Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1 по курсу
«Компьютерная графика»

Студент: Ажаурова А.К.
Группа: М8О-303Б-22
Преподаватель: Филиппов Г.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

1. Цель 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.

2. Постановка задачи

Требования

Вы должны использовать C++ (OpenGL + SFML) или C# (OpenTK).

Программа должна работать в реальном времени, обновляя изображение в цикле.

Визуальный результат необходимо продемонстрировать на экране с возможностью управления через интерфейс.

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.

3. Общие сведения о программе

Данная программа реализует анимацию 2D-шестиугольника с использованием OpenGL. В программе используется линейная интерполяция для плавного изменения положения, размера и формы объекта в пространстве. Шестиугольник перемещается по экрану, изменяет свой размер и вращается. Анимация выполняется в цикле с возможностью динамического управления через клавиатуру, позволяя пользователю изменять скорость перемещения и вращения объекта. Программа демонстрирует основы работы с графическими API для отрисовки и анимации 2D-примитивов с применением матричных операций для трансформаций.

4. Общий алгоритм решения

1. *Инициализация OpenGL и GLFW*: Программа начинается с инициализации библиотеки GLFW, создается окно с использованием OpenGL для рисования.

2. *Генерация вершин шестиугольника*: Вершины шестиугольника рассчитываются с использованием формул на основе синуса и косинуса для равномерного распределения точек по окружности радиусом R.

3. *Линейная интерполяция*: Для плавного изменения параметров (положение и радиус) объекта используется линейная интерполяция. Значения изменяются между начальной и конечной точками, создавая эффект плавного движения.

4. *Применение трансформаций*: Для перемещения и вращения шестиугольника используются стандартные 2D-трансформации:
   - *Поворот*: осуществляется с использованием тригонометрических функций для вычисления новых координат.
   - *Перемещение*: выполняется с добавлением смещений к текущим координатам.

5. *Цикл анимации*: Основной цикл программы обновляет экран, применяет все трансформации и отображает результат.

6. *Обработка ввода с клавиатуры*: Программа реагирует на клавиши для изменения скорости движения и вращения шестиугольника (вверх/вниз для скорости перемещения и влево/вправо для изменения угловой скорости).

7. *Циклическая анимация*: Программа реализует циклическую анимацию, в которой объект изменяет форму и возвращается в исходное состояние, создавая эффект непрерывного движения.

5. Реализация

Реализация программы состоит из нескольких ключевых компонентов:

1. *Генерация шестиугольника*: Для генерации координат вершин шестиугольника используется цикл, в котором для каждой вершины рассчитывается угол и соответствующие координаты на окружности.

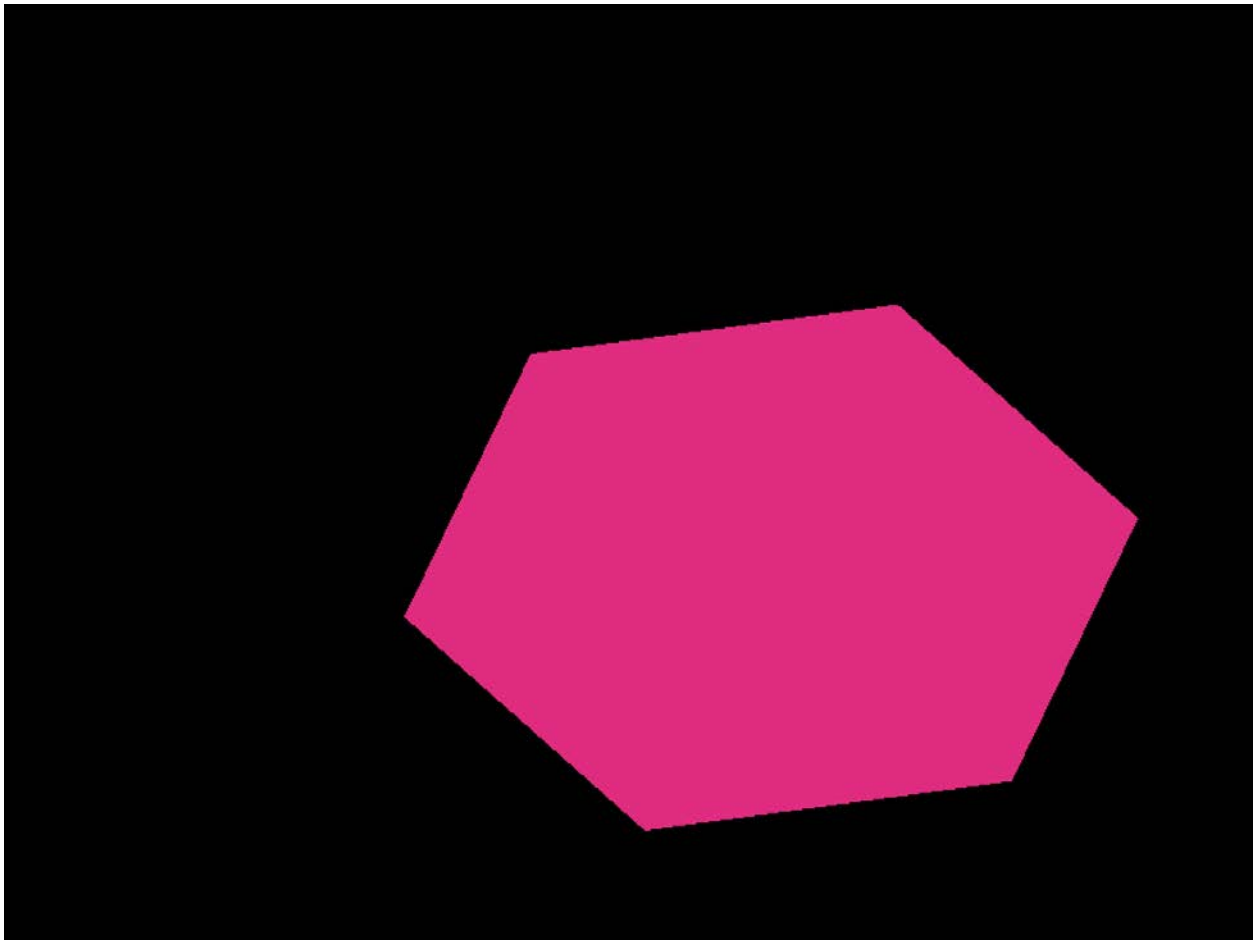
2. *Интерполяция*: Используется функция линейной интерполяции для плавного изменения позиции и радиуса шестиугольника в зависимости от времени.

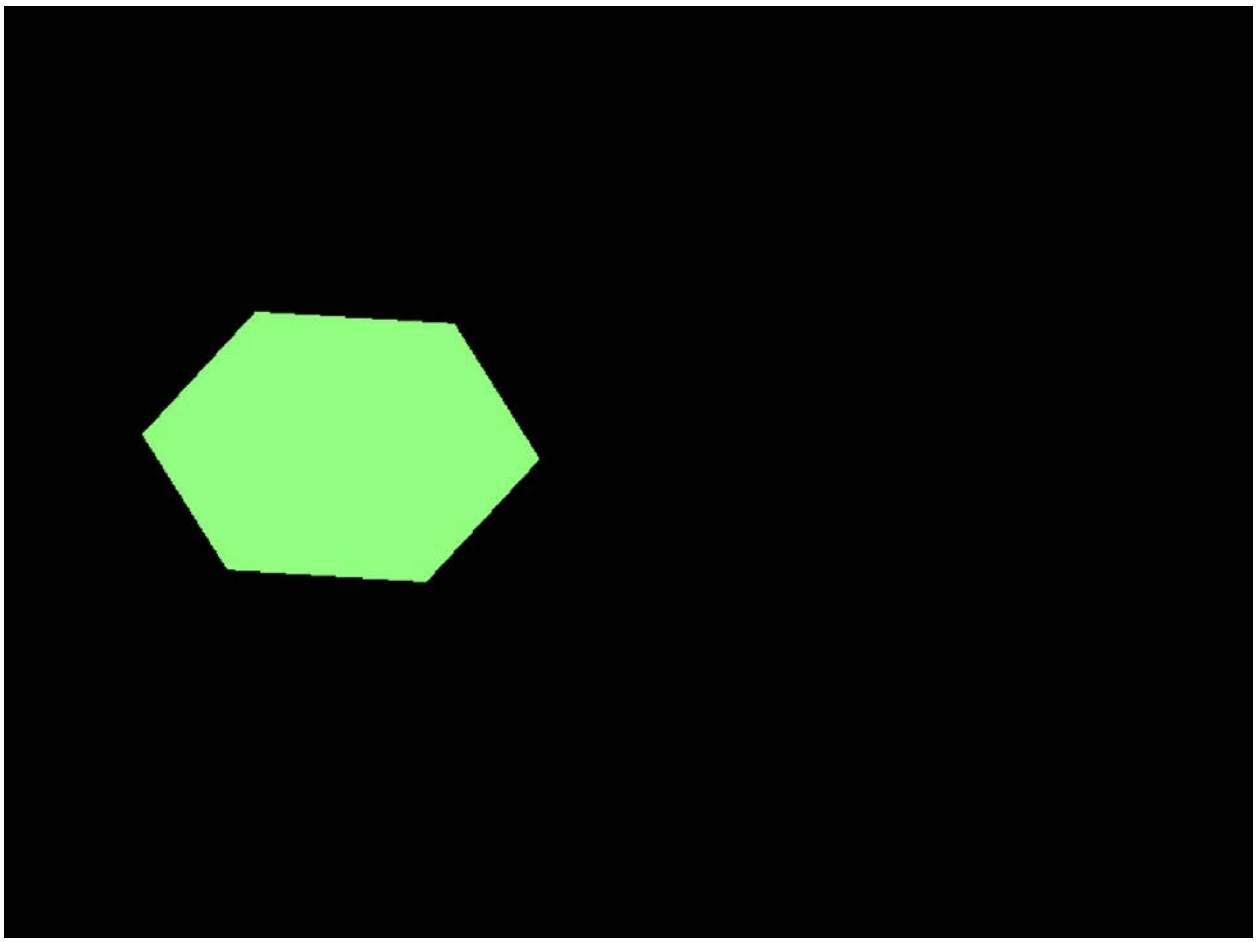
3. *Трансформации*: Для поворота шестиугольника применяется стандартная матричная формула поворота, которая использует тригонометрические функции для вычисления новых координат вершин после поворота. Для перемещения добавляются сдвиги к координатам.

4. *Отрисовка*: Отрисовка шестиугольника осуществляется через OpenGL, используя команду *glBegin(GL_POLYGON)*, где каждой вершине присваивается цвет с использованием функции *glColor3f*.

5. *Обработка клавиатурного ввода*: Для управления скоростью и направлением анимации используются клавиши стрелок, которые изменяют параметры скорости и угловой скорости поворота.

6. Пример работы





7. Выводы

В результате выполнения лабораторной работы были достигнуты следующие результаты:

1. Освоены основы работы с графическим API OpenGL для отрисовки и анимации 2D-объектов.
2. Изучены методы линейной интерполяции для плавного изменения параметров объекта (размера, положения).
3. Реализованы базовые 2D-трансформации, такие как поворот и перемещение, для изменения положения и ориентации многоугольника на экране.
4. Реализована циклическая анимация, где многоугольник изменяет свою форму и возвращается в исходное состояние.
5. Программа предоставляет удобный интерфейс для управления анимацией через клавиатуру, что позволяет динамически изменять параметры анимации в реальном времени.

Таким образом, лабораторная работа помогла получить практические навыки в области 2D-графики, трансформаций объектов и анимации, а также усовершенствовать навыки работы с OpenGL для создания визуальных эффектов в реальном времени.

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №2 по курсу
«Компьютерная графика»

Студент: Ажаурова А.К.
Группа: М8О-303Б-22
Преподаватель: Филиппов Г.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

1. Цель 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.

2. Постановка задачи

Требования

Вы должны использовать OpenGL + SFML или OpenTK + C#.

Программа должна работать в реальном времени, с возможностью динамической смены проекции и трансформаций объектов.

Все объекты должны корректно отрисовываться с учетом проекции и иметь возможность взаимодействия с пользователем.

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.

3. Общие сведения о программе

Программа представляет собой 3D-сцену, на которой отображаются три объекта: куб, пирамида и сфера. Для отображения сцены используется OpenGL с помощью библиотеки SFML. Камера сцены позволяет перемещать её по осям X и Y, а также изменять расстояние до сцены, управляя глубиной отображения объектов. Все объекты отображаются с учетом перспективной проекции, что обеспечивает правильное восприятие глубины и размеров объектов на экране.

Программа также предусматривает взаимодействие с пользователем через клавиатуру. Пользователь может управлять движением камеры, а также изменять параметры сцены с помощью нажатий клавиш.

4. Общий алгоритм решения

1. *Инициализация OpenGL и создание окна*: Создается окно с использованием библиотеки SFML, а также инициализируется OpenGL для рендеринга 3D-сцены.

2. *Установка камеры*: Для отображения объектов используется перспектива, которая задается с помощью матрицы проекции в OpenGL. Камера вращается и перемещается в пространстве с помощью изменения углов и расстояния до сцены.

3. *Отрисовка 3D-объектов*:
   - Куб отрисовывается с использованием команд OpenGL для рисования квадратных граней.
   - Пирамида рисуется с использованием треугольных граней и квадратной основы.
   - Сфера рисуется с помощью квадрики OpenGL (gluSphere), которая позволяет создавать сферы с заданным радиусом и сегментацией.

4. *Динамическое изменение проекции и трансформаций*: Камера может изменять свою ориентацию (вращение по осям X и Y), а также приближаться и удаляться от сцены. Эти действия реализованы с помощью клавиш, которые влияют на параметры камеры.

5. *Обработка ввода с клавиатуры*: Программа реагирует на клавиши, позволяя пользователю управлять движением камеры: вращать её по осям X и Y, а также изменять расстояние от объектов (приближать или удалять камеру).

6. *Рендеринг сцены*: В основном цикле программы каждый объект отрисовывается с использованием соответствующих функций. После отрисовки сцены окно обновляется для отображения изменений.

5. Реализация

1. *Создание окна и инициализация OpenGL*:

```
sf::Window window(sf::VideoMode(800, 600), "3D Scene", sf::Style::Default, sf::ContextSettings(32));
window.setFramerateLimit(60); // Ограничение кадров в секунду
glEnable(GL_DEPTH_TEST); // Включение теста глубины для 3D-отрисовки
```

2. *Установка проекционной матрицы*: Для применения перспективной проекции используется функция *gluPerspective*, которая задает угол обзора и границы отображаемой области.

```
glMatrixMode(GL_PROJECTION); // Установка режима проекционной матрицы
gluPerspective(45.0, 800.0 / 600.0, 1.0, 100.0); // Перспективная проекция
glMatrixMode(GL_MODELVIEW); // Установка режима модельно-видовой матрицы
```

3. *Отрисовка куба*: Куб создается с помощью команд *glBegin(GL_QUADS)* для рисования квадратных граней:

```
glBegin(GL_QUADS);
glColor3f(1, 0.5, 0); // Задняя грань (оранжевая)
// Задание координат каждой вершины
glEnd();
```

4. *Отрисовка пирамиды*: Пирамида рисуется с использованием треугольных граней для боковых сторон и квадратной основы:

```
glBegin(GL_TRIANGLES); // Боковые грани пирамиды
glColor3f(1, 0, 0); // Грань 1 (красная)
// Задание координат для каждой грани
glEnd();
```

5. *Отрисовка сферы*: Сфера создается с использованием библиотеки GLU, которая предоставляет функцию *gluSphere* для рисования сферы с заданными параметрами:

```
GLUquadric *quad = gluNewQuadric(); // Создание квадрики
gluSphere(quad, 1.0, 32, 32); // Радиус сферы и сегментация
gluDeleteQuadric(quad); // Удаление квадрики
```

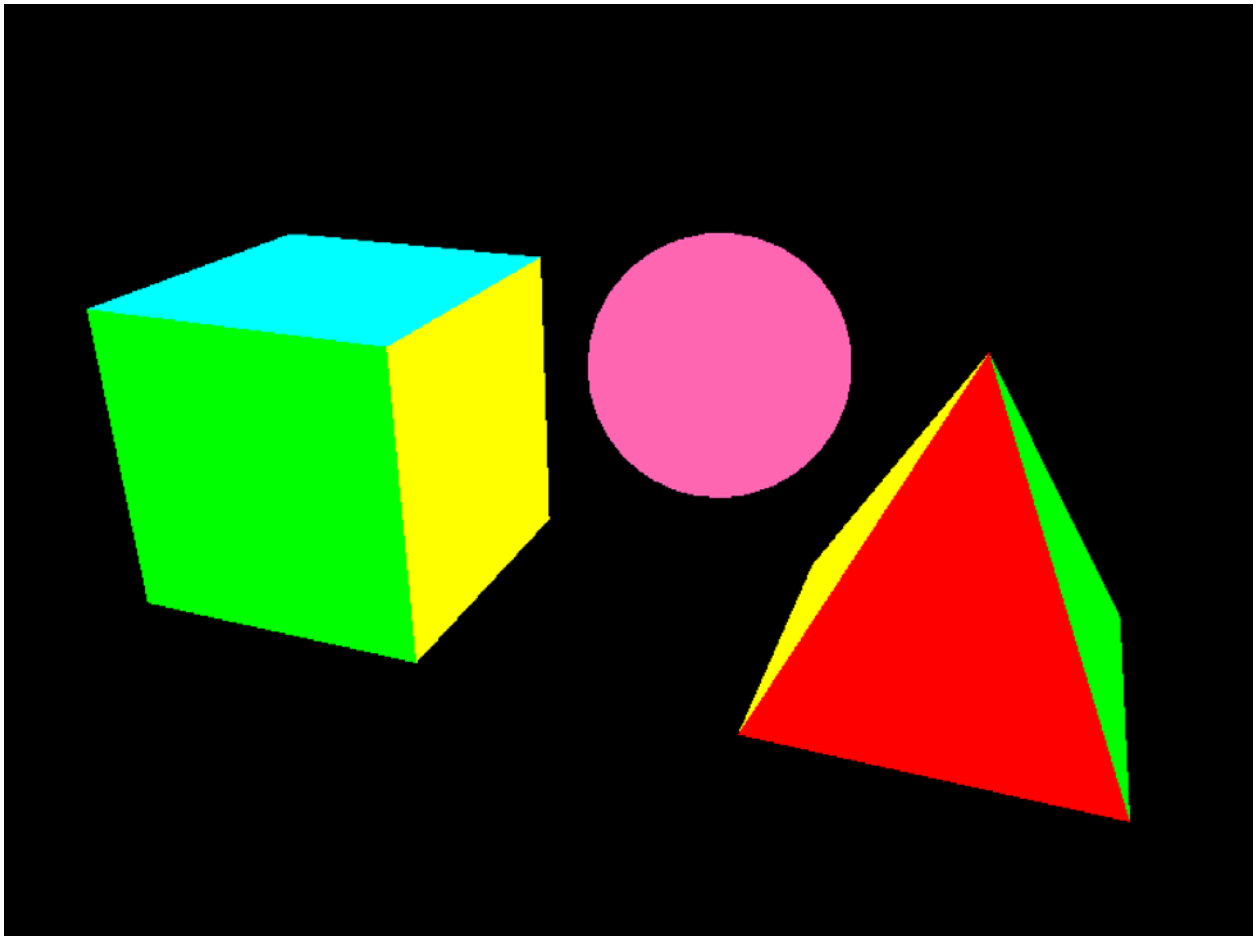
6. *Перемещение камеры*: Камера изменяет свою ориентацию и положение с помощью клавиш:

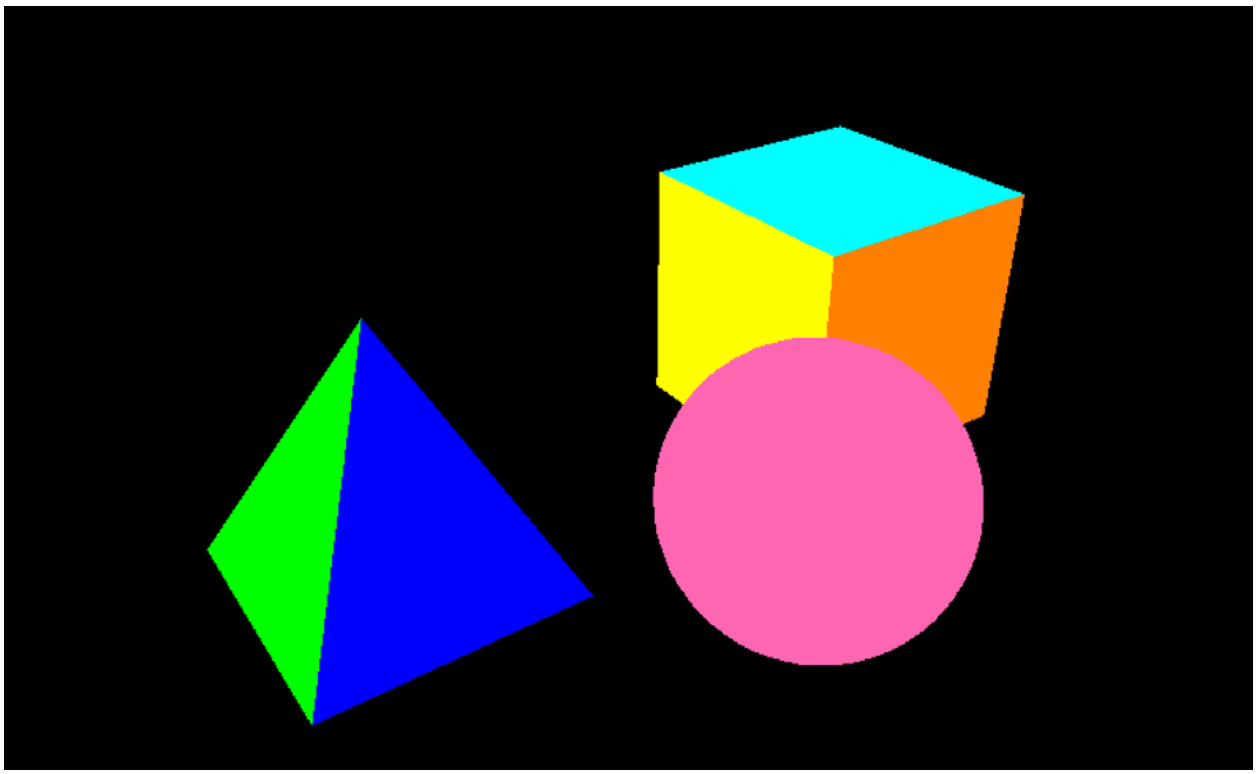
```
if (event.key.code == sf::Keyboard::Up) cameraAngleX -= 5;
if (event.key.code == sf::Keyboard::Down) cameraAngleX += 5;
if (event.key.code == sf::Keyboard::Left) cameraAngleY -= 5;
if (event.key.code == sf::Keyboard::Right) cameraAngleY += 5;
if (event.key.code == sf::Keyboard::Equal) cameraDistance -= 0.5;
if (event.key.code == sf::Keyboard::Hyphen) cameraDistance += 0.5;
```

7. *Обновление сцены*: Сцена обновляется каждый кадр, отрисовывая объекты и обновляя их состояние в соответствии с вводом пользователя:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); // Очистка экрана
setupCamera(); // Установка камеры
drawCube(); // Отрисовка куба
drawPyramid(); // Отрисовка пирамиды
drawSphere(); // Отрисовка сферы
window.display(); // Отображение сцены
```

6. Пример работы





7. Выводы

1. Основы 3D-графики: В ходе выполнения лабораторной работы были изучены основные принципы 3D-графики, такие как создание и трансформация 3D-объектов, использование матриц для проекций, а также работа с различными типами проекций (перспективной и ортографической).

2. Реализация сцены: Было построено простое 3D-сценарное пространство, содержащее несколько объектов, таких как куб, пирамида и сфера. Объекты были отрисованы с использованием стандартных примитивов OpenGL.

3. Управление камерой: Реализовано управление камерой с помощью клавиш, что позволяет пользователю перемещать камеру по сцене, вращать её по осям и изменять расстояние до объектов.

4. Интерактивность и анимация: Программа позволяет динамически изменять параметры сцены, что улучшает восприятие 3D-графики и добавляет интерактивности в процессе работы с программой.

Таким образом, лабораторная работа способствует укреплению навыков работы с OpenGL и концепциями 3D-графики.

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №3 по курсу
«Компьютерная графика»

Студент: Ажаурова А.К.
Группа: М8О-303Б-22
Преподаватель: Филиппов Г.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

1. Цель 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.

2. Постановка задачи

Требования

Вы должны использовать OpenGL + SFML или OpenTK + C#.

Программа должна работать в реальном времени, с возможностью динамической смены положения камеры и трансформаций объектов. Управление камеры должно быть назначено на клавиатуру или мышь

Все объекты должны корректно отрисовываться с учетом положения камеры и примененных трансформаций.

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.

3. Общие сведения о программе

Программа реализует работу с 3D-объектами в виртуальной сцене, предоставляя возможность управлять их положением, поворотом и масштабированием. Для взаимодействия используется библиотека SFML (для обработки событий и управления окном) и OpenGL (для рендеринга 3D-сцены). Камера, управляемая пользователем, позволяет перемещаться и изменять угол обзора. Программа предоставляет интерфейс для переключения между объектами (например, куб и пирамида) и трансформации каждого из них в реальном времени.

Целью программы является обучение базовым 3D-трансформациям и работе с камерой, что способствует освоению принципов построения трехмерных сцен.

4. Общий алгоритм решения

Основные этапы:

1. *Инициализация окна и настроек OpenGL:*
   - Создание окна с использованием SFML.
   - Настройка буфера глубины и проекционной матрицы.
2. *Создание объектов сцены*:
   - Реализация базового класса *SceneObject* для объектов.
   - Добавление классов для куба и пирамиды с их геометрическими особенностями.
3. *Настройка камеры*:
   - Использование переменных для хранения позиции и поворота камеры.
   - Реализация управления камерой через клавиатуру.
4. *Реализация трансформаций объектов*:
   - Перемещение, поворот и масштабирование объектов с помощью матричных операций OpenGL.
   - Обработка событий для динамического изменения трансформаций.
5. *Рендеринг сцены*:
   - Очистка экрана и буфера глубины.
   - Применение трансформаций камеры.
   - Отрисовка каждого объекта с учетом его индивидуальных трансформаций.

5. Реализация

1. *Преобразование модели (Model Transform)*: Применяется для изменения положения, ориентации и масштаба объекта относительно его локальной системы координат. Используются матрицы:

- Перемещение:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Поворот вокруг оси X, Y или Z:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Масштабирование:

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. *Преобразование камеры (View Transform)*: Для перемещения камеры используются аналогичные матрицы, но с инверсными значениями. Например, перемещение камеры назад эквивалентно перемещению всех объектов вперед.

3. *Перспективная проекция*: Используется матрица проекции, чтобы отображать удаленные объекты меньшего размера:

$$P = \begin{bmatrix} \frac{1}{\tan(fov/2) \cdot aspect} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(fov/2)} & 0 & 0 \\ 0 & 0 & \frac{z_f + z_n}{z_n - z_f} & \frac{2 z_f z_n}{z_n - z_f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Ключевые моменты реализации:

1. *Инициализация OpenGL*: Настраивается перспектива с помощью функции *glFrustum*.

```
glEnable(GL_DEPTH_TEST);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glFrustum(-fW, fW, -fH, fH, zNear, zFar);
glMatrixMode(GL_MODELVIEW);
```

2. *рансформации объектов*: Для каждого объекта применяются последовательные преобразования:

```
glTranslatef(transform.position.x, transform.position.y, transform.position.z);
glRotatef(transform.rotation.x, 1, 0, 0);
glRotatef(transform.rotation.y, 0, 1, 0);
glRotatef(transform.rotation.z, 0, 0, 1);
glScalef(transform.scale.x, transform.scale.y, transform.scale.z);
```

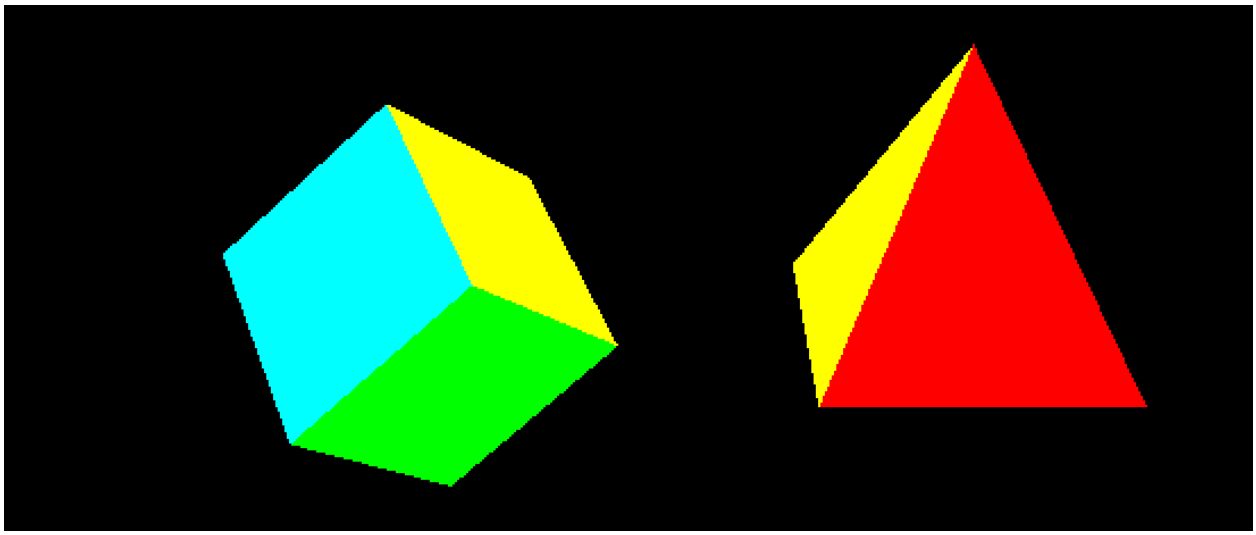
3. *Управление камерой*: Положение камеры изменяется через *glTranslatef* и *glRotatef* в матрице вида:

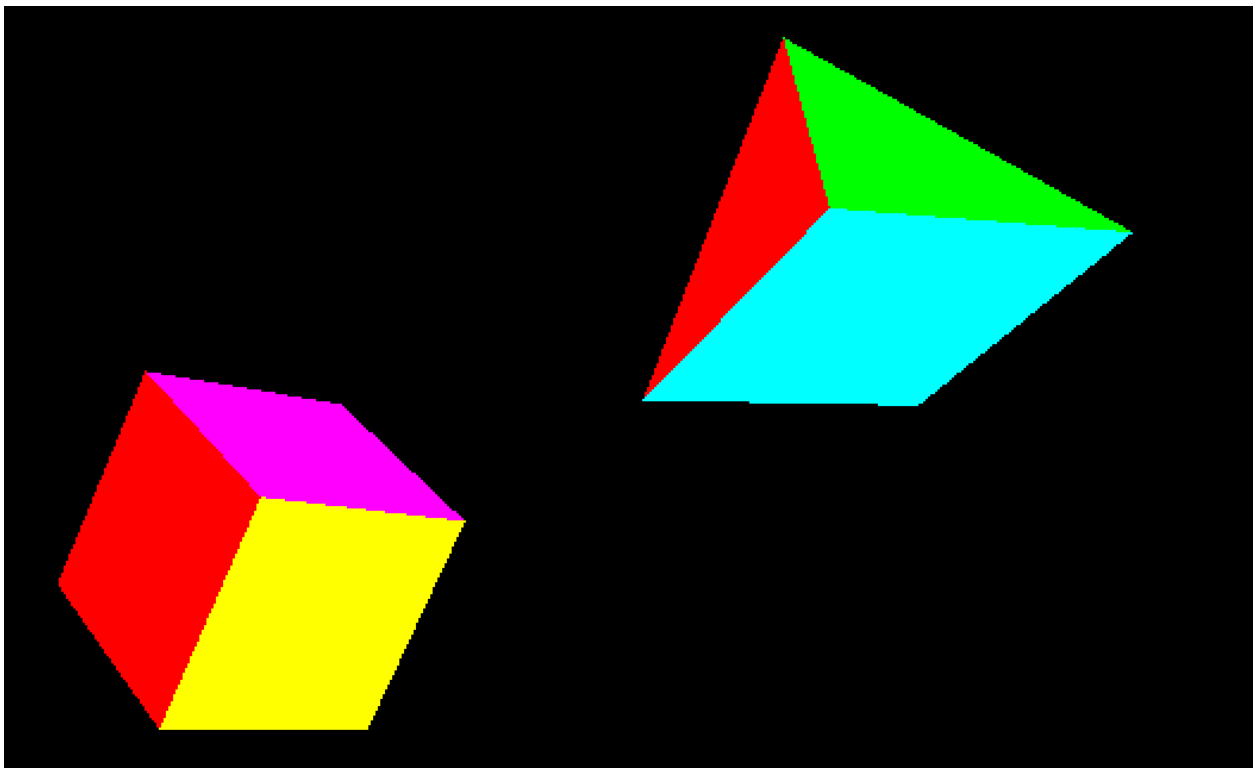
```
glLoadIdentity();
glRotatef(cameraRotation.x, 1, 0, 0);
glRotatef(cameraRotation.y, 0, 1, 0);
glTranslatef(-cameraPosition.x, -cameraPosition.y, -cameraPosition.z);
```

4. *Управление объектами*: Смена активного объекта осуществляется через индексацию массива:

```
SceneObject* currentObject = objects[currentObjectIndex];
```

2. Пример работы





3. Выводы

В данной лабораторной работе была реализована базовая 3D-сцена с управляемыми объектами (куб и пирамида) и камерой. Программа демонстрирует применение ключевых математических методов для работы с трехмерными трансформациями. Управление трансформациями позволяет эффективно изучить свойства перемещения, поворота и масштабирования.

Достижения:

1. Освоены базовые принципы работы с камерой и объектами в 3D-пространстве.
2. Реализовано динамическое управление трансформациями в реальном времени.
3. Изучен процесс создания сцены с несколькими объектами и применения к ним различных трансформаций.

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4 по курсу
«Компьютерная графика»

Студент: Ажаурова А.К.
Группа: М8О-303Б-22
Преподаватель: Филиппов Г.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

1. Цель 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.

2. Постановка задачи

Требования

Вы должны использовать OpenGL + SFML или OpenTK + C#.

Программа должна корректно отображать освещение с учетом типов источников света, используя написанные вами шейдеры.

В каждом варианте задания должны быть задействованы как минимум один тип освещения

(например, направленное освещение, точечный источник света или прожектор).

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.

3. Общие сведения о программе

Программа реализует освещение в 3D-пространстве с использованием OpenGL и SFML, применяя нормальные карты (normal mapping) для детализации поверхности куба. В ней используются вершинный и фрагментный шейдеры для расчета освещения и затенения, а также механизмы переключения между нормальной картой и стандартным затенением. Это позволяет визуально сравнивать результаты и оценивать влияние нормальных карт на реалистичность изображения.

4. Общий алгоритм решения

1. *Инициализация:*
   - Подготовка окна приложения с использованием SFML.
   - Настройка OpenGL (включение буфера глубины, шейдеров и текстур).
   - Загрузка данных для куба (вершины, индексы, текстуры).

2. *Создание шейдеров*:
   - Вершинный шейдер: вычисление позиции вершин, текстурных координат и TBN-матрицы (тангент-битангента-нормаль).
   - Фрагментный шейдер: реализация нормального затенения и затенения с использованием нормальных карт.

3. *Работа с буферами*:
   - Создание VAO, VBO и EBO для управления вершинными данными.
   - Установка атрибутов для передачи позиции, нормали, текстурных координат, тангентов и битангента.

4. *Основной цикл*:
   - Обработка пользовательских событий, включая переключение между нормальной картой и стандартным затенением.
   - Вычисление модельной, видовой и проекционной матриц.
   - Передача данных в шейдеры.
   - Рендеринг куба с использованием текущего метода освещения.

5. *Очистка ресурсов*:
   - Удаление созданных буферов, текстур и шейдерных программ.

5. Реализация

1. *Настройка шейдеров*:
   - Фрагментный шейдер получает нормали либо из нормальной карты, либо из вершинного шейдера в зависимости от переключателя *useNormalMap*.
   - Вершинный шейдер формирует TBN-матрицу для преобразования нормалей из пространства текстур в мировое пространство.

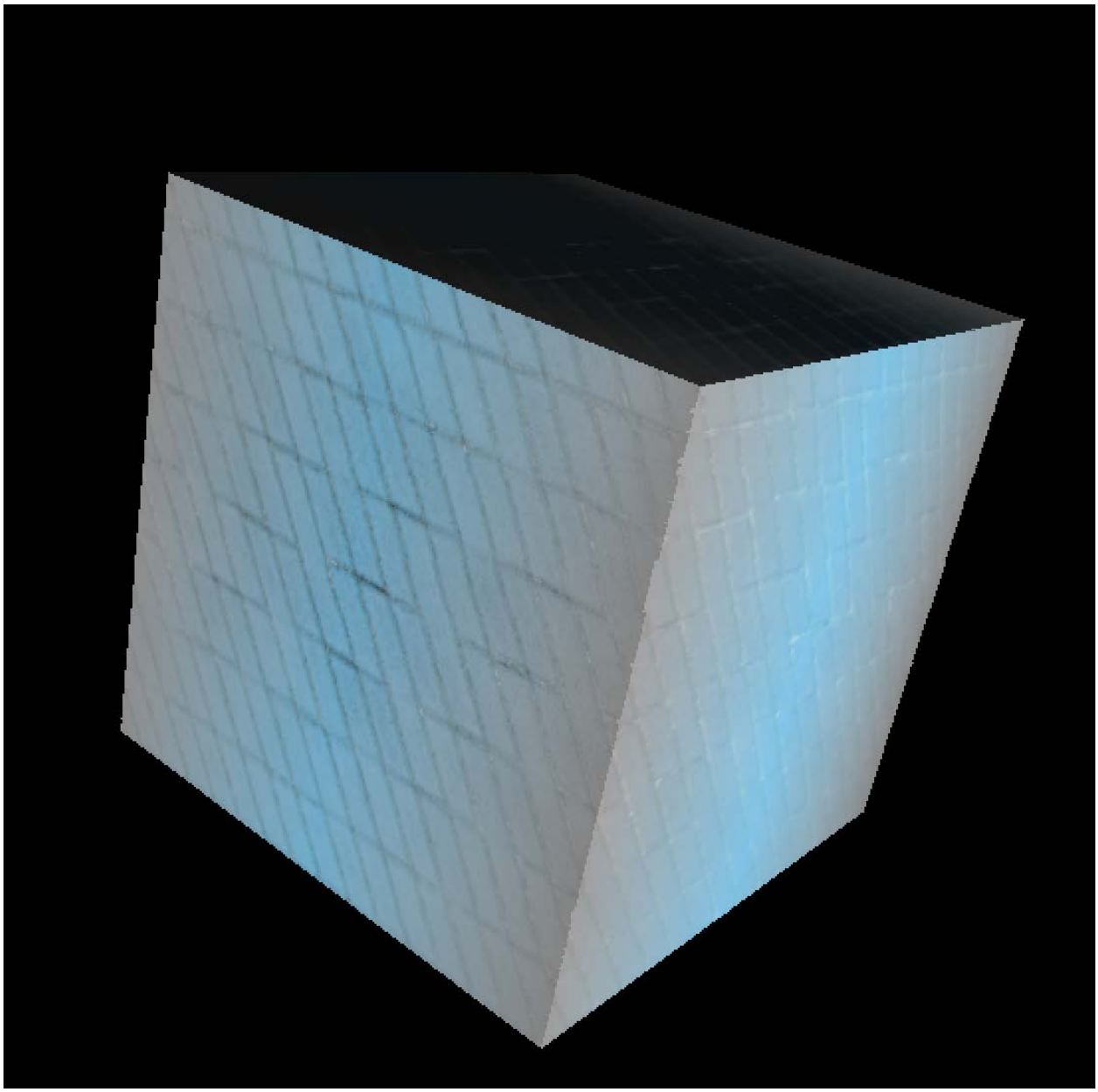
2. *Освещение*:
   - Вычисляется диффузное и зеркальное освещение с использованием нормалей.

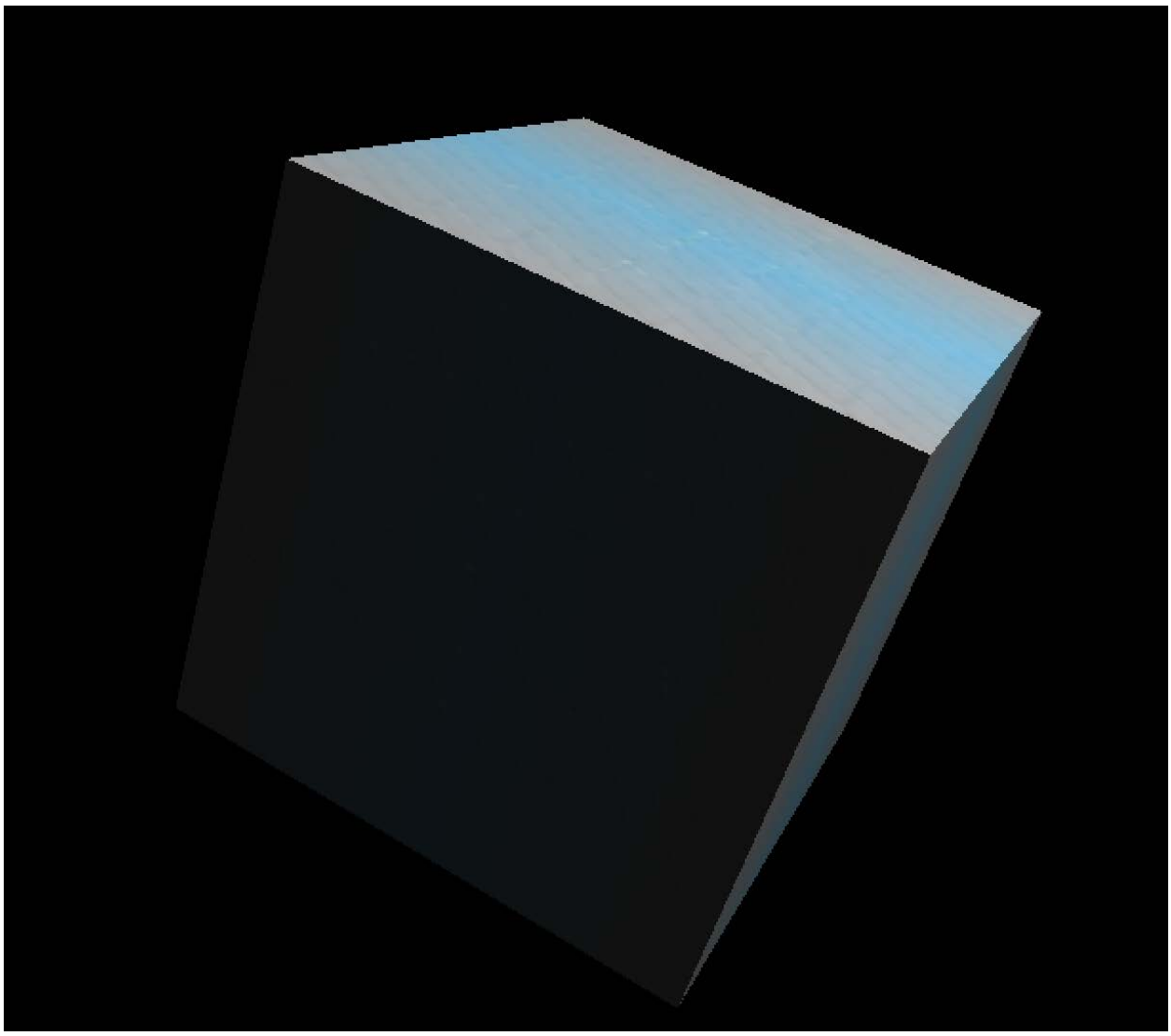
- Реализована базовая модель Фонга с добавлением фонового (ambient) освещения.

3. *Переключение между режимами*:
   - Событие нажатия клавиши (пробела) изменяет режим освещения.
   - Визуальная разница видна благодаря влиянию нормальных карт на детализацию.

4. *Отображение куба*:
   - Куб вращается вокруг произвольной оси для демонстрации эффектов освещения.

1. Пример работы





2. Выводы

Программа демонстрирует основы работы с освещением и шейдерами в OpenGL. Использование нормальных карт позволяет улучшить визуальную детализацию поверхности без увеличения количества полигонов. Возможность переключения между стандартным затенением и нормальной картой наглядно показывает преимущества метода.

Данная реализация предоставляет базовую основу для изучения работы с шейдерами и освещением. В дальнейшем можно расширить функционал, добавив поддержку других типов источников света или более сложные эффекты освещения, такие как параллакс-маппинг.

Московский Авиационный Институт
(Национальный Исследовательский Университет)
Институт № 8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №5 по курсу
«Компьютерная графика»

Студент: Ажаурова А.К.
Группа: М8О-303Б-22
Преподаватель: Филиппов Г.С.
Оценка: _____
Дата: _____
Подпись: _____

Москва, 2024

Содержание

1. Цель 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.

2. Постановка задачи

Требования

Реализуйте алгоритм трассировки лучей для отрисовки простой сцены, используя минимальный набор примитивов (сферы, плоскости и т.д.).

Реализуйте базовые эффекты: отражения, тени и освещение.

Трассировка должна быть реализована как на CPU, так и с возможной оптимизацией на GPU (опционально).

Программа должна корректно отображать сцены в зависимости от выбранного задания.

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.

3. Общие сведения о программе

Программа реализует алгоритм трассировки лучей (Ray Tracing) для визуализации 3D-сцены, содержащей два текстурированных объекта (стена и пол) и одну сферу с эффектом отражения. Основная цель программы — создание реалистичной сцены с текстурированием и контролем зеркальности сферы. Программа написана на языке C++ с использованием библиотеки OpenCV для визуализации и OpenMP для оптимизации вычислений. Реализовано базовое освещение, отражения и обработка текстур. Пользователь может управлять параметрами сцены, включая позицию камеры и зеркальность сферы, с помощью клавиатуры.

Программа обеспечивает следующие функции:

- Отрисовка текстурированных плоскостей (пол и стена).
- Создание реалистичных отражений на сфере, учитывающих текстуры.
- Интерактивное управление положением камеры и изменением зеркальности сферы.
- Сохранение результата визуализации в файл изображения.

4. Общий алгоритм решения

1. *Инициализация сцены:*
   - Загрузка текстур для пола и стены.
   - Создание примитивов: плоскостей (пол и стена) и сферы.
   - Настройка материалов объектов, включая зеркальность сферы.

2. *Установка камеры*:
   - Определение позиции камеры и направления её взгляда.
   - Реализация преобразования координат экрана в пространственные направления лучей.

3. *Трассировка лучей*:
   - Генерация лучей для каждого пикселя изображения.
   - Проверка пересечения лучей с объектами сцены.
   - Вычисление цвета пикселя на основе текстур, отражений и освещения.

4. *Рендеринг изображения*:
   - Параллельная обработка пикселей с использованием OpenMP.
   - Преобразование вычисленного цвета в диапазон [0, 255] для отображения.

5. *Интерактивное управление*:
   - Обработка ввода с клавиатуры для изменения положения камеры и зеркальности сферы.
   - Сохранение изображения при нажатии соответствующей клавиши.

6. *Вывод результатов*:
   - Отображение отрендеренной сцены в реальном времени.
   - Сохранение изображения в файл.

5. Реализация

1. *Инициализация и загрузка текстур:*
   - Использование cv::imread для загрузки текстур стены и пола.
   - Проверка успешности загрузки с выводом сообщений об ошибках.

2. *Создание объектов сцены:*
   - Реализация классов Plane и Sphere, которые поддерживают пересечение с лучами.

Учет текстур при расчете цвета плоскостей.

3. *Трассировка лучей:*
   - Реализация функции trace, которая обрабатывает пересечения лучей с объектами сцены.
   - Вычисление отраженных лучей для создания реалистичных отражений на сфере.

4. *Рендеринг и отображение:*
   - Генерация изображения с использованием трассировки лучей.
   - Применение OpenMP для ускорения вычислений.

5. *Интерактивность:*
   - Управление камерой и зеркальностью сферы через ввод с клавиатуры.
   - Сохранение изображения в файл.

6. *Очистка памяти:*
   - Удаление созданных объектов для предотвращения утечек памяти.

1. Пример работы

2. Выводы

В ходе лабораторной работы была реализована программа, демонстрирующая базовые принципы трассировки лучей. Основные задачи, такие как создание текстурированных объектов, расчёт отражений и освещения, были успешно выполнены. Реализация позволяет интерактивно управлять параметрами сцены, что делает её удобной для экспериментов и изучения.

Достигнуты следующие результаты:

- Сцена с текстурированными объектами и сферой отображается корректно.
- Реализованы отражения, включая отражение текстур.
- Пользователь имеет возможность изменять параметры зеркальности сферы, что позволяет наблюдать влияние отражений на визуализацию.
- Программа предоставляет высокую гибкость и интерактивность, а также демонстрирует ключевые аспекты физически корректного рендеринга.

Работа показала, что алгоритм трассировки лучей является мощным инструментом для создания реалистичной графики, но требует значительных вычислительных ресурсов. Возможными направлениями улучшения являются оптимизация алгоритма и использование GPU для ускорения вычислений.