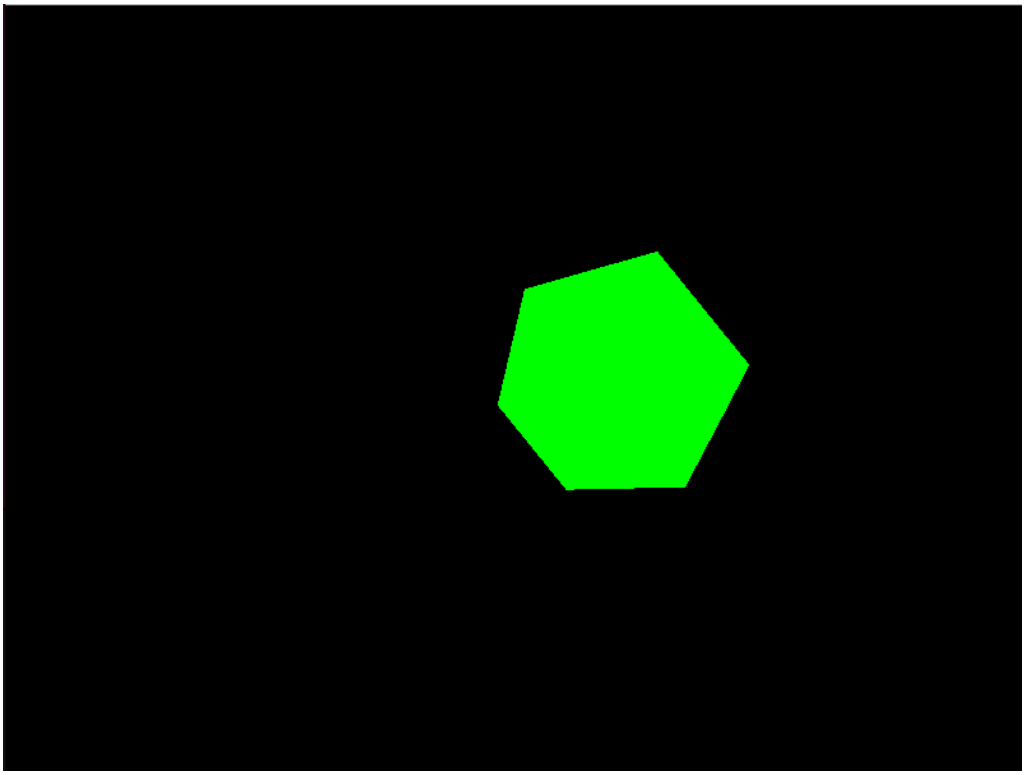
Решение

Генерация шестиугольника: Для генерации шестиугольника используется функция generatePolygon, которая вычисляет координаты его вершин, исходя из радиуса и центра. Вершины расположены на равном расстоянии друг от друга по окружности.

- **Интерполяция**: Для плавного перемещения шестиугольника используется функция interpolate, которая вычисляет промежуточную точку между начальной и конечной точками, с параметром интерполяции t, который изменяется от 0 до 1. Этот процесс плавно перемещает шестиугольник по экрану.
- **Вращение**: Функция rotatePolygon выполняет вращение многоугольника вокруг центра. Для этого каждый пиксель многоугольника перемещается относительно центра и затем применяется формула вращения на основе угла.
- **Деформация**: Для деформации используется изменение координат вершин в зависимости от текущей степени деформации. Параметр деформации изменяется в диапазоне от 0 до 0.2 и регулирует искажение формы шестиугольника.
- **Анимация**: В процессе анимации используется цикл, который изменяет параметры движения, деформации и вращения. Направление движения и деформации меняется, когда они достигают максимального или минимального значения.
- **Управление через клавиши**: Для управления параметрами анимации в программе предусмотрены горячие клавиши:

- Пробел включает или выключает деформацию.
- Клавиша "R" включает или выключает вращение.
- Клавиши влево и вправо изменяют скорость вращения.
- Клавиши вверх и вниз регулируют скорость интерполяции.

- **Отображение**: После вычисления новых координат и применении всех трансформаций, шестиугольник отрисовывается на экране с использованием SFML. Вершины шестиугольника передаются в объект sf::ConvexShape, который отображает фигуру с заданными точками и цветом.



Вывод

Лабораторная работа демонстрирует навыки работы с OpenGL и базовыми графическими трансформациями. Все вычисления для преобразований выполняются вручную, что соответствует требованиям задания. Программа подтверждает понимание принципов линейной алгебры и их применения в графике, а также основ визуализации. Возможность интерактивного управления через клавиатуру улучшает пользовательский опыт, делая приложение более динамичным.

Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Компьютерная графика" №2

Студент: Андрюшин Л. Д.

Группа: М8О-312Б-22

Москва, 2024

Основы 3D-графики и проекция

Цель лабораторной работы:
В этой лабораторной работе вы познакомитесь с основами 3D-графики: построением простых 3D-объектов, проекцией на 2D-плоскость, а также научитесь работать с матрицами перспективы и ортографической проекции.

Вариант 7. Построение 3D-сцены с несколькими объектами
Постройте сцену, содержащую куб, пирамиду и сферу.
Используйте перспективную проекцию для отображения сцены.
Реализуйте возможность перемещения камеры по сцене с помощью клавиатуры.
Дополнительно: Реализуйте возможность приближения и удаления камеры от объектов с плавным изменением угла обзора.

Решение

Инициализация окна и настройки OpenGL:
В первой части программы создается окно с использованием SFML. Для обеспечения работы с 3D-графикой и глубиной сцены включается тест на глубину (GL_DEPTH_TEST). Это позволяет корректно отрисовывать объекты, избегая их наложения друг на друга.

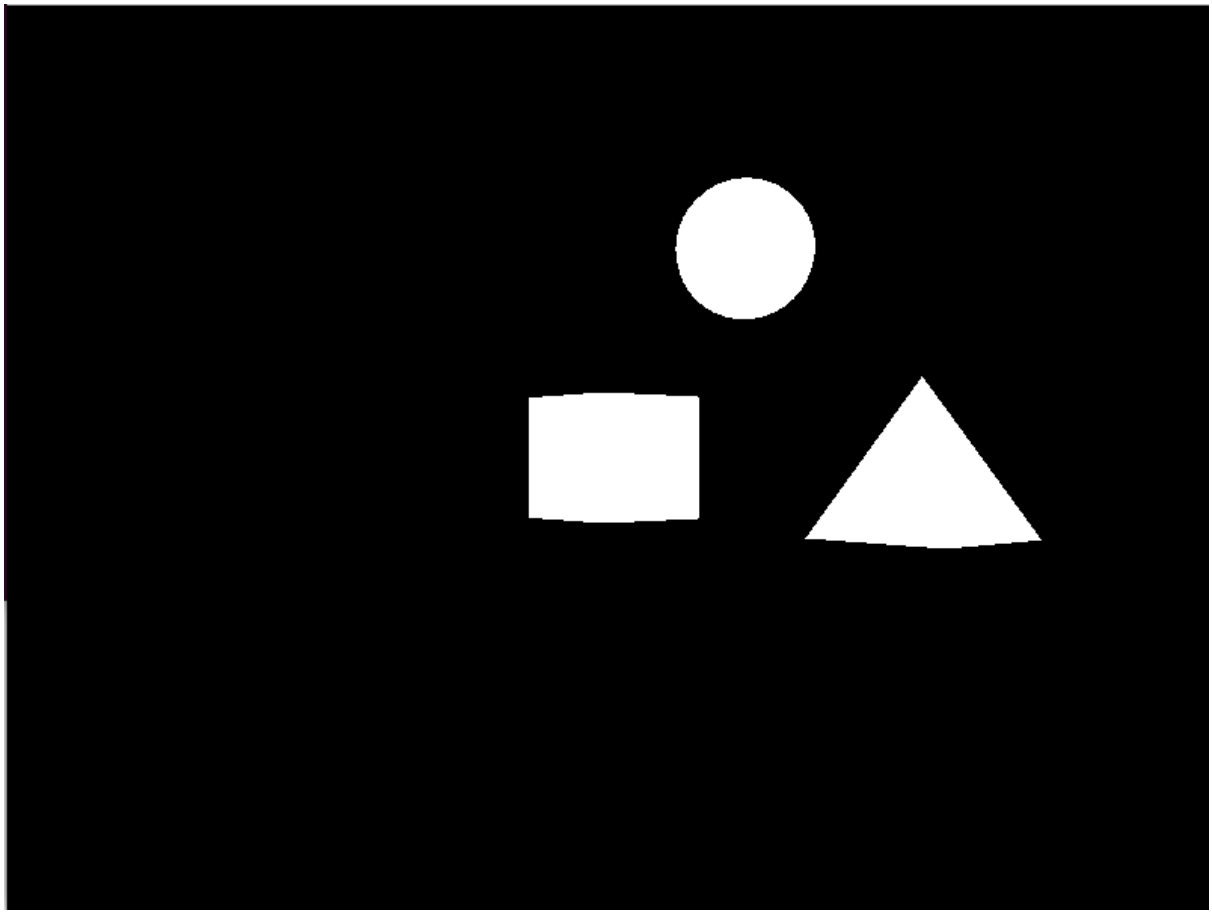
- **Настройка перспективной проекции**:
  В программе используется функция setPerspectiveProjection(), которая настраивает перспективную проекцию с помощью OpenGL. Это позволяет правильно отображать объекты на 3D-сцене с учетом перспективы, а также управлять полем зрения (FOV), что помогает изменять восприятие сцены в зависимости от ввода пользователя.

- **Реализация 3D-объектов**:
  Для создания графических объектов (куб, пирамида, сфера) используются соответствующие функции:

- drawCube() рисует куб с использованием метода GL_QUADS, определяя его вершины для каждой из шести граней.
- drawPyramid() рисует пирамиду с использованием GL_TRIANGLES, определяя треугольники для каждой из четырех граней.
- drawSphere() рисует сферу с помощью GL_QUAD_STRIP, создавая полигоны для разбиения сферы на сечения по вертикали и горизонтали.

- **Управление камерой**:
  Камера может перемещаться по сцене с помощью клавиш W, A, S, D, изменяя положение камеры по осям X, Y и Z. Также предусмотрено управление углами наклона (pitch) и вращения (yaw) камеры с помощью стрелок на клавиатуре. Это позволяет пользователю свободно исследовать сцену.

- **Интерактивность**:
  Программа поддерживает управление с клавиатуры для изменения позиции камеры и поля зрения (FOV), что позволяет пользователю динамично взаимодействовать с 3D-сценой. Изменение углов наклона и вращения камеры позволяет пользователю "осматривать" объекты со всех сторон.
- **Отображение объектов на экране**:
  В основном цикле программы объекты (куб, пирамида, сфера) отрисовываются с использованием функции glPushMatrix() и glPopMatrix(), чтобы применить трансформации (перемещение, вращение) отдельно к каждому объекту. Эти объекты отрисовываются в разных местах на сцене, что позволяет создать динамичную и разнообразную картину.



Вывод

Лабораторная работа позволила освоить основы работы с 3D-графикой в OpenGL, включая создание и отрисовку 3D объектов, а также управление

камерой. Программа продемонстрировала использование математических методов для управления углами наклона и вращения камеры, а также для создания перспективной проекции. Взаимодействие с пользователем через клавиши позволяет динамически изменять параметры камеры и сцену, создавая интерактивный опыт. Работа подтверждает понимание принципов 3D-графики, включая трансформации, перспективу и обработку пользовательского ввода.

Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Компьютерная графика" №3

Студент: Андрюшин Л. Д.

Группа: М8О-312Б-22

Москва, 2024

Камера и базовые 3D-трансформации

Цель лабораторной работы:
В этой лабораторной работе вы научитесь работать с камерой в 3D-пространстве, управлять её положением и направлением, а также освоите базовые трансформации (перемещение, поворот и масштабирование) объектов в 3D.

Вариант 7. Комбинация трансформаций (перемещение, масштабирование, поворот)
Постройте несколько объектов (например, куб и пирамиду) в разных местах сцены.
Реализуйте для каждого объекта возможность перемещения, масштабирования и поворота.
Управление должно осуществляться через клавиатуру, при этом трансформации должны применяться последовательно.
Дополнительно: Реализуйте интерфейс для переключения между объектами, чтобы трансформировать их по отдельности.

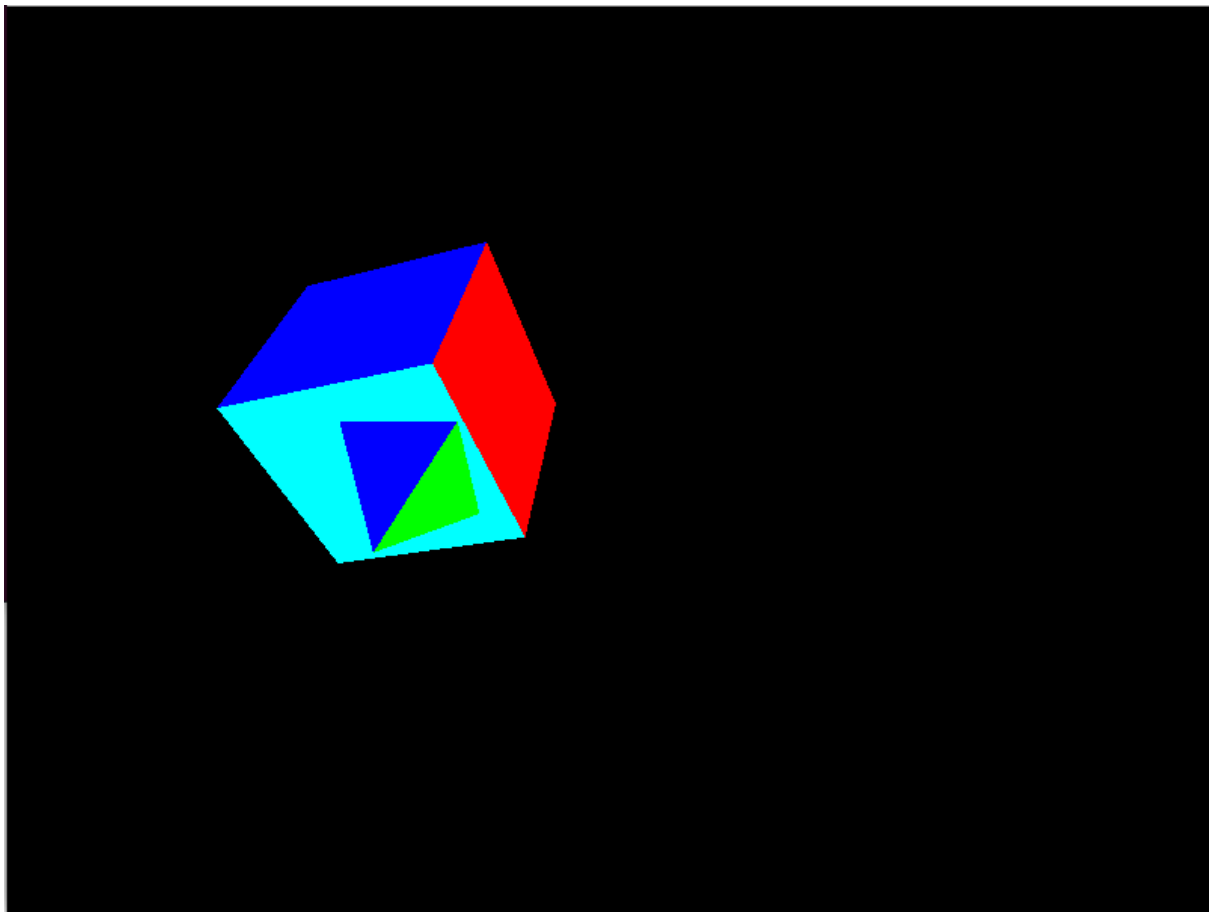
Решение

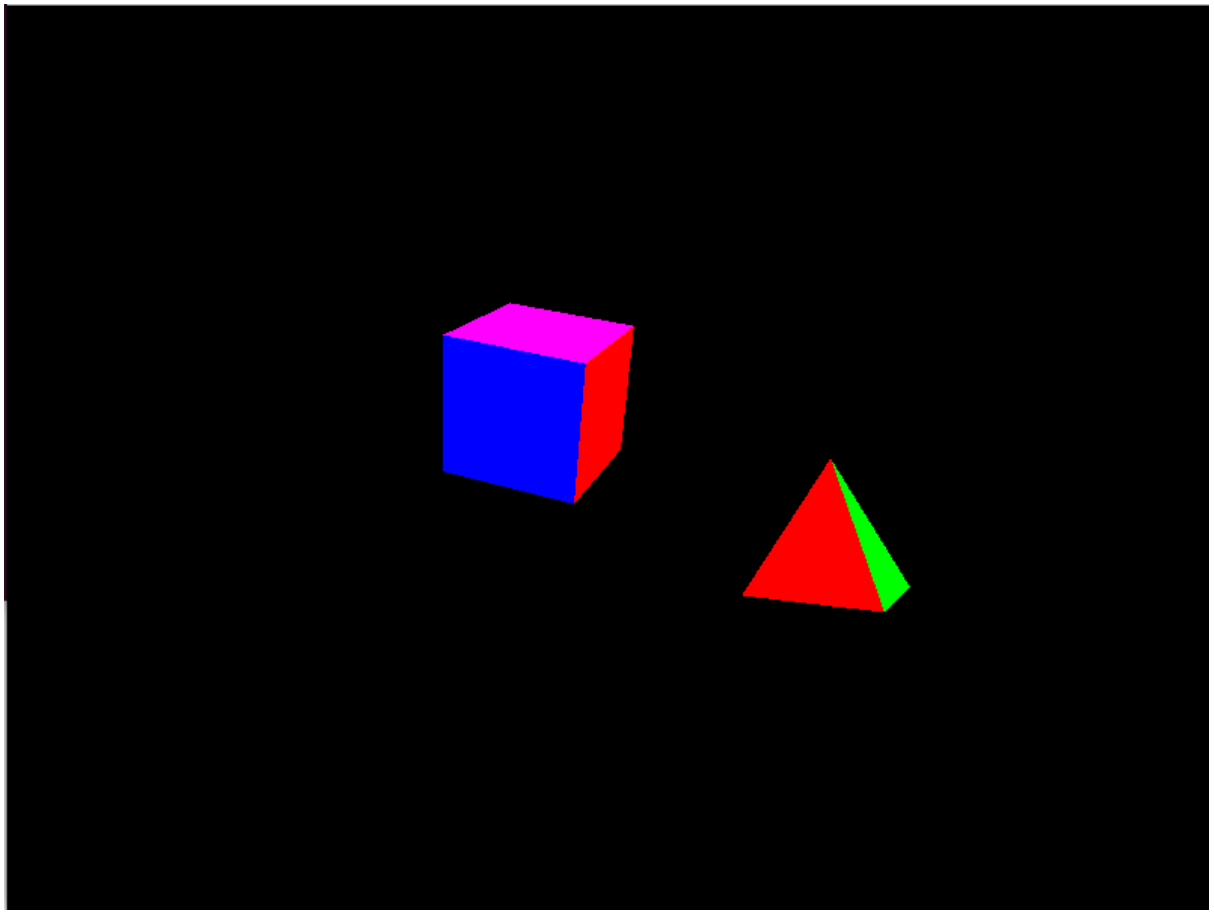
Создание классов для управления камерой и объектами:

- Класс Camera управляет позиционированием и поворотами камеры. Он позволяет перемещать камеру в 3D-пространстве с использованием углов наклона (pitch) и поворота (yaw), а также применять преобразования с использованием OpenGL функций для изменения вида сцены.
- Класс Transformable управляет объектами в сцене, такими как куб и пирамида. Он включает методы для перемещения, вращения и масштабирования объектов в 3D-пространстве. Также, как и в классе Camera, используются преобразования с помощью OpenGL.
- **Настройка OpenGL**:
- Использование OpenGL для рендеринга 3D-объектов и управления перспективой с помощью функции gluPerspective. Активируется тест глубины для корректной отрисовки объектов на различных расстояниях.
- Определение матрицы проекции и моделирования для преобразования координат объектов и камеры.
- **Обработка ввода от пользователя**:
- Управление положением камеры и объектов с помощью клавиш. Используется система обработки событий SFML для перемещения объектов по оси X, Y, Z, изменения их масштаба и поворота в 3D-пространстве. С помощью клавиши

Tab можно переключаться между активными объектами (камерой, кубом или пирамидой).

- Клавиши W, A, S, D управляют перемещением, а Q и E — масштабированием. Клавиши стрелок позволяют изменять углы наклона и поворота объектов.
- **Отрисовка объектов**:
- Для визуализации объектов использовалась функция рисования куба (drawCube) и пирамиды (drawPyramid), где каждый объект рисуется с использованием квадрантов и треугольников. Для каждого объекта применяется соответствующий цвет и координаты для формирования геометрической модели.
- **Обновление сцены**:
- Каждый кадр сцена обновляется с помощью очистки экрана и применения всех преобразований объектов и камеры. После этого объекты отрисовываются в новом положении и масштабе.





Вывод

В ходе выполнения лабораторной работы были изучены основы работы с OpenGL для создания и отображения 3D-объектов, а также способы взаимодействия с пользователем через клавиатуру. Реализованные методы управления камерой и объектами (перемещение, поворот, масштабирование) показали, как математические преобразования влияют на отображение сцены. Работа позволила расширить знания в области обработки ввода, настройки камеры и применения трансформаций для создания интерактивных графических приложений.

Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Компьютерная графика" №4

Студент: Андрюшин Л. Д.

Группа: М8О-312Б-22

Москва, 2024

Освещение и работа с шейдерами

Цель лабораторной работы:
В этой лабораторной работе вы научитесь работать с освещением в 3D-пространстве, используя различные типы источников света, и освоите основы написания шейдеров. Вы реализуете освещение объектов в сцене с использованием простейших моделей освещения и настроите эффекты при помощи вершинных и фрагментных шейдеров.

Вариант 7. Использование нормальных карт для создания детализации
Постройте куб и примените к нему текстуру.
Реализуйте нормальные карты (normal mapping) для добавления детализации на поверхности куба без увеличения количества полигонов.
Используйте фрагментный шейдер для расчета освещения с учетом нормальной карты.
Дополнительно: Добавьте возможность переключения между нормальной картой и стандартным затенением для сравнения результатов.

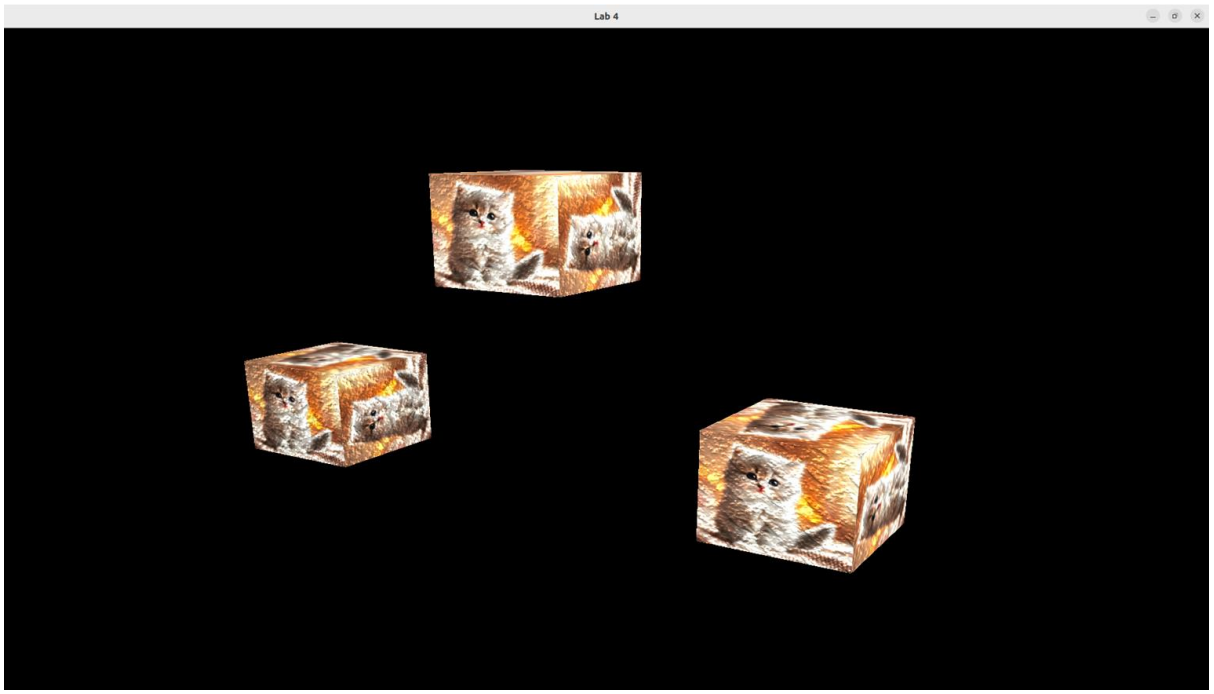
Решение

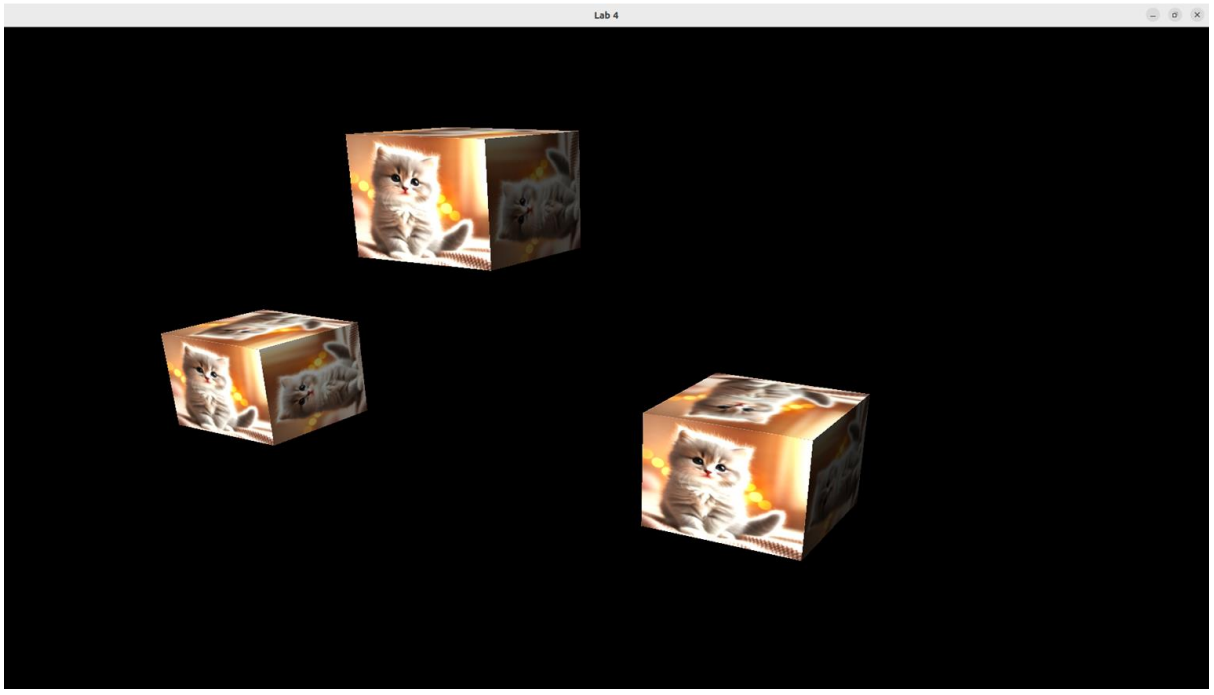
Инициализация камеры и управления: Для управления камерой была реализована система обработки ввода с клавиатуры и мыши. В частности, с помощью мыши можно изменять угол обзора (поворот камеры), а с клавиатуры — управлять движением камеры по осям. Камера определяется с использованием углов yaw и pitch, а также вектора направления. При изменении положения мыши вычисляются смещения и соответствующим образом обновляется позиция камеры.

- **Загрузка текстур**: Для добавления текстур на 3D-объекты используется функция loadTexture. Она загружает изображение с помощью библиотеки stb_image и применяет его к объектам сцены. Текстура обрабатывается с использованием OpenGL, где устанавливаются параметры фильтрации и повторения текстуры, а также генерируются мип-мапы для улучшения качества изображения на разных расстояниях.
- **Шейдеры**: Для рендеринга объектов используются вершинные и фрагментные шейдеры. Функции loadShaderSource и compileShader загружают исходный код шейдера и компилируют его, а функция createShaderProgram связывает вершинный и фрагментный шейдеры в одну программу. Эти шейдеры управляют отображением объектов на экране, включая текстуры и освещение.
- **Создание и рендеринг объекта**: Для отображения куба создается массив вершин, где каждая вершина описана координатами, нормалями и текстурными координатами. Функция setupCube инициализирует массив вершин и создаёт буфер для их передачи в видеокарту. Вершины описываются с помощью трех координат (позиция), нормали и текстурных координат, что позволяет применять текстуры и освещение на объект.

- **Обработка событий**: Взаимодействие с пользователем осуществляется через события, обрабатываемые библиотекой SFML. При изменении положения мыши обновляется направление камеры, а также осуществляется управление положением объекта в сцене. Реализована возможность изменения масштаба объекта с помощью клавиш, что позволяет динамично изменять размер отображаемого куба.
- **Загрузка текстур**: Для загрузки текстур и нормальных карт используется функция loadTexture. Она читает изображения с диска и преобразует их в текстуры, которые затем могут быть использованы в OpenGL. Если загрузка текстуры не удалась, программа выведет ошибку и завершится.
- **Создание шейдерной программы**: В программе используется два шейдера — вершинный и фрагментный. Эти шейдеры компилируются и соединяются в единую программу с помощью функции createShaderProgram. Если по какой-то причине шейдерная программа не создается, программа также завершится с ошибкой.
- **Настройка объектов (например, кубов)**: Кубы создаются с помощью функции setupCube, которая настраивает буферы и массивы для хранения вершин и их атрибутов. Это необходимо для того, чтобы OpenGL знал, как отрисовывать кубы.
- **Обработка пользовательского ввода**: Камера в сцене управляется с помощью клавиш для движения (W, A, S, D) и мыши для поворота. Когда мышь перемещается, программа вычисляет смещение и обновляет угол наклона камеры, позволяя пользователю вращать ее по осям X и Y. Также при нажатии клавиши можно переключать использование нормальной карты для отображения поверхности.
- **Переключение между нормальной картой и стандартным затенением**: Когда пользователь нажимает клавишу "N", происходит переключение между использованием нормальной карты и стандартным затенением. Это контролируется флагом useNormalMapping, который передается в шейдеры для изменения визуализации объекта.
- **Рендеринг объектов с использованием нормальных карт и освещения**: Все объекты (в данном случае кубы) рендерятся с учетом направленного света и точечного света, их параметры передаются в шейдеры. В зависимости от того, включена ли нормальная карта, будет использоваться соответствующий способ затенения для объектов. Параметры камеры (позиция) и источников света передаются в шейдеры через униформы.
- **Отрисовка кубов**: Для каждого куба рассчитывается его модельная матрица, которая описывает его положение в пространстве. Эта матрица передается в шейдер, а затем кубы рисуются с помощью команды glDrawArrays, которая отрисовывает треугольники, составляющие куб.
- **Очистка ресурсов**: После завершения работы программы ресурсы, такие как вершинные массивы (VAO), буферы (VBO) и шейдерные программы,

очищаются. Это важно для предотвращения утечек памяти и для корректного завершения работы программы.





Вывод

В процессе выполнения лабораторной работы были изучены методы шейдинга в OpenGL, а также основные принципы работы с 3D-объектами и освещением. Были реализованы два типа шейдинга: плоский и Gouraud, что позволило продемонстрировать различные подходы к освещению и рендерингу поверхностей в 3D-графике. Взаимодействие света с объектом на сцене было визуализировано с помощью нормалей и применения матричных преобразований для вращения и отображения куба. Работа углубила понимание использования шейдеров и продемонстрировала важность правильного выбора методов освещения для достижения реалистичных визуальных эффектов в 3D-пространстве.

Московский авиационный институт (национальный исследовательский университет)

Институт информационных технологий и прикладной математики

«Кафедра вычислительной математики и программирования»

Лабораторная работа по предмету "Компьютерная графика" №5

Студент: Андрюшин Л. Д.

Группа: М8О-312Б-22

Москва, 2024

Трассировка лучей (Ray Tracing)

Цель лабораторной работы:
В этой лабораторной работе вы научитесь работать с техникой трассировки лучей для создания реалистичной 3D-графики. Вы реализуете алгоритм Ray Tracing, который позволяет рассчитывать физически корректные отражения, преломления, тени и свет в сцене. Лабораторная работа подводит к пониманию основ рендеринга, работающего с лучами света, а также к созданию реалистичных сцен.

Вариант 7. Отражения и текстурированные поверхности
Постройте сцену с двумя текстурированными плоскостями (стена и пол) и одной сферой.
Реализуйте трассировку лучей с поддержкой текстурирования объектов.
Включите отражения на сфере и учтите отраженные текстуры на её поверхности.
Дополнительно: Реализуйте управление зеркальностью поверхности сферы для изменения интенсивности отражений.

Решение

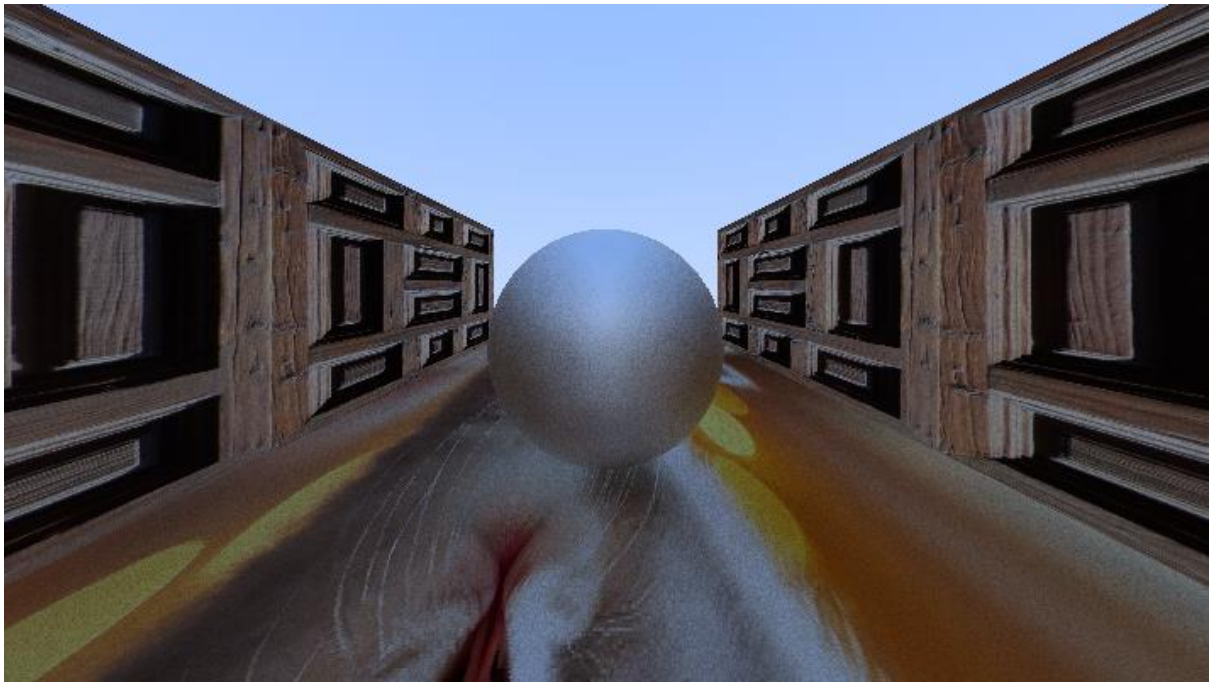
Метод решения состоит в реализации рендеринга 3D-сцены с использованием техники трассировки лучей для вычисления цвета каждого пикселя. В процессе работы программы создаются объекты сцены, такие как сферы и плоскости, с заданными текстурами и материалами.
В файле Scene.cpp основная логика построена вокруг функции get_color, которая находит цвет для каждого луча, пересекающего объекты сцены. Для этого программа проверяет, пересекает ли луч какие-либо формы, и, если да, вычисляет точку пересечения и нормаль поверхности. В зависимости от материала объекта (зеркального или диффузного) выбирается подходящий алгоритм для дальнейшего распространения луча: либо отражение луча (если объект зеркальный), либо рассеяние на основе случайного направления (для диффузных материалов).

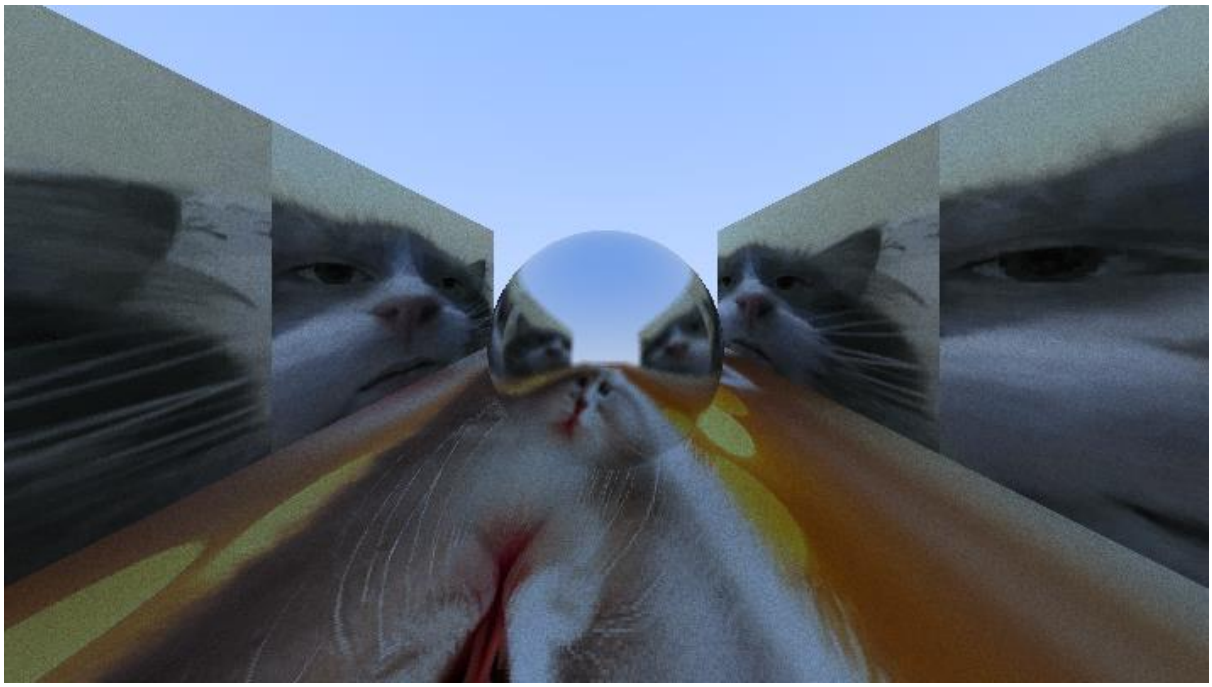
Также реализован механизм глубины рекурсии, который позволяет учитывать несколько отражений или рассеяний от объектов сцены. Каждый новый луч в рекурсии, в свою очередь, пересекает объекты и продолжает передавать информацию о цвете и свете, пока не достигнет максимальной глубины или не попадет в пустое пространство.

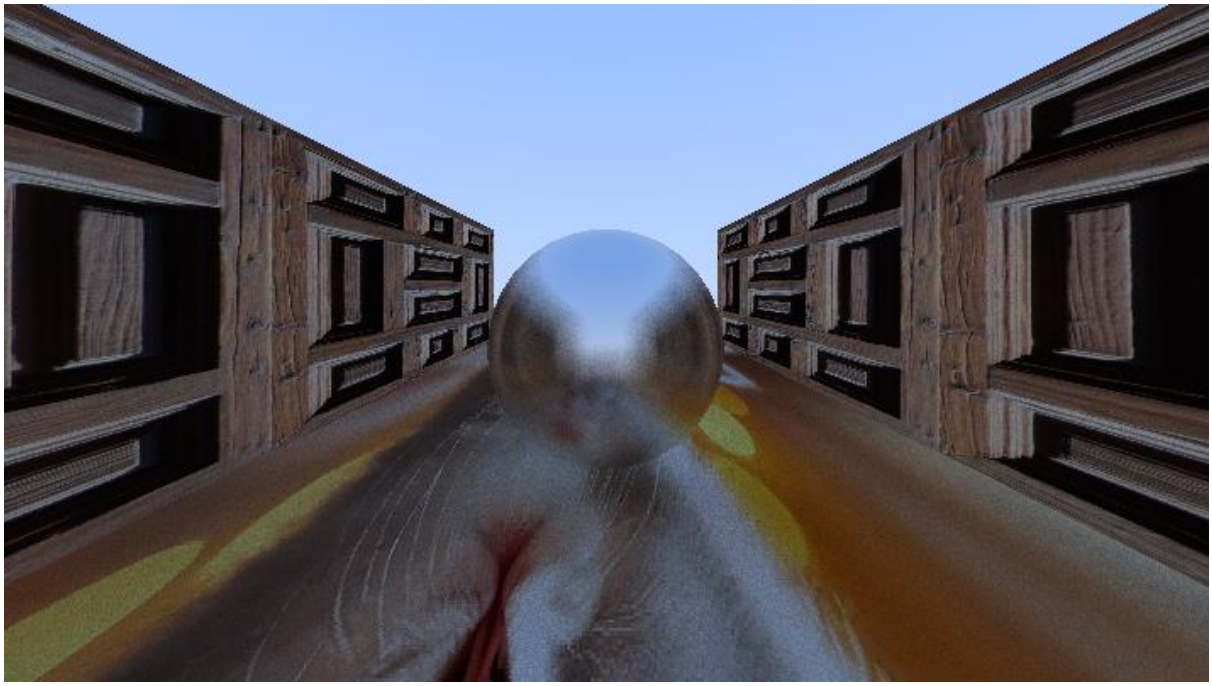
Основная функция рендеринга render выполняет трассировку лучей для каждого пикселя изображения, используя камеру для вычисления направления лучей. Для повышения качества изображения применяется метод суперсэмплинга: каждый пиксель обрабатывается несколько раз с различными сдвигами, после чего результаты усредняются.

Для оптимизации работы программа использует параллельные вычисления, чтобы ускорить обработку большого количества пикселей, что достигается через использование директивы #pragma omp parallel for. Это позволяет значительно улучшить производительность при рендеринге.

На основе полученных данных о цветах каждого пикселя создается финальное изображение, которое затем сохраняется в файл. Параметр зеркальности объекта можно настроить через аргументы командной строки, что позволяет изменять визуальные эффекты для отражающих объектов.







Вывод

В результате выполнения лабораторной работы была реализована сцена с использованием алгоритма трассировки лучей, которая включает объекты,

такие как сферы и плоскости, с эффектами освещения, отражений и мягких теней. Были изучены ключевые методы трассировки лучей, включая расчет пересечений лучей с объектами и применение различных моделей освещения, таких как модель Фонга для диффузного и зеркального освещения. Также была реализована методика создания мягких теней с использованием случайных направлений для смещения. Полученные результаты позволили углубить знания в области рендеринга 3D-сцен, освоить применение алгоритмов трассировки лучей и научиться учитывать физические эффекты освещения для достижения более реалистичной визуализации.