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 ИУ «Информатика, искусственный интеллект и системы управления»

КАФЕДРА ИУ7 «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

Разработка программы для моделирования упругих столкновений объектов в пространстве.

Студент ИУ7-54Б _____ К. А. Рунов
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ А. А. Павельев
(Подпись, дата) (И. О. Фамилия)

2023 г.

Оглавление

Введение

На сегодняшний день компьютерная графика является неотъемлемой частью нашей жизни и используется повсеместно. Ей находится применение в самых разных областях человеческой деятельности: она используется в науке, в бизнесе, в кино, играх и везде, где нужно визуальное представление информации на электронном дисплее. [1]

Часто перед разработчиками графического программного обеспечения стоит задача синтеза реалистического изображения. Для решения этой задачи существует множество алгоритмов, но, как правило, алгоритмы, дающие наилучшие результаты являются наиболее трудозатратными как по объёму требуемой памяти, так и по количеству требуемых вычислений, потому что учитывают множество световых явлений (дифракция, интерференция, преломление, поглощение, множественное отражение). Поэтому, в зависимости от задачи и от имеющихся вычислительных мощностей, программистам приходится выбирать наиболее целесообразные в их случае алгоритмы и жертвовать либо временем генерации кадра, либо его реалистичностью.

Но генерация реалистического изображения – это лишь одна из многих задач, которые стоят перед программистами графических приложений. Часто требуется, чтобы разрабатываемое приложение было интерактивным – давало возможность как-то взаимодействовать с объектами, находящимися в виртуальном пространстве: перемещать их, сталкивать друг с другом, разрушать, деформировать.

Моей целью во время практики будет изучение и выбор алгоритмов для разработки программы моделирования упругих столкновений объектов. Исследуемые алгоритмы можно будет разделить на две категории: графические – относящиеся к изображению объектов на экране (удаление невидимых ребер и поверхностей, освещение) и физические (обнаружение столкновений объектов, реагирование на столкновение и оптимизация этих процессов).

1 Аналитический раздел

1.1 Представление объектов

Существует множество способов представления трехмерных объектов в пространстве. Наиболее популярные из них: каркасное, поверхностное и твердотельное.

Каркасная модель – объекты представляются в виде набора вершин и ребер (точек и линий). Самый простой способ и наименее требовательный к памяти из всех рассматриваемых.

Поверхностная модель – объекты представляются в виде набора поверхностей. Поверхность, в свою очередь, также можно задать несколькими способами, например, уравнением или набором плоскостей, аппроксимирующим исходную поверхность. Сложные объекты часто бывает сложно или даже невозможно описать в виде системы уравнений, а аппроксимировать, с другой стороны, возможно всегда. [9]

Твердотельная модель – существует два самых популярных метода представления твердотельных моделей:

1. Метод конструктивного представления (англ. Constructive representation, сокращённо C-rep);

2. Метод граничного представления (англ. Boundary representation, сокращённо B-rep).

Первый метод заключается в комбинировании базовых составляющих элементов, называемых твердотельными примитивами и осуществления между ними операций пересечения, объединения и разности для построения твердотельных моделей. Второй метод позволяет создавать точное представления твердого тела. Он хранит точное описание границ модели, всех поверхностей и вершин. Твердотельные модели наиболее требовательны к памяти, но позволяют точно описывать объекты, из-за чего повсеместно используются в инженерной промышленности. [10]

В разрабатываемой программе обнаружения коллизий нет надобности в точном описании объектов, и каркасной модели будет достаточно.

1.2 Формализация объектов синтезируемой сцены

- **Точечный источник света** – точка в пространстве, излучающая свет во всех направлениях. Интенсивность света убывает по мере удаления от источника. Точечный источник света характеризуется:
  (a) Положением в пространстве (трехмерные координаты)
  (b) Цветом (RGB)
  (c) Интенсивностью **I** (действительное число от 0 до 1)
  (d) Направлением (когда источник света расположен в бесконечности)
- **Объект сцены** – набор трехмерных примитивов, формирующих полигональную сетку. Объект сцены характеризуется:
  (a) Положением в пространстве (трехмерные координаты)
  (b) Ориентацией в пространстве (матрица модели)
  (c) Цветом (RGB)
- **Земля** – объект сцены, представляющий собой рельефную территорию.
- **Плоскость Земли** – горизонтальная плоскость (параллельная XY), проходящая через самую низкую точку Земли.
- **Камера** характеризуется:
  (a) Положением в пространстве (трехмерные координаты)
  (b) Направлением взгляда (трехмерный вектор)
  (c) Ориентацией в пространстве (трехмерный вектор, указывающий, в каком направлении у камеры верх)
  (d) Расстояниями до ближней и дальней граней пирамиды видимости
  (e) Углом обзора
  (f) Соотношением сторон

1.3 Анализ алгоритмов удаления невидимых линий и поверхностей

Алгоритмы удаления невидимых линий и поверхностей можно разделить на две группы. Одни работают в объектном пространстве – для каждого объекта проверяется, заслоняют ли его другие объекты или нет. Таким образом, потребуется $O(n^2)$ сравнений объектов. К таким алгоритмам относится, например, алгоритм Робертса. Другие работают в пространстве изображения – для каждого пикселя изображения определяется, какой объект сцены в нём виден. То есть, потребуется $O(Nn)$ сравнений, где N – количество пикселей. Таковы алгоритмы трассировки лучей, Варнока или z-буфера.

На первый взгляд может показаться, что, если мы редко имеем дело с более чем 1920×1080 объектами, то эффективнее будет использовать алгоритм, работающий в объектном пространстве, чтобы было меньше сравнений, однако это далеко не всегда так.

1.3.1 Алгоритм Робертса

Одно из требований алгоритма – тела должны быть выпуклыми. Поэтому, если какие-то из объектов сцены не выпуклые, то их следует разбить на выпуклые составляющие. На последующих этапах алгоритма удаляются:

1. Невидимые рёбра, экранируемые самим телом;
2. Невидимые рёбра, экранируемые другими телами сцены;
3. Новые невидимые рёбра, возникающие при «протыкании» тел друг другом.

К преимуществам данного алгоритма можно отнести его математическую точность и простоту идеи. [7, с. 250]

К недостаткам – следующие:

1. Трудности при работе с невыпуклыми объектами; их нужно разделять на выпуклые составляющие, что само по себе задача трудоёмкая.
2. Для каждого выпуклого тела составляется матрица размерности $4 \times N$, где N – количество граней тела. Если в программе используются модели

с большим количеством полигонов, эта матрица может быть очень большой, и выполнение операций над ней может значительно повлиять на время генерации кадра.

3. После разбиения объектов на выпуклые составляющие, количество объектов, для которых нужно выполнить проверку перекрытия возрастает, что увеличивает время выполнения алгоритма.

1.3.2 Алгоритм обратной трассировки лучей

Суть алгоритма для обработки скрытых или видимых поверхностей следующая: выпускаем луч через каждый пиксель перпендикулярно плоскости экрана и следим за его траекторией до тех пор, пока он не пересечёт поверхность видимого непрозрачного тела. Необходимо проверить пересечение каждого объекта сцены с каждым лучом, далее пересечения упорядочиваются по глубине. Пересечение с максимыльным значением z представляет видимую поверхность для данного пикселя. [7, с. 362]

Процедура поиска пересечений играет ключевую роль в алгоритме трассировки лучей, поскольку $75-95\%$ времени выполнения алгоритма приходится на неё. В связи с этим существуют многие оптимизации алгоритма, позволяющие не искать пересечения для объектов, которые луч точно не пересечёт. Например, заключение объекта или кластера объектов в сферическую или прямоугольную оболочки и проверка пересечения с оболочкой; если пересечения с оболочкой нет, значит луч не пересекает и объекты, в ней содержащиеся. [7, с. 363]

Преимуществом данного алгоритма является его независимость от характеристик когерентности той сцены, для которой ведётся поиск её видимых участков. Рассчёты для каждого пикселя производятся отдельно, независимо от других, следовательно, имеется возможность выполнять вычисления параллельно.

Ещё одним преимуществом алгоритма является возможность учёта прозрачности объектов при некоторой его модификации.

1.3.3 Алгоритм Варнока

Алгоритм основывается на рекурсивном разбиении экрана и работает тем быстрее, чем меньше на экране пересечений объектов. Подробнее преимущества и недостатки алгоритма будут рассмотрены в таблице.

1.3.4 Алгоритм, использующий z-буфер

Алгоритм работает следующим образом:

1. Создаётся z-буфер, размерность которого равна размерности буфера кадра, и инициализируется минимальными значениями z.

2. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесен в z-буфер. Если это сравнение показывает, что новый пиксел расположен впереди пиксела, находящегося в буфере кадра, то новый пиксеп заносится в этот буфер и производится корректировка z-буфера новым значением z. [7, с. 321]

Данный алгоритм может эффективно обрабатывать сложные сцены, но при этом требует большой объём памяти для хранения z-буфера и немалые вычислительные мощности – при его обновлении. Также, в связи с произвольным порядком сравнения объектов, возникают сложности с учётом их прозрачности.

1.3.5 Вывод

	Алгоритм Робертса	Алгоритм обратной трассировки лучей	Алгоритм Варнока	Алгоритм, использующий z-буфер
Рабочее пространство	Объектное	Изображения	Изображения	Изображения
Асимптотическая сложность алгоритма (n – количество граней объектов, N – количество пикселей)	$O(n^2)$	$O(Nn)$	$O(Nn)$	$O(Nn)$
Возможность выполнять вычисления параллельно для каждого пикселя	Нет	Есть	Нет	Есть
Возможность учёта прозрачности объектов	Нет	Есть	Нет	Есть (но сложно наладить корректную работу)
Эффективность не зависит от сложности сцены	Нет	Да	Нет	Да
Эффективность обработки сложных сцен	Низкая	Средняя	Низкая	Высокая
Сложность реализации	Низкая	Средняя	Средняя	Низкая

В программе обнаружения коллизий важную роль имеет скорость генерации кадра и менее важную – реалистичность изображения, которую можно было бы повысить, используя алгоритм обратной трассировки лучей. В связи с этим фактом и результатами, отображёнными в таблице выше, для удаления невидимых линий и поверхностей в программе был выбран алгоритм, использующий z-буфер.

1.4 Анализ алгоритмов обнаружения коллизий

1.4.1 Алгоритм AABB

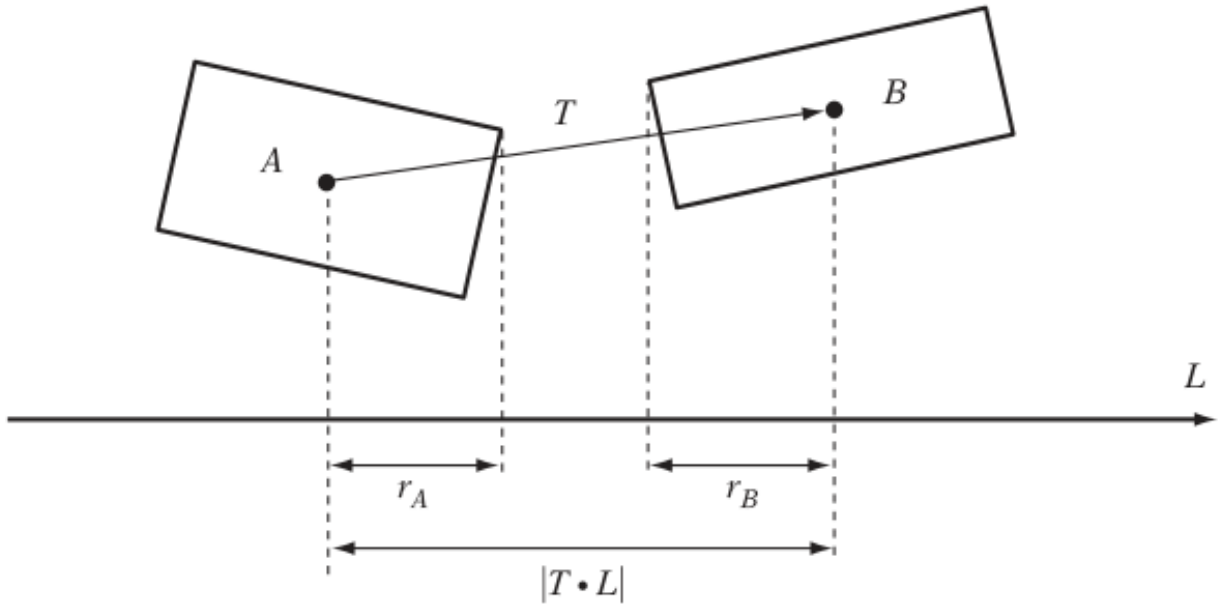
Алгоритм ограничивающего параллепипеда с гранями, выравненными по осям координат (англ. Axis-Aligned Bounding Box algorithm, сокращённо AABB) заключается в ограничении объектов параллелепипедами с гранями, выравнен-

ными по осям координат и последующем обранужении коллизий, используя теорему о разделяющей гиперплоскости [11]. То, что грани обоих AABB гарантированно параллельны общему набору осей координат, говорит о том, что разделяющая ось, если таковая существует, будет одной из этих осей координат [8, с. 827]. Обнаружение пересечений двух AABB-параллелепипедов очень просто в связи с параллельностью их граней координатным плоскостям: если проекции параллелепипедов на каждую координатную плоскость пересекаются, то пересекаются и сами параллелепипеды, иначе – пересечения нет.

1.4.2 Алгоритм OBB

В алгоритме ориентированного ограничивающего параллелепипеда (англ. Oriented Bounding Box algorithm, сокращённо OBB), в отличие от AABB, объекты ограничиваются параллелепипедом, грани которого не обязательно параллельны координатным плоскостям. Обнаружение пересечений двух OBB-параллелепипедов сложнее, чем для AABB-параллелепипедов, но в его основе также лежит теорема о разделяющей гиперплоскости [11], суть которой заключается в том, что проекции двух фигур на разделяющую плоскость не пересекаются, если не пересекаются сами фигуры. Проекции этих же фигур на неразделяющую ось могут пересекаться.

Рис. 1: Обнаружение пересечений двух OBB-прямоугольников. [4, с. 102]



Тест на обнаружение пересечений двух OBB-параллелепипедов следующий: Два OBB-параллелепипеда не пересекаются, если сумма проекций их радиусов на разделяющую ось меньше расстояния между проекциями их центров (Рис. 1).

1.4.3 Алгоритм GJK

Алгоритм Гильберта – Джонсона – Кирти (англ. Gilbert – Johnson – Keerthi algorithm, сокращённо GJK) основан на геометрической операции, известной как *Разность Минковского*. Она довольно проста: мы берём каждую точку внутри формы B и последовательно вычитаем её из каждой точки внутти формы A. Полученное множество точек $\{(\mathbf{A}_i - \mathbf{B}_j)\}$ и будет разностью Минковского. [8, с. 829]

У разности Минковского есть одно полезное свойство: если применить её к двум выпуклым формам, она будет *содержать начало координат* тогда и только тогда, когда эти две формы пересекаются. Можно интуитивно понять, почему оно является верным: под пересечением форм A и B на самом деле имеется в виду, что внутри A есть такие точки, которые находятся также внутри B. Вычитая каждую точку, составляющую B, из каждой точки, составляющей A, мы можем ожидать, что в какой-то момент нам попадется точка, общая для двух форм. В таком случае вычитание даст ноль, поэтому разность Минковского будет содержать начало координат тогда и только тогда, когда формы A и B имеют общие точки.

Разность Минковского для двух выпуклых форм сама является выпуклой формой. Но нас интересует лишь её оболочка, а не все её внутренние точки. Основной принцип работы GJK состоит в поиске тетраэдра, размещённого на оболочке разности Минковского и включающего в себя начало координат. Если такую фигуру можно найти, формы пересекаются, в противном случае – нет. [8, с. 830]

Подробнее с алгоритмом помогут ознакомиться следующие источники: [5], [6], [3], [2].

1.4.4 Вывод

	Алгоритм GJK	Алгоритм AABB	Алгоритм OBB
Асимптотическая сложность (n – количество вершин в полигональном представлении объекта)	$O(n^2)$ (худшее)	$O(1)$	$O(n^2)$ (худшее)
Точность обнаружения коллизий	Высокая	Низкая	Средняя
Эффективность на сложных формах	Высокая	Низкая	Средняя
Сложность реализации	Высокая	Низкая	Средняя

В программе для обнаружения коллизий важнейшими критериями при выборе алгоритма являются точность обнаружения коллизий и его эффективность на сложных формах. В связи с этим был выбран алгоритм GJK, несмотря на сложность его реализации.

2 Конструкторский раздел

2.1 Требования к программному обеспечению

Пользователь должен иметь следующие возможности:

- Создание объектов: куб, шар, чайник, прямоугольный параллелепипед.
- Изменение свойств объектов: размер, местоположение, цвет, масса, начальная скорость.
- Удобное управление камерой и возможность рассмотра сцены с разных сторон.
- Изменение значения гравитации.
- Запуск режима моделирования, в котором программа будет моделировать движение и столкновение объектов друг с другом.

Требования к программе:

- Программа должна генерировать кадр не более, чем за $\frac{1}{24}$ секунды.
- Программа должна учитывать освещение сцены при моделировании движения объектов.
- Никакие действия пользователя не должны приводить к аварийному завершению программы.

3 Технологический раздел

3.1 Выбор и обоснование языка программирования и среды разработки

В качестве языка программирования был выбран **C++** по следующим причинам:

- Имеется опыт разработки на данном языке
- **C++** – один из основных языков программирования, используемых для создания графических программ; огромное количество графического программного обеспечения в мире написано именно на языке **C++**
- Компилятор языка переводит текст программы в машинный код, который исполняется быстро
- Язык поддерживает объектно-ориентированную парадигму программирования, что позволит создавать удобные абстракции во время написания кода
- Существуют библиотеки (**glad**, **GLEW**), предоставляющие доступ к функциям **OpenGL**
- Наличие математических библиотек, использование которых ускорит процесс разработки (**glm**, **eigen**)
- Язык поддерживается отладчиком **gdb**
- Имеется возможность профилирования программы с помощью утилиты **gprof**
- Наличие графического отладчика **RenderDoc**
- Наличие кроссплатформенной утилиты для автоматической сборки программы **cmake**

В качестве среды разработки был выбран **neovim** по следующим причинам:

- Поддержка комбинаций клавиш текстового редактора **vim**, что позволит быстрее писать и редактировать написанный код
- Поддержка протокола **Language Server**, что позволит определять наличие ошибок времени компиляции в коде непосредственно во время написания кода, что ускорит процесс разработки
- Наличие множества плагинов ещё больше ускоряющих процесс разработки: **vim-fugitive** для использования системы контроля версий **git** не покидая среду разработки, **harpoon** и **telescope-nvim** для быстрой навигации между файлами проекта, **nvim-treesitter** для подсветки синтаксиса, **nvim-lspconfig** для использования протокола **Language Server**, **nvim-cmp**, **luasnip**, **friendly-snippets** и **cmp-luasnip** для автодополнения кода

Заключение

В ходе выполнения практики все поставленные цели были выполнены:

- Были исследованы алгоритмы для разработки программы моделирования упругих столкновений объектов в пространстве
- Был обоснован выбор алгоритмов, которые будут использованы при разработке программы
- Была начата разработка программы

Проделанная работа помогла получить мне практические навки сравнения алгоритмов, выделения критериев их сравнения, и дала базовые навыки проведения исследовательской работы.

Список используемой литературы

[1] Куров А. В. *Курс лекций по дисциплине «Компьютерная графика»*. 2023.

[2] *A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space*. Режим доступа: `https://graphics.stanford.edu/courses/cs448b-00-winter/papers/gilbert.pdf`. (Дата обращения: 16.07.2023).

[3] *Continuous Collision Detection: Progress and Challenges*. Режим доступа: `https://www.laas.fr/~nic/MOVIE/Workshop/Slides/Gino.vander.Bergen.ppt`. (Дата обращения: 16.07.2023).

[4] Christer Ericson. *Real-Time Collision Detection*. – Elsevier Inc., 2005 – 593 с. ISBN: 1-55860-732-3.

[5] *IMPLEMENTING GJK (2006)*. Режим доступа: `https://caseymuratori.com/blog_0003`. (Дата обращения: 16.07.2023).

[6] *The Gilbert-Johnson-Keerthi (GJK) Algorithm*. Режим доступа: `https://realtimecollisondetection.net/pubs/SIGGRAPH04_Ericson_the_GJK_algorithm.ppt`. (Дата обращения: 16.07.2023).

[7] Роджерс Д. *Алгоритмические основы машинной графики*. Пер. с англ. – М.: Мир, 1989 – 512 с. ISBN: 5-03-000476-9.

[8] Грегори Дж. *Игровой движок. Программирование и внутреннее устройство. Третье издание.* – СПб.: Питер, 2022 – 1136 с. ISBN: 978-5-4461-1134-3.

[9] *Методы 3D моделирования*. Режим доступа: `https://studbooks.net/1413342/bzhd/metody_modelirovaniya`. (Дата обращения: 15.07.2023).

[10] *Методы твердотельного моделирования*. Режим доступа: https://studref.com/605454/tehnika/metody_tverdotelnogo_modelirovaniya. (Дата обращения: 15.07.2023).

[11] *Теорема о разделяющей гиперплоскости*. Режим доступа: https://www.princeton.edu/~aaa/Public/Teaching/ORF523/S16/ORF523_S16_Lec5_gh.pdf. (Дата обращения: 16.07.2023).