Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **«**Информатика и системы управления**»**

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент Шавиш Тарек

Группа ИУ7и-41Б

Тип практики Технологическая

Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7

Студент _____ Шавиш Т.

подпись, дата

Руководитель практики _____ Волкова Л.Л

подпись, дата

Оценка _____

Москва 2024г.

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ИУ7
_____ Рудаков И.В.
« ___ » _____ 20 __ г.

ЗАДАНИЕ
на прохождение производственной/учебной практики

_____
Тип практики

Студент Шавиш Тарек 2 курса группы ИУ7и-41Б
в период с 11. 07.2024г. по 08.09.2024г.

Предприятие: «МГТУ им. Н. Э. Баумана»

Подразделение: _____
(отдел/сектор/цех)

Руководитель практики от предприятия (наставник): Волкова Л.Л

Руководитель практики от кафедры: Куров Андрей Владимирович, кандидат технических наук, доцент

Задание:

Дата выдачи задания « 10» июля 2024 г.

Руководитель практики от предприятия _____/ Волкова Л.Л
Руководитель практики от кафедры _____/ Куров А.В.

Студент _____/ Шавиш Т.

Оглавление

1. Введение

В последние годы трёхмерная графика и физическое моделирование становятся неотъемлемой частью многих приложений, начиная от видеоигр и заканчивая инженерными симуляциями и обучающими программами. Точность и реалистичность взаимодействия объектов в виртуальном пространстве играют ключевую роль в создании иммерсивных и интерактивных сред, что требует использования различных методов рендеринга и физической симуляции.

Данный проект направлен на разработку программы для моделирования захвата и перемещения кубов в трёхмерном пространстве. Основной целью проекта является создание системы, которая позволяет точно симулировать процесс взаимодействия объектов (кубов) с помощью методов физической симуляции, включая захват, перемещение, столкновения и взаимодействие с окружающей средой. Важными аспектами проекта являются реализация корректного управления камерой, что даёт пользователю возможность контролировать ракурс и точку зрения, а также применение моделей освещения и рендеринга для создания реалистичной визуализации сцены.

В ходе работы над проектом были изучены и применены такие технологии, как C++, OpenGL, GLFW и библиотеки для работы с математикой и линейной алгеброй. Для достижения реалистичности сцены использовались методы Z-буферизации для отсечения невидимых поверхностей, а также алгоритмы освещения, включая модель освещения Фонга. Всё это позволило создать эффективную систему симуляции с высокой степенью интерактивности и визуальной достоверности.

В отчёте представлены аналитическая часть, описывающая подходы к моделированию, конструкторская часть, детализирующая архитектуру программы, а также техническая часть, описывающая использованные библиотеки и методы реализации. В заключении рассматриваются результаты работы и перспективы дальнейшего развития проекта.

2. Аналитическая часть

2.1 Моделирование захвата и перемещения кубов

Захват и перемещение кубов в трёхмерном пространстве требует комплексного подхода, который включает физическое моделирование, обработку столкновений и управление взаимодействиями объектов. В данном проекте используется симуляция, основанная на методах физического моделирования и инверсной кинематики для захвата и перемещения объектов. Также важно правильно представлять геометрию кубов и обновлять их положение при взаимодействии с внешними силами.

2.1.1 Физическое моделирование захвата кубов

Физическое моделирование заключается в том, чтобы точно имитировать поведение твёрдых тел, таких как кубы, в трёхмерном пространстве. Для этого необходимо учитывать такие параметры, как:

- **Масса объекта**: масса кубов определяет, насколько легко их можно будет захватить и перемещать. Чем больше масса, тем больше требуется сила для их перемещения.
- **Силы, действующие на объект**: на кубы могут воздействовать силы гравитации, трения, столкновения с другими объектами и сила захвата виртуальной рукой или манипулятором.

Захват кубов моделируется через добавление виртуальной руки, которая перемещается в пространстве и захватывает куб с помощью алгоритмов **инверсной кинематики (IK)**. Алгоритм инверсной кинематики позволяет вычислить точные углы поворотов всех сегментов руки для того, чтобы достичь целевой позиции (например, центра куба). Использование инверсной кинематики требует расчёта следующих параметров:

- Положение и ориентация конечного эффектора (захватывающего механизма).
- Углы поворота каждого звена манипулятора для достижения цели.

Основное преимущество использования инверсной кинематики заключается в том, что она позволяет точно управлять "рукой", что особенно важно при сложных взаимодействиях с кубами в динамических сценах.

2.1.2 Моделирование перемещения кубов

После захвата куба виртуальной рукой необходимо корректно смоделировать его перемещение. Для этого используется физический движок, который управляет изменениями положения куба в зависимости от прикладываемых к нему сил. Перемещение куба в пространстве описывается уравнениями движения Ньютона:

$$F = m \times a$$

где F — сила, действующая на куб, m — его масса, а a — ускорение. На основании этого уравнения вычисляется новая скорость и положение куба:

$$V_{new} = V_{current} + a \times \Delta_t$$

$$P_{new} = P_{current} + V_{new} \times \Delta_t$$

где V_{new} *и* P_{new} *— новые значения скорости и положения куба, Δt — временной шаг симуляции. Важно учитывать взаимодействие кубов друг с другом и с поверхностью сцены. Для этого используются алгоритмы* ***обнаружения столкновений*** *(collision detection), которые определяют, когда и где кубы сталкиваются. После столкновения применяются силы реакции, и положение кубов обновляется.*

Примером такого взаимодействия является, когда один куб толкает другой: после столкновения движение первого куба замедляется, а второй начинает двигаться под действием силы удара.

2.2 Моделирование сетки и её деформация

Для корректного отображения кубов в трёхмерной среде важно правильно представлять их геометрию и учитывать изменения этой геометрии при движении и взаимодействии объектов. В компьютерной графике объекты, такие как кубы, представляются с помощью ***полигональных сеток****. Каждая сетка состоит из множества вершин (точек), соединённых рёбрами, которые формируют грани объекта.*

2.2.1 Представление кубов с помощью полигональных сеток

Каждый куб в сцене моделируется как полигональная сетка, которая состоит из шести граней, каждая из которых является квадратом, состоящим из двух треугольников (так как в компьютерной графике треугольники являются основными примитивами для рендеринга). Таким образом, куб состоит из 12 треугольников, соединённых 8 вершинами и 12 рёбрами. Каждая вершина задаётся вектором координат в трёхмерном пространстве (x,y,z).

Во время движения куба необходимо обновлять положение каждой вершины, чтобы правильно отобразить его новое положение в пространстве. Это достигается путём применения матриц трансформации, которые изменяют координаты вершин в зависимости от применяемой к объекту силы. Основная формула для изменения положения вершин:

$$V_{new} = T \times V_{current}$$

где V_{new} *— новое положение вершины,* T *— матрица трансформации (которая может включать в себя трансляцию, вращение и масштабирование), а* $V_{current}$ *— текущее положение вершины. Важно учитывать, что трансформации могут происходить одновременно по всем трём осям x, y, и z, что позволяет достигать сложных движений кубов в трёхмерной сцене.*

2.2.2 Деформация сетки при перемещении объектов

При динамическом перемещении объектов, таких как кубы, сетка объекта может деформироваться в зависимости от прикладываемых к нему сил и взаимодействий с другими телами. В данном проекте моделирование деформации кубов имеет важное значение, так как во время перемещения кубы могут столкнуться друг с другом или с окружающей средой. Эти взаимодействия вызывают временные изменения формы кубов, которые необходимо корректно отображать.

Для моделирования деформации сетки используются **алгоритмы изменения вершин сетки**. В отличие от статических объектов, для динамически изменяющихся тел, таких как кубы, необходимо каждый раз пересчитывать положение вершин при изменении их положения в пространстве. Деформация сетки моделируется изменением координат

вершин куба с течением времени, при этом учитываются применяемые силы, такие как сжатие, растяжение и ударные взаимодействия.

Основная формула обновления координат вершин при деформации выглядит следующим образом:

$$V_{new} = V_{current} + \Delta V * t$$

где:

- V_{new} — новое положение вершины,
- $V_{current}$— текущее положение вершины,
- ΔV — изменение скорости,
- t — прошедшее время.

Деформация сетки особенно актуальна в случае столкновений, когда необходимо корректно моделировать реакцию объекта на удар. При столкновении куба с другим объектом его сетка может временно деформироваться в точке касания, что создаёт более реалистичный эффект взаимодействия твёрдых тел. После того, как силы взаимодействия перестают действовать, сетка куба возвращается к своей первоначальной форме, если объект обладает достаточной жёсткостью (в случае несжимаемых тел).

Пример: При столкновении двух кубов одна из граней может временно изменять свою форму, вытягиваясь или сжимаясь в зависимости от силы удара. После этого грань возвращается в исходное состояние благодаря симуляции физических свойств куба.

Кроме того, на деформацию может влиять масштабирование объектов, если требуется изменять размеры кубов при выполнении определённых операций (например, при уменьшении или увеличении куба перед захватом). Это также отражается на сетке объекта, которая пересчитывается с учётом новых размеров и пропорций куба.

Таким образом, для корректного отображения взаимодействий кубов с окружающей средой и друг с другом необходимо учитывать изменения в их сетке и корректно обновлять положения вершин в процессе симуляции. Применение матриц трансформации позволяет достигнуть точной деформации объектов, что делает сцену более реалистичной и точной.

2.3 Тени, освещение и рендеринг

Визуализация сцены в трёхмерной графике требует не только правильного представления объектов, но и применения освещения, затенения и различных методов рендеринга для создания реалистичной картинки. Важную роль в этом играют алгоритмы освещения и тени, которые помогают создавать иллюзию объёмных объектов и взаимодействия их с окружающей средой.

2.3.1 Модели освещения: Фонг и Гуро

Для симуляции освещения в трёхмерных сценах используются различные модели, каждая из которых по-своему рассчитывает, как свет падает на объекты и как он отражается от их поверхности. В данном проекте используются две основные модели освещения — **модель Фонга** и **модель Гуро**.

Модель освещения Фонга:

- Эта модель вычисляет освещение для каждого пикселя на поверхности объекта. Основное её преимущество заключается в том, что она даёт более реалистичные блики и отражения света, поскольку учитывает три компонента освещения:
    1. **Окружающий свет (ambient)** — общий свет, равномерно освещающий сцену.
    2. **Диффузное освещение (diffuse)** — свет, который падает на объект и рассеивается в разные стороны, создавая мягкое освещение.
    3. **Зеркальное отражение (specular)** — яркие блики, которые появляются при отражении света от гладких поверхностей.

Основная формула освещения Фонга выглядит следующим образом:

$$I = I_{ambient} + I_{diffuse} + I_{specular}$$

где:

- I — итоговое освещение пикселя,
- $I_{ambient}$— интенсивность окружающего света,
- $I_{diffuse}$— интенсивность диффузного света,
- $I_{specular}$ — интенсивность зеркального отражения.

Этот метод особенно хорошо подходит для симуляций с бликами и гладкими объектами, такими как кубы, когда важна точность отображения света.

***Модель освещения Гуро**:*

- В отличие от Фонга, модель Гуро вычисляет освещение не для каждого пикселя, а только для вершин объекта, после чего освещение интерполируется между вершинами для всех остальных пикселей. Это значительно снижает вычислительные затраты, но может привести к менее точным результатам в случае бликов или резких изменений освещения.

$$I_{vertex} = I_{ambient} + I_{diffuse} + I_{specular}$$

где освещение вычисляется для каждой вершины, после чего происходит интерполяция по поверхности объекта.

Сравнение моделей:

- **Фонг**: Высокая точность бликов и отражений, но требует больших вычислительных ресурсов.
- **Гуро**: Быстрая модель, подходящая для более простых сцен, но менее точная в отображении бликов.

Таким образом, модель Фонга предпочтительна для создания более реалистичных сцен с детализированным освещением, в то время как модель Гуро может использоваться для ускорения рендеринга в менее критичных сценариях.

2.3.2 Алгоритм Z-буфера и создание теней

Одним из важнейших аспектов рендеринга в 3D-графике является правильное отображение глубины объектов, особенно в тех случаях, когда один объект закрывает другой. Для решения этой задачи используется **алгоритм Z-буфера**, который позволяет отсекать невидимые поверхности, обеспечивая корректное отображение объектов в кадре.

Z-буферизация — это метод, при котором для каждого пикселя на экране хранится его глубина (значение Z). Когда новый объект рендерится в той же точке экрана, его Z-значение сравнивается с тем, что уже записано в Z-буфер. Если новый объект ближе к камере (меньшее Z-значение), его пиксель отображается на экране, а если он дальше, то игнорируется.

$$if\ Z_{new} < Z_{buffer}[x][y] : Z_{buffer}[x][y] = Z_{\square}(x, y)$$

где:

- Z_{new} — глубина нового объекта в точке `(x, y)`,
- $Z_{buffer}[x][y]$ — текущее значение глубины в буфере для этого пикселя.

Этот алгоритм позволяет эффективно решать проблему пересечения объектов в трёхмерной сцене, обеспечивая корректное отображение объектов, находящихся ближе к камере.

***Создание теней**: Тени в трёхмерной графике играют важную роль в создании ощущения глубины и реалистичности сцены. В проекте используется метод рендеринга теней, основанный на **теневых картах** (shadow maps). Этот метод предполагает, что перед основным рендерингом сцены происходит расчёт теней с позиции источника света.*

1. Сначала сцена рендерится с точки зрения источника света, и для каждого пикселя сохраняется его глубина в теневой карте.
2. Затем, при рендеринге с точки зрения камеры, проверяется, находится ли каждый пиксель объекта в тени, сравнивая его координаты с сохранёнными в теневой карте.

Если глубина пикселя в теневой карте меньше, чем в текущем кадре, значит объект находится в тени и его освещение соответственно уменьшается.

Основная формула для проверки тени:

$$if\ Z_{camera} > Z_{\square} : pixel_{\square} = True$$

$$else : pixel_{\square} = False$$

Таким образом, тени и Z-буферизация вместе создают реалистичную трёхмерную сцену, обеспечивая правильное отображение объектов и их взаимодействие со светом.

2.4 Управление камерой: Эйлеровы и Лагранжевы перспективы

Правильное управление камерой в трёхмерной среде является важным аспектом для создания интерактивных и реалистичных симуляций. В зависимости от задачи могут использоваться различные подходы для описания движения камеры и её ориентации. В данном проекте применяются две основные перспективы — **Эйлерова** и **Лагранжева**, каждая из которых имеет свои особенности и области применения.

2.4.1 Эйлерова перспектива (Eulerian Perspective)

Эйлерова перспектива является одним из самых распространённых методов управления камерой в 3D-графике, особенно в видеоиграх и интерактивных приложениях. В этом подходе камера фиксируется в глобальной системе координат и может изменять своё положение и ориентацию относительно осей координат. Основное управление камерой происходит через изменение трёх углов: **Yaw** (рыскание), **Pitch** (тангаж) и **Roll** (крен).

1. Основные углы управления камерой:

- **Yaw (рыскание)**: это поворот камеры вокруг вертикальной оси Y, который позволяет перемещать взгляд камеры влево и вправо.
- **Pitch (тангаж)**: поворот камеры вокруг горизонтальной оси X, который контролирует движение взгляда вверх и вниз.
- **Roll (крен)**: поворот камеры вокруг оси Z, который наклоняет камеру по часовой или против часовой

Формула преобразования камеры: Для расчёта нового направления взгляда камеры после изменения углов используется векторное представление и тригонометрия. Камера направлена на точку в пространстве, и вектор направления взгляда Front обновляется по следующим формулам:

$$Front.x = \cos(Yaw) * \cos(Pitch)$$
$$Front.y = \sin(Pitch)$$
$$Front.z = \sin(Yaw) * \cos(Pitch)$$

После вычисления нового вектора направления камера может двигаться в любом направлении с учётом углов поворота.

2. Проблема замка кардана (Gimbal Lock): Эйлерова перспектива, несмотря на свою интуитивность, может сталкиваться с проблемой, называемой **замком кардана**. Это происходит, когда оси поворота камеры совпадают, что приводит к потере одной степени свободы. Это означает, что камера не может поворачиваться в одном из направлений. Чтобы избежать этой проблемы, часто используют **кватернионы**, которые позволяют корректно вращать объекты в трёхмерном пространстве, минуя замок кардана. Однако для целей данного проекта использование кватернионов не требуется, так как движения камеры достаточно просты.

2.4.2 Лагранжева перспектива (Lagrangian Perspective)

В отличие от Эйлеровой перспективы, Лагранжева перспектива основана на движении камеры, которое связано не с её собственным положением в глобальной системе координат, а с отслеживанием определённого объекта или точки в сцене. Этот подход особенно полезен в симуляциях, где необходимо наблюдать за движением объекта или группы объектов, например, кубов, которые перемещаются в пространстве.

1. Следование за объектом: В Лагранжевой перспективе камера "привязывается" к объекту и следует за ним по мере его движения. Это особенно удобно в ситуациях, когда важно видеть объект с разных ракурсов по мере его перемещения, например, при симуляции захвата кубов. В этом случае камера может быть запрограммирована так, чтобы оставаться на определённой дистанции от объекта и отслеживать его передвижения в реальном времени.

Основной принцип Лагранжевой перспективы заключается в том, что положение камеры обновляется относительно позиции отслеживаемого объекта. Формула обновления позиции камеры в таком случае:

$$Camera_{position} = Object_{position} + Offset$$

где:

- $Camera_{position}$ — новое положение камеры,
- $Object_{position}$ — текущее положение отслеживаемого объекта,
- $Offset$ — вектор смещения камеры относительно объекта (например, на определённой высоте и расстоянии от объекта).

Таким образом, камера остаётся на фиксированной дистанции от объекта и следит за его движениями. Этот метод позволяет более точно отслеживать объекты в динамических сценах.

2. Отслеживание сложных траекторий: Лагранжев метод особенно полезен для отслеживания объектов, которые движутся по сложным траекториям, например, при моделировании падения или перемещения кубов под действием силы тяжести или столкновений. Камера может плавно перемещаться вокруг объекта, сохраняя фокус на нём, что создаёт более естественное ощущение наблюдения.

2.4.3 Сравнение Эйлеровой и Лагранжевой перспектив

Эйлерова перспектива является более традиционным и интуитивным способом управления камерой, особенно в интерактивных приложениях, таких как видеоигры или симуляции, где важно гибко управлять направлением взгляда. Однако проблема замка кардана требует дополнительных вычислений или использования кватернионов, что усложняет реализацию.

Лагранжева перспектива, с другой стороны, более удобна для задач, связанных с отслеживанием объектов в динамических симуляциях. Этот подход позволяет легко удерживать объект в центре кадра и следить за его движениями по сложной траектории. Однако он менее интуитивен для пользователя и требует дополнительных настроек для управления камерой в глобальном пространстве.

3. Конструкторская часть

В данной части отчёта описывается архитектура программы, используемая для моделирования захвата и перемещения кубов в трёхмерном пространстве. Программа состоит из нескольких модулей, каждый из которых отвечает за выполнение определённых задач, таких как симуляция движения, рендеринг объектов и управление камерой. Общая архитектура программы была разработана с учётом возможности расширения и добавления новых функций.

3.1 Общая архитектура программы

Программа разделена на несколько модулей, каждый из которых выполняет свою задачу в процессе моделирования и рендеринга трёхмерной сцены. Основные модули программы:

3.1.1 Модуль ObjectManipulator

ObjectManipulator — это основной модуль, отвечающий за захват и перемещение кубов в трёхмерной сцене. Этот модуль реализует взаимодействие объектов с пользователем (или виртуальной рукой) и поддерживает выполнение следующих функций:

1. **Захват кубов**: Модуль обрабатывает команды, связанные с захватом объектов. Для этого используется алгоритм инверсной кинематики, который позволяет виртуальной руке захватывать куб в заданной точке. Примером может быть вычисление конечного положения руки для захвата объекта с использованием вектора цели.

2. **Перемещение кубов**: После захвата куб перемещается в пространстве в соответствии с направлением и силой, прикладываемой виртуальной рукой. Модуль рассчитывает новую позицию куба на основании законов физики (например, уравнения движения Ньютона), что обеспечивает плавное и реалистичное перемещение.

3.1.2 Модуль RigidBodySimulator

RigidBodySimulator отвечает за физическое моделирование взаимодействий твёрдых тел, таких как столкновения кубов друг с другом или с поверхностью сцены. Основная задача этого модуля — корректно рассчитывать силы, действующие на объекты, и обновлять их положение на основании этих взаимодействий.

Функции модуля:

- **Обнаружение столкновений (collision detection)**: При перемещении кубов RigidBodySimulator отслеживает, когда кубы соприкасаются друг с другом или с другими объектами в сцене.
- **Расчёт силы столкновения**: Модуль рассчитывает силу реакции при столкновении и обновляет скорости и положения кубов в зависимости от силы удара. Для этого используются уравнения законов сохранения импульса и энергии.

3.2 Модуль CameraController

CameraController управляет положением и ориентацией камеры в трёхмерной сцене. Этот модуль отвечает за изменения углов камеры (Yaw, Pitch, Roll) или её перемещение за объектами в Лагранжевой перспективе. CameraController использует матрицы трансформаций для обновления положения камеры в зависимости от действий пользователя или движений объектов в сцене.

3.2.1 Управление камерой в Эйлеровой перспективе

Для управления камерой в Эйлеровой перспективе используются углы рыскания (Yaw), тангажа (Pitch) и крена (Roll). CameraController обновляет эти углы на основании пользовательского ввода или предопределённых сценариев в зависимости от направления движения объектов.

3.2.2 Управление камерой в Лагранжевой перспективе

В Лагранжевой перспективе камера "привязана" к объекту и перемещается вместе с ним. CameraController обеспечивает корректное обновление позиции камеры, которая остаётся на заданной дистанции от объекта и следует за ним в процессе его движения.

3.3 Модуль Renderer

Renderer — это модуль, который отвечает за визуализацию всех объектов в трёхмерной сцене. Он рендерит кубы, освещение и тени, обеспечивая правильную отрисовку каждого объекта в зависимости от его положения, освещения и взаимодействия с другими объектами. Важными аспектами рендеринга являются правильное отображение глубины (использование Z-буфера) и применение освещения по модели Фонга для достижения реалистичной картинки.

3.3.1 Рендеринг кубов с использованием Z-буфера

Для корректной отрисовки кубов, когда один объект закрывает другой, используется **алгоритм Z-буфера**. Этот алгоритм позволяет отсекать невидимые поверхности, обеспечивая правильную глубину каждого объекта на экране. Модуль Renderer обновляет Z-буфер для каждого пикселя, сравнивая глубину нового объекта с уже отрисованными объектами.

3.3.2 Освещение и тени

Для создания реалистичной сцены необходимо применять различные модели освещения. В данном проекте используется **модель освещения Фонга**, которая учитывает как диффузное, так и зеркальное освещение для создания бликов на поверхностях кубов. Renderer вычисляет освещение для каждого объекта в зависимости от его положения относительно источника света.

3.4 Взаимодействие модулей программы

Все модули программы тесно связаны между собой, и каждый из них играет свою роль в общей симуляции и визуализации сцены. Взаимодействие между модулями происходит следующим образом:

1. **ObjectManipulator** управляет захватом и перемещением кубов, взаимодействуя с **RigidBodySimulator**, который обрабатывает физические столкновения и корректирует положение объектов в зависимости от их взаимодействий.
2. **CameraController** контролирует положение камеры, передавая информацию в **Renderer** для правильного рендеринга сцены с учётом положения камеры и объектов.
3. **Renderer** использует данные от всех модулей для создания финальной визуализации сцены. Он обрабатывает рендеринг кубов, освещение, тени и обеспечивает корректное отображение глубины с помощью Z-буфера.

4. Техническая часть

В данной части отчёта рассматриваются технические аспекты реализации программы, включая используемые языки программирования, библиотеки, а также взаимодействие между модулями. Основное внимание уделяется тому, как различные компоненты системы работают вместе для достижения цели моделирования и визуализации захвата и перемещения кубов в трёхмерном пространстве.

4.1 Используемые язык программирования и библиотеки

Программа написана на языке **C++**, который был выбран за его высокую производительность и возможности низкоуровневого программирования. C++ является одним из наиболее распространённых языков для разработки графических приложений и игр, что делает его идеальным выбором для этой симуляции.

4.1.1 Библиотеки

В проекте используются следующие ключевые библиотеки:

1. **OpenGL**: Основная библиотека для рендеринга 2D и 3D графики. OpenGL обеспечивает высокую производительность при работе с графикой и поддерживает аппаратное ускорение. Она используется для отрисовки объектов в сцене, применения освещения и управления тенями.
2. **GLFW**: Библиотека для управления окнами и обработки пользовательского ввода. GLFW позволяет легко создавать окна, управлять контекстами OpenGL и обрабатывать события клавиатуры и мыши, что делает её полезной для интерактивных приложений.
3. **GLM (OpenGL Mathematics)**: Библиотека для работы с векторами и матрицами, разработанная специально для использования с OpenGL. GLM обеспечивает удобные функции для выполнения математических операций, таких как преобразования, вращения и вычисления нормалей.
4. **Eigen**: Библиотека для линейной алгебры, используемая в проекте для решения задач, связанных с векторами и матрицами. Eigen поддерживает различные алгоритмы линейной алгебры и является высокопроизводительной и гибкой библиотекой.

4.1.2 Структура проекта

Структура проекта организована таким образом, чтобы модули могли легко взаимодействовать друг с другом. Основные каталоги проекта включают:

- `src/1`: Исходные файлы программы, включающие реализацию всех модулей (ObjectManipulator, RigidBodySimulator, CameraController и Renderer).
- `include/`: Заголовочные файлы, содержащие определения классов и функций.
- `lib/`: Библиотеки, используемые в проекте, включая OpenGL, GLFW и другие.
- `assets/`: Ресурсы, используемые в проекте, такие как текстуры, модели и шейдеры.

4.2 Взаимодействие модулей программы

Каждый модуль программы взаимодействует друг с другом через чётко определённые интерфейсы и функции. Взаимодействие между модулями организовано следующим образом:

4.2.1 Обмен данными

- **ObjectManipulator** отправляет данные о текущем состоянии кубов в **RigidBodySimulator**, который обрабатывает физику взаимодействий и возвращает обновлённые позиции и скорости объектов. Это позволяет обеспечить корректную симуляцию захвата и перемещения кубов.
- **CameraController** получает информацию о текущем состоянии объектов и их перемещениях от **ObjectManipulator**, чтобы обновить положение камеры в зависимости от движений захваченного куба или других объектов.
- **Renderer** получает данные о текущих позициях и состояниях кубов от **RigidBodySimulator** и **CameraController**. Он использует эти данные для рендеринга сцены, применения освещения и создания теней.

4.3 Использование Z-буфера для отсечения невидимых поверхностей

Z-буфер используется для отсечения невидимых поверхностей, обеспечивая правильный порядок отрисовки объектов в сцене. Каждый пиксель на экране хранит значение глубины, которое обновляется во время рендеринга.

4.3.1 Алгоритм работы Z-буфера

1. При рендеринге объекта для каждого пикселя рассчитывается его глубина (Z-значение) относительно камеры.
2. Сравнивается это значение с текущим значением в Z-буфере:
    - Если новое Z-значение меньше (объект ближе), обновляется Z-буфер и пиксель отображается.
    - Если новое Z-значение больше, объект игнорируется и не отображается.

4.4 Структура кода

Структура кода организована для обеспечения чёткого разделения между модулями, что улучшает читаемость и поддерживаемость проекта. Каждый модуль включает в себя следующие элементы:

- **Классы**: Каждому модулю соответствует свой класс, который инкапсулирует все функции и данные, связанные с его работой. Это позволяет организовать код и разделить ответственность.
- **Методы**: Каждый класс содержит методы, которые реализуют функциональность модуля. Методы имеют понятные названия и выполняют чётко определённые задачи.
- **Комментарии**: В коде используются комментарии для пояснения логики работы функций и классов, что облегчает понимание кода другим разработчикам.

1. Функция для захвата куба (ObjectManipulator)

```
void ObjectManipulator::grabCube(Cube& cube, const glm::vec3& handPosition) {
    // Проверяем, находится ли рука в пределах захвата куба
    if (glm::distance(cube.position, handPosition) < grabThreshold) {
        // Устанавливаем куб в состояние захваченного
        cube.isGrabbed = true;
        cube.position = handPosition; // Перемещаем куб к позиции руки
    }
}
```

2. Функция для обновления положения куба (RigidBodySimulator)

```
 void RigidBodySimulator::updateCubePosition(Cube& cube, float deltaTime) {
    // Применяем силы к кубу
    cube.velocity += cube.force / cube.mass * deltaTime; // Обновление скорости
    cube.position += cube.velocity * deltaTime; // Обновление позиции

    // Сбрасываем силы после применения
    cube.force = glm::vec3(0.0f);
 }
```

3. Функция для обновления положения камеры (CameraController)

```
void CameraController::updateCameraPosition(float deltaTime) {
   // Обновляем углы камеры на основе пользовательского ввода
   cameraPosition += cameraSpeed * glm::normalize(cameraFront) * deltaTime;

   // Обновляем матрицу вида
   viewMatrix = glm::lookAt(cameraPosition, cameraPosition + cameraFront, upVector);
}
```

4. Функция для рендеринга объекта с освещением (Renderer)

```
void Renderer::renderCube(const Cube& cube, const LightSource& light, const Camera&
camera) {
   applyPhongLighting(cube, light, camera); // Применение освещения
   // Код для рендеринга куба с OpenGL
   glBindVertexArray(cubeVAO);
   glDrawElements(GL_TRIANGLES, cube.indices.size(), GL_UNSIGNED_INT, 0);
}
```

Заключение

В ходе выполнения данного проекта была создана программа для моделирования захвата и перемещения кубов в трёхмерном пространстве. Основная цель заключалась в том, чтобы реализовать систему, которая использует методы физической симуляции для точного взаимодействия объектов и их корректного отображения в 3D-сцене с реалистичным освещением, тенями и управлением камерой.

Программа была разработана с использованием языка программирования C++ и таких библиотек, как OpenGL, GLFW, и GLM. OpenGL стал основным инструментом для рендеринга графики, обеспечивая высокую производительность и поддержку аппаратного ускорения, что позволило отрисовывать сцены в реальном времени. Библиотеки GLFW и GLM, в свою очередь, упростили работу с окнами, вводом пользователей и математическими операциями, необходимыми для преобразования координат в 3D-пространстве.

Особое внимание в проекте уделялось работе с камерой. Были реализованы два подхода к управлению камерой — Эйлерова и Лагранжева перспективы. Эйлерова перспектива позволила пользователю гибко управлять углами камеры (Yaw, Pitch и Roll), обеспечивая полный контроль над ориентацией камеры в сцене. В то же время Лагранжева перспектива использовалась для отслеживания объектов в процессе их перемещения, что создало более динамичное и интуитивное наблюдение за движениями кубов. Это добавило симуляции ощущение взаимодействия в реальном времени и сделало её более интерактивной.

В заключение, проект по моделированию захвата и перемещения кубов продемонстрировал важность интеграции различных технологий и методов для создания реалистичной симуляции. Успешная реализация проекта подтвердила эффективность выбранных решений и методологий, таких как инверсная кинематика, физическое моделирование твёрдых тел и рендеринг с использованием модели освещения Фонга. Программа обладает потенциалом для дальнейшего использования и может быть полезной для исследований, разработки игр или образовательных целей.

Список использованных источников

1. **Куров А. В.** Моделирование волн на поверхности жидкости. — МГТУ им. Н.Э. Баумана, 2013.
2. **Книги и статьи по 3D графике**:
    - "Computer Graphics: Principles and Practice" — Джон Д. Фредерик (John F. Hughes), Андрея 3. Кобера (Andries van Dam), Дэвид Ф. Прат (David F. Pritchard) и другие.
    - "Fundamentals of Computer Graphics" — Питер Ширли (Peter Shirley), Стивен Т. Туол (Steve Marschner) и другие.
3. CMake Build System Guide, https://cmake.org