Оглавление

Введение

ToDo: тильду перед всеми ссылками на формулы

В современном мире компьютерная графика используется достаточно широко. Типичная область ее применения – это кинематография и компьютерные игры.

На сегодняшний день большое внимание уделяется алгоритмам получения реалистичных и динамических изображений (анимации). Можно заметить, что чем качественнее мы хотим получить изображение на выходе алгоритма, тем больше времени и памяти потребуется для его синтеза. Это и становится проблемой при создании динамической сцены, так как на каждом временном интервале расчёт визуализации сцены необходимо производить повторно.

Однако не всегда необходимо, чтобы динамическое изображение было реалистичным. Для некоторых задач (например, для демонстрации работы некоторой системы) может быть достаточно обозначить все действующие объекты условно и отобразить эти условные обозначения в программном обеспечении (ПО).

Целью данной работы является разработка ПО с пользовательским интерфейсом, позволяющего получить динамическое изображение трёхмерной сцены с анимированной моделью робота Ф2.

Для достижения поставленной цели требуется решить следующие **задачи**.

1. Описать состав сцены.
2. Выбрать алгоритм удаления невидимых линий и поверхностей.
3. Выбрать модель освещения.

4. Выбрать способ анимации.

5. Разработать модель описания сцены и анимации.

6. Разработать алгоритм получения изображения.

7. Реализовать ПО с функциональностью получения динамического изображения трёхмерной сцены и провести его тестирование.

8. Выполнить анализ быстродействия программы.

1. Аналитическая часть

1.1 Описание объектов сцены

Сцена должна состоять из наблюдателя (камеры), плоскости пола, двух людей, источника направленного света и робота Ф-2. Модель робота Ф-2 должен состоять из нескольких частей тела: головы, туловища, рук. Голова робота должна быть представлена в виде параллелепипеда, а руки и туловище — в виде цилиндров. Модели людей также должны состоять из головы — сферы — а также туловища и рук — цилиндров. Плоскость пола — это некая ограничивающая плоскость. Предполагается, что под ней не расположено никаких объектов.

Анимация робота должна состоять из следующей последовательности действий.

1) Робот должен повернуть голову в сторону одного человека, затем в сторону другого.

2) Робот должен помахать руками первому человеку.

3) Робот должен приблизиться ко второму человеку и протянуть ему руку.

1.2 Анализ и выбор формы представления трехмерных моделей

Отображением формы и размеров объектов являются модели. Обычно используются следующие три способа описания моделей.

1. Каркасная (проволочная) модель – одна из простейших форм задания модели, в ней хранится информация только о вершинах и ребрах объекта. Недостаток данной формы состоит в том, что модель не всегда точно передает представление о форме объекта.

2. Поверхностная модель. Такой тип модели часто используется в компьютерной графике. Поверхности могут быть заданы разными способами: либо аналитически, либо с использованием полигональной аппроксимации. Недостаток данной формы состоит в том, что она не нест информации о том, с какой стороны расположен материал.

3. Объемная (твердотельная) модель. Данная форма отличается от поверхностной тем, что содержит информацию о материале. Это делается с помощью указания направления внутренней нормали.

Вывод: для решения данной задачи имеет смысл использовать поверхностные модели, так как каркасные модели могут привести к неправильному восприятию формы, а объемные модели повлекут излишние затраты памяти.

1.3 Анализ способа представления поверхностных моделей

Также необходимо определить каким образом будут заданы поверхностные модели. Существуют два основных способа:

Аналитический способ. Поверхность модели описывается функцией. Например, поверхность сферы имеет вид $x^2 + y^2 + z^2 = R^2$. Преимущества аналитического способа задания поверхностных моделей.

1) Минимальные затраты памяти.

2) Быстрое выполнение операций переноса, вращения, масштабирования.

3) Поверхность задаётся с абсолютной точностью, а не аппроксимируется.

Недостатки аналитического способа задания поверхностных моделей.

1) Задать сложную модель аналитически может быть практически невоз-

можно.

2) Скорость вычислений зависит от сложности подобранной функции.

3) Проблемы при наложении текстур.

Аппроксимация полигональной сеткой. Поверхность модели описывается набором многоугольников (как правило, используются треугольники)

Преимущества подхода на основе аппроксимации полигональной сеткой:

1) Можно достаточно точно передать поверхность любого объекта реального мира.

2) Преобразование всех моделей выполняется одинаково.

3) Добавление новой модели не требует написания кода.

4) Большинство алгоритмов удаления невидимых линий используют данное представление.

5) Информации, содержащейся в данном представлении достаточно для необычных преобразований модели (например, для разделения модели на несколько).

Недостатки подхода на основе аппроксимации полигональной сеткой.

1) Большие затраты памяти.

2) Скорость вычислений зависит от количества вершин у модели.

3) Модель может быть передана некачественно.

Вывод: в поставленной задаче нет необходимости в сложных поверхностях, так что наиболее подходящим булет аналитический способ представления моделей. Ключевыми для принятия этого решения оказались экономия памяти и быстрота преобразований в пространстве, достигаемые аналитическим представлением моделей. Самым ярким примером будет сравнение сложности представления сферы обоими способами.

1.4 Анализ и выбор алгоритма удаления невидимых линий и поверхностей

Перед выбором алгоритма удаления невидимых рёбер и поверхностей необходимо отметить, что для эффективного решения поставленной задачи алгоритм должен удовлетворять следующим требованиям: быстродействие, малое потребление памяти и возможность использовать выбранный формат представления модели (аналитический).

1.4.1 Алгоритм, использующий Z-буфер

Суть данного алгоритма – это использование двух буферов: буфера кадра, в котором хранятся атрибуты каждого пикселя, и Z-буфера, в котором хранятся информация о координате Z для каждого пикселя.

Первоначально в z-буфере находятся минимально возможные значения Z, а в буфере кадра располагаются пиксели, описывающие фон. Каждый многоугольник преобразуется в растровую форму и записывается в буфер кадра.

В процессе подсчета глубины нового пикселя, он сравнивается с тем значением, которое уже лежит в z-буфере. Если новый пиксель расположен ближе к наблюдателю, чем предыдущий, то он заносится в буфер кадра и происходит корректировка z-буфера. Для решения задачи вычисления глубины Z каждый многоугольник описывается уравнением $ax + by + cz + d = 0$. При c=0 многоугольник для наблюдателя вырождается в линию.

Для некоторой сканирующей строки $y = const$, поэтому имеется возможность рекуррентно высчитывать z' для каждого $x' = x + dx$.

Преимущества алгоритма, использующего Z-буфер.

1) Простота реализации алгоритма.

2) Оценка трудоемкости линейна.

Недостатки алгоритма, использующего Z-буфер.

1) Большие затраты памяти.

2) Сложное добавление функционала освещения и прозрачности.

3) Требует придставления модели в виде полигональной сетки.

Вывод: Алгоритм не удовлетворяет ключевым требованиям.

1.4.2 Алгоритм художника

У вышеуказанного алгоритма “говорящее” название, потому что его работа схожа с рисованием картины художником. Как художники рисуют картину: предварительно наносятся на рисунок более далёкие объекты и только позднее наносятся близкие к наблюдателю объекты. При реализации данного алгоритма используют сортировку (по глубине). Цель сортировки в том, что все грани сортируются таким образом, чтобы отрисовка наиболее близких граней была произведена в конце, то есть сначала рисуются самые далёкие грани и только потом – ближние грани. Рассмотрим положительные стороны и недостатки данного алгоритма.

Преимущества алгоритма художника.

1. Линейно-логарифмическая оценка трудоемкости.

2. Малое использование памяти.

Недостатки алгоритма художника.

1. Сложность реализации пересекающихся граней.

2. Результирующая картина не обладает достаточной реалистичностью.

3. Многократное перекрашивание одного и того же пиксела.

4. Требует представления модели в виде полигональной сетки.

Вывод: Алгоритм не удовлетворят ключевым требованиям.

1.4.3 Алгоритм обратной трассировки лучей

Суть алгоритма обратной трассировки лучей (в оригинале - Ray Tracing) состоит в том, что наблюдатель видит объект с помощью испускаемого света, который согласно законам оптики доходит до наблюдателя некоторым путем. Отслеживать пути лучей от источника к наблюдателю неэффективно с точки

зрения вычислений, поэтому наилучшим способом будет отслеживание путей в обратном направлении, то есть от наблюдателя к объекту.

Преимущества алгоритма обратной трассировки лучей.

1. Высокая реалистичность синтезируемого изображения.

2. Работа с поверхностями в аналитической форме.

3. Могут быть применены параллельные вычисления.

Основной недостаток алгоритма обратной трассировки лучей — низкая производительность реализаций алгоритма.

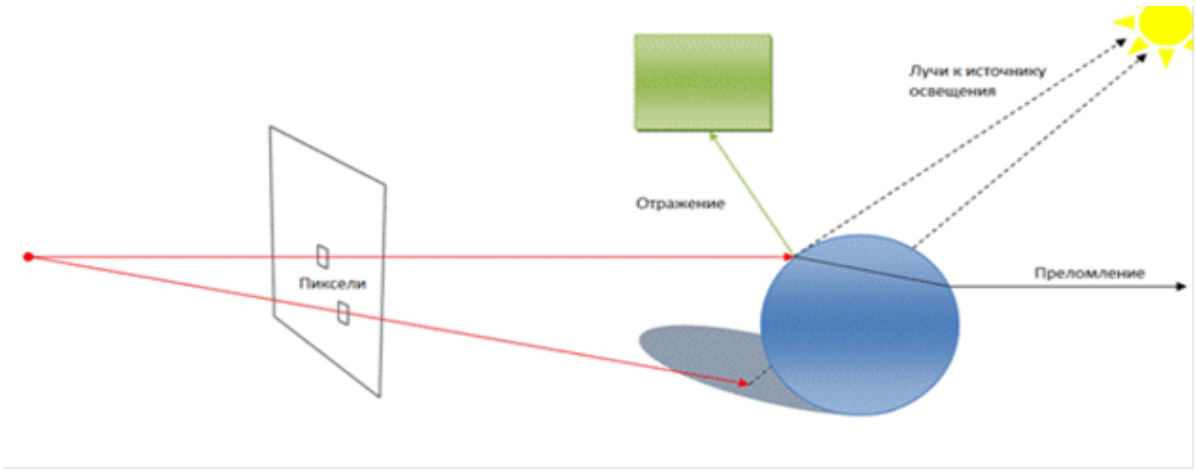

Рисунок 1.1. Иллюстрация обратной трассировки лучей

Вывод: данный алгоритм в первую очередь используется для получения реалистичного изображения, что сказывается на его быстродействии, однако алгоритм может быть более эффективным, если его адаптировать к решаемой задаче.

1.4.4 Алгоритм Робертса

Алгоритм Робертса имеет свои особенности, он работает в объектном пространстве, но при этом он рисует сцену только с выпуклыми телами – это необходимое условие для валидной отработки алгоритма.

Данный алгоритм имеет 3 этапа и один предварительный:

Предварительный этап - подготовка исходных данных. Необходимо для каждого тела сформировать матрицу тела размерностью 4xN, где N – количество граней тела.

1. Удаление рёбер, экранируемых самим телом Определим вектор E = (0, 0, -1, 0) как вектор взгляда наблюдателя. Чтобы понять, какие грани отрисовать нужно, а какие – не нужно, нужно умножить вектор взгляда на матрицу тела. Отрицательные величину результирующей матрицы свидетельствуют о невидимых гранях.

2. Удаление рёбер, экранируемых другими телами На этом шаге используется луч, который “испускается” от точки наблюдателя до точки на ребре. Если луч пересекает какую-либо грань, значит точка невидима.

3. Удаление линий пересечения тел, экранируемых самими телами и другими телами, связанными отношением протыкания.

Если тела связаны отношением взаимного протыкания, то будут образовываться новые рёбра, и также требуется определить их видимость. На данном этапе на рассматриваются рёбра, которые соединяют невидимые точки, так как в этом нет смысла. И в результате новые рёбра проверяются на экранирование другими телами сцены и самими телами.

Преимущества алгоритма Робертса.

1. Высокая точность вычислений.

2. Алгоритм используется в объектном пространстве.

Недостатки алгоритма Робертса.

1. Квадратичная сложность алгоритма, в зависимости от количества объектов.

2. Объекты сцены обязательно должны быть выпуклыми многоугогранниками.

3. Высокая сложность реализации алгоритма.

Вывод: Алгоритм не удовлетворят ключевым требованиям.

1.4.5 Алгоритм Варнока

Алгоритм Варнока является одним из примеров алгоритма, основанного на разбиении картинной плоскости на части, для каждой из которых исходная задача может быть решена достаточно просто.

Поскольку алгоритм Варнока нацелен на обработку картинки, он работает в пространстве изображения. В пространстве изображения рассматривается окно и решается вопрос о том, пусто ли оно, или его содержимое достаточно просто для визуализации. Если это не так, то окно разбивается на фрагменты до тех пор, пока содержимое фрагмента не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения.

Сравнивая область с проекциями всех граней, можно выделить случаи, ко-

гда изображение, получающееся в рассматриваемой области, определяется сразу:

1. Проекция ни одной грани не попадает в область.

2. Проекция только одной грани содержится в области или пересекает область. В этом случае проекции грани разбивают всю область на две части, одна из которых ответствует этой проекции.

3. Существует грань, проекция которой полностью накрывает данную область, и эта грань расположена к картинной плоскости ближе, чем все остальные грани, проекции которых пересекают данную область. В данном случае область соответствует этой грани.

Если ни один из рассмотренных трех случаев не имеет места, то снова разбиваем область на четыре равные части и проверяем выполнение этих условий для каждой из частей. Те части, для которых таким образом не удалось установить видимость, разбиваем снова и т. д.

Ключевое преимущество алгоритма Варнока — малые затраты по времени в случае области, содержащей мало информации.

Недостатки алгоритма Варнока:

1. Алгоритм работает только в пространстве изображений.

2. Большие затраты по времени в области с высоким информационным содержимым.

3. Требует придставления модели в виде полигональной сетки.

Вывод: алгоритм не удовлетворят ключевым требованиям.

1.4.6 Алгоритм маршрирования лучей

В алгоритме маршрирования лучей (в оригинале - Ray Marching) вся сцена определяется в терминах функции расстояния со знаком. Чтобы найти пересечение между лучом обзора и сценой, постепенно перемещается точка от камеры вдоль направления луча. На каждом шаге, при помощи функции расстояния определяется, вошла ли точка в какое-либо тело. [5]

В целях повышения производительности реализаций алгоритма, расстояние, на которое смещается точка не является фиксированно-маленьким, оно вычисляется каждую итерацию, как максимально допустимое, не приводящее к прохождению луча сквозь поверхность.

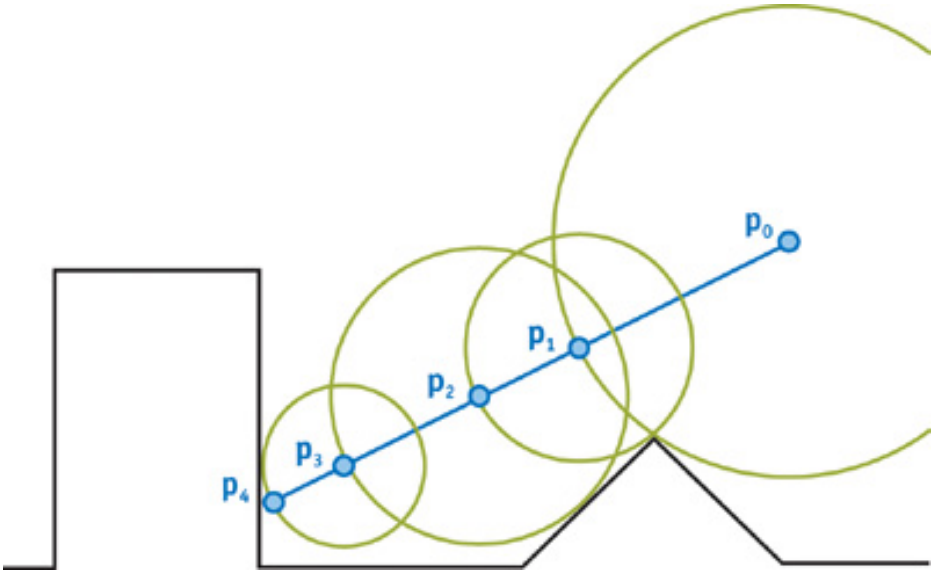

Рисунок 1.2.

Преимущества алгоритма маршрирования лучей:

1. Алгоритм использует модели в аналитической форме.

2. Не требователен по памяти.

3. Позволяет визуализировать необычные поверхности, например, трёхмерные фракталы.

4. Изображение получается идеально гладким.

5. Могут быть применены параллельные вычисления.

Недостатки алгоритма маршрирования лучей.

1. Алгоритм очень сильно теряет в производительности с ростом количества объектов на сцене.

2. Все недостатки аналитического представления поверхности.

Вывод: алгоритм может быть использован для решения задачи, однако его реализация будет неэффективной по времени.

Вывод из анализа алгоритмов удаления невидимых линий и поверхностей: из всех перечисленных алгоритмов только два могут быть приме-

нены к аналитически заданным поверхностям. Из этих алгоритмов выбор пал на обратную трассировку лучей, так как она окажется более быстродейственной, при оптимизации под нашу задачу.

1.5 Анализ и выбор модели освещения

Модель Ламберта

Модель Ламберта включает только идеальное диффузное освещение, то есть свет при попадании на поверхность рассеивается равномерно во все стороны. При такой модели освещения учитывается только ориентация поверхности (N) и направление источника света (L) (рис. 1.3):

$$I = I_p \cdot k_p + \frac{I_u \cdot k_d \cdot \cos\theta}{d + K} \tag{1.1}$$

где I_p — интенсивность рассеянного света;

k_p — коэффициент диффузного отражения рассеянного света

I_u — интенсивность точечного источника света

k_d — коэффициент диффузного отражения

θ — угол между нормалью к поверхности и направление света

d — расстояние от центра проекции до объекта

K — произвольная константа

ToDo: оформление списков, умножение в формулах, тире тройные, для диапазонов чисел — двойные: 1–2 И картинку 1.4 подписать и перенести, как 1.3, по аналогии

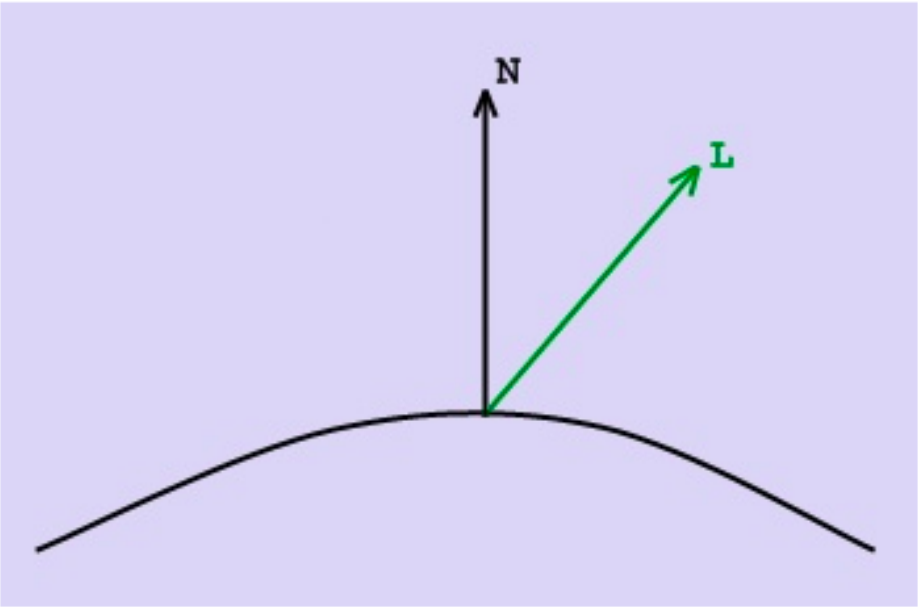

Рисунок 1.3.

Эта модель является одной из самых простых моделей освещения и очень часто используется в комбинации с другими моделями. Она может быть удобна для анализа свойств других моделей, за счет того, что ее легко выделить из любой модели и анализировать оставшиеся составляющие.

Модель Фонга

Модель Фонга (простая модель освещения) представляет собой комбинацию диффузной и зеркальной составляющих. Работает модель таким образом, что кроме равномерного освещения на материале могут появляться блики. Местонахождение блика на объекте определяется из закона равенства углов падения и отражения. Чем ближе наблюдатель к углам отражения, тем выше яркость соответствующей точки:

$$I = I_p \cdot k_p + \frac{I_u}{d + K} \cdot (k_d \cdot \cos(\theta) + k_з \cdot cos^n(\alpha)), \tag{1.2}$$

где I_p — интенсивность рассеяного света

k_p — коэффициент диффузного отражения рассеяного света

I_u — интенсивность точечного источника света

k_d — коэффициент диффузного отражения

θ — угол между нормалью к поверхности (n) и направлением света (L)

d — расстояние от центра проекции до объекта

K — произвольная константа

$k_з$ — коэффициент зеркального отражения

α — угол между вектором наблюдения (S) и отраженным лучом (R)

n — степень, аппроксимирующая пространственное распределение зеркально отраженного света

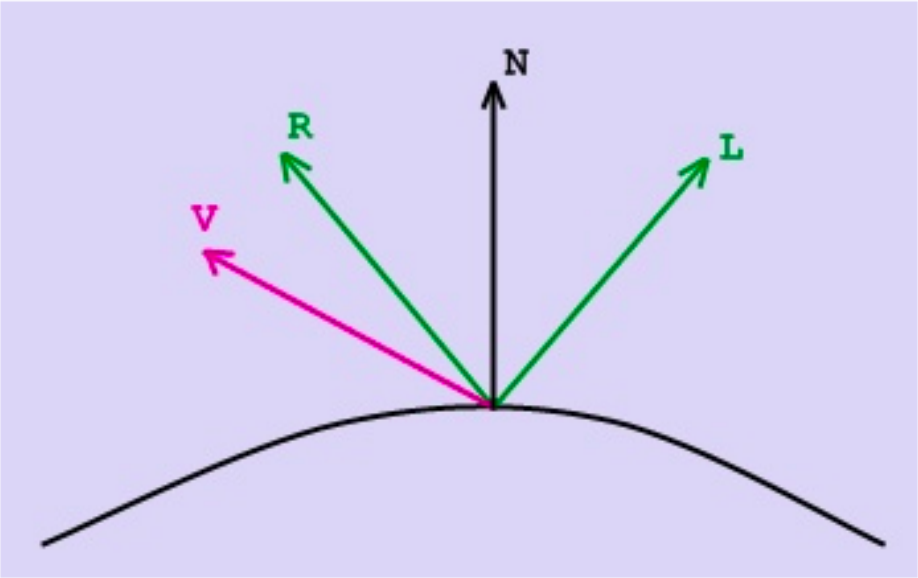

Рисунок 1.4.

Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения (рис. 1.4). Нормаль делит угол между лучами на две равные части. L — направление источника света, R — направление отраженного луча, V — направление на наблюдателя.

Глобальная модель освещения

Глобальная модель будет содержать в себе те же составляющие, что и простая модель освещения, но ещё учитывать интенсивность рассеянного освещения, зеркально отраженный свет от других поверхностей и пропускание света прозрачными поверхностями.

$$I = I_p \cdot k_p + k_d \cdot \sum_j I_{uj} \cdot \cos\theta_j + k_s \cdot \sum_j I_u \cdot cos^n \alpha_j + k_s \cdot I_s + k_t \cdot I_t \qquad (1.3)$$

где

I_p — интенсивность рассеяного света

k_p — коэффициент диффузного отражения рассеяного света

I_u — интенсивность точечного источника света

k_d — коэффициент диффузного отражения

θ — угол между нормалью (n) к поверхности и направлением света (v)

d — расстояние от центра проекции до объекта

K — произвольная константа

k_s — коэффициент зеркального отражения

α — угол между вектором наблюдения (S) и отраженным лучом (r)

n — степень, аппроксимирующая пространственное распределение зеркально отраженного света

k_t — коэффициент пропускания

I_t — интенсивность света, падающего в точку Q по направлению **p**

I_s — интенсивность зеркально отраженного света

Вывод: глобальная модель освещения является слишком затратной по времени, а модель Ламберта недостаточно хорошо передаёт геометрию сцены, поэтому была выбрана модель освещения Фонга.

1.6 Анализ и выбор способа анимирования робота Ф-2

Метод ключевых кадров

При данном подходе анимационная последовательность задается в терминах ключей. Каждый ключ представляет позицию и ориентацию, которые должен

занять заданный объект в определенный момент времени. Позиция и ориентация объекта на протяжении анимационной последовательности вычисляется на основе этих ключей [7].

Метод кривых движения

При данном подходе анимационная последовательность задается с помощью кривых движения. Кривые движения – это представление перемещения или трансформации объекта в виде графиков для каждой из его координат XYZ [7].

Вывод

Отличие методов анимирования по производительности несущественно, поэтому выбран метод ключевых кадров как более простой в реализации и использовании.

2. Конструкторская часть

2.1 Описание представления объектов сцены

Аналитический способ представления трехмерных объектов сцены означает, что для каждой формы придется использовать собственное уравнение (или систему уравнений)

Представление сферы

Поверхность сферы радиуса r, расположенной в начале координат, аналитически может быть представлена в виде уравнения

$$x^2 + y^2 + z^2 = r^2$$

Следовательно структура данных, описывающя сферу, должна хранить два поля:

1) position : (float, float, float) — смещение по (x, y, z);

2) r : float — радиус, r $>$ 0.

Представление цилиндра

Поверхность правильного цилиндра радиуса r и высоты $h = 2 \cdot l$, расположенного в начале координат, аналитически может быть представлен в виде системы:

$$\begin{cases} x^2 + y^2 = r^2, \text{ При -l <= z <= l : боковая поверхность} \\ x^2 + y^2 <= r^2, \text{ При } |z| = l : \text{основания цилиндра} \end{cases}$$

Также необходимо учесть, что цилиндр может быть расположен в пространстве под произвольным углом относитьльно осей координат, для чего вводится отдельное поле в структуре данных.

1. position : (float, float, float) - смещение по (x, y, z)
2. rotation : (float, float, float) - поворот относительно (Ox, Oy, Oz)
3. r : float, r $>$ 0 - радиус
4. l : float, l $>$ 0 - высота цилиндра

Представление параллелепипеда

Поверхность прямоугольного параллелепипеда со сторонами, параллельными координатным плоскостям может быть однозначно задана парой вершин параллелепипеда, примыкающих к его диагонали.

Тогда структура данных, описывающая параллелепипед должна иметь вид.

1. p1, p2 : (float, float, float) - вершины
2. rotation : (float, float, float) - поворот относительно (Ox, Oy, Oz)

2.2 Общий алгоритм решения поставленной задачи

Общий алгоритм решения поставленной задачи включает следующие этапы.

1. Задать объекты сцены.
2. Задать анимацию.

3. Каждый кадр анимации выполнить:

    3.1. Использовать ключевые кадры для вычисления позиции и вращения объектов.

    3.2. Применить алгоритм обратной трассировки лучей для синтеза изображения.

    3.3. Вывести изображение на экран.

2.3 Алгоритм обратной трассировки лучей с использованием модели освещения Фонга

Ниже описан алгоритм обратной трассировки лучей с использованием параллельных вычислений для модели освещения Фонга.

1. Разбить проецирующую плоскость на равные участки.

2. Выделить потоки для обработки участков.

3. Разбить участок на пиксели.

4. Для каждого пикселя:

    4.1. Выпустить первичный луч.

    4.2. Определить пересечения первичного луча со всеми объектами сцены.

    4.3. Если пересечений нет, покрасить пиксель в цвет фона.

    4.4. Иначе:

        4.4.1. Выбрать ближайшую к началу луча точку пересечения.

        4.4.2. Из точки пересечения выпустить теневые зонды.

        4.4.3. Учесть источники света, до которых дошли теневые зонды.

        4.4.4. Покрасить пиксель с соответствующей интенсивностью.

2.4 Алгоритмы пересечения луча с объектами

2.4.1 Описание алгоритма пересечения луча со сферой

Сфера аналитечески может быть представлена в виде

$x^2 + y^2 + z^2 = r^2$

Рассматривается луч, исходящий из точки $(x0, y0, z0)$ в направлении вектора (l, m, n), где $l^2 + m^2 + n^2 = 1$ (необходимо перевести в СК сферы).

Луч выражается параметрически

$x = lt + x0$

$y = mt + y0$

$z = nt + z0$

$t > 0$

В уравнении сферы проводится подставновка.

$$(lt + x0)^2 + (mt + y0)^2 + (nt + z0)^2 = r^2 \tag{2.1}$$

Получается квадратное уравнение, которое решается через дискриминант.

$$(l^2 + m^2 + n^2)t^2 + 2(lx0 + my0 + nz0)t + (x0^2 + y0^2 + z0^2 - r^2) = 0 \tag{2.2}$$

Если квадратное уравнение не имеет решения - луч не пересекает сферу.

Если решение есть, выбирается наименьший t, как t, характеризующий ближайшую к началу луча точку.

2.4.2 Описание алгоритма пересечения луча с цилиндром

Правильный цилиндр аналитечески может быть представлен в виде

$x^2 + y^2 = r$, $r > 0$, $-l <= z <= l$

Рассматривается луч, исходящий из точки $(x0, y0, z0)$ в направлении векто-

ра (l, m, n), где $l^2 + m^2 + n^2 = 1$ (необходимо перевести в СК цилиндра).

Луч выражается параметрически

$x = lt + x0$

$y = mt + y0$

$z = nt + z0$

$t > 0$

В первую очередь можно проверить луч на пересечение с основаниями цилиндра. Для этого z приравнивается к l и -l, вычисляется значение парамера $t = (z - z0)/n$.

По полученному t вычисляются x, y.

Если полученные точки удовлетворяют $x^2 + y^2 <= r$ То Луч пересекает основания цилиндра. В противном случае ищется пересечение луча с боковой поверхностью.

В уравнении цилиндра проводится подставновка.

$$(lt + x0)^2 + (mt + y0)^2 = r \tag{2.3}$$

Получается квадратное уравнение, которое решается через дискриминант.

$$(l^2 + m^2)t^2 + 2(lx0 + my0)t + (x0^2 + y0^2 - r) = 0 \tag{2.4}$$

Если квадратное уравнение не имеет решения - луч не пересекает боковую поверхность цилиндра.

Если решение есть, для полученных значений параметра t вычисляется значение z, которое проверяется на условие -l $<$ z $<$ l.

Если z удовлетворяет условию — пересечения найдены, выбирается ближайшая к началу луча точка.

2.4.3 Описание алгоритма пересечения луча с параллелепипедом

Прямоугольный параллелепипед со сторонами, параллельными координатным плоскостям, однозначно определяется любыми двумя своими вершинами, примыкающими к одной из диагоналей параллелепипеда.

Например, вершинами $(x1, y1, z1), (x2, y2, z2)$.

Рассмотрим луч, исходящий из точки $(x0, y0, z0)$ в направлении вектора (l, m, n), где $l^2 + m^2 + n^2 = 1$

Рассматривается пара плоскостей, параллельных плоскости $yz : x = x1$ и $x = x2$

При l=0 заданный луч параллелен этим плоскостям и, если $x0 < x1$ или $x0 > x2$, то он не пересекает рассматриваемый прямоугольный параллелепипед. Если же l не равно 0, то вычисляются отношения $t1x = (x\_x0)/l$, $t2x = (x + -x0)/l$.

Можно считать, что найденные величины связаны неравенством $t1x < t2x$.

Пусть $tn = t1x, tf = t2x$

Считая, что $m > 0$, и рассматривая вторую пару плоскостей, несущих грани заданного параллелепипеда, $y = y1$ и $y = y2$, можно найти вершины величины

$t1y = (y\_y0)/m, t2y = (y + -y0)/m$.

Если $t1y > tn$, тогда пусть $tn = t1y$.

Если $t2y < tf$, тогда пусть $tf = t2y$.

При $tn > tf$ или при $tf < 0$ заданный луч проходит мимо прямоугольного параллелепипеда.

Считая $n > 0$, рассматриваем последнюю пару плоскостей $z = z1$ и $z = z2$, можно найти величины $t1z = (z\_z0)/n$, $t2z = (z + -z0)/n$ затем повторяются предыдущие сравнения.

Если в итоге всех проведенных операций $0 < tn < tf$ или $0 < tf$, то заданный луч непременно пересечет исходный параллелепипед со сторонами, параллельными координатным осям.

Если луч протыкает прямоугольный параллелепипед (т.е. начальная точка лежит вне параллелепипеда), то расстояние от начала луча до точки его входа в параллелепипед равно tn, а до точки выхода луча из параллелепипеда tf.

2.5 Анимация методом ключевых кадров

Для анимации методом ключевых кадров необходимо ввести сущность — кадр. Сущность будет реализована классом KeyFrame. Кадр должен содержать номер кадра анимации, а также смещение и вращение объекта на этот кадр.

Дополнительно будет разработан класс Animation, для хранения списка ключевых кадров, относящихся к одному объекту.

Для отрисовки каждого кадра предварительно будут высчитываться позиция и вращение объекта, как значение между двумя соседними ключевыми кадрами, пропорциональное расстояниям между текущим кадром и ключевыми.

3. Технологическая часть

3.1 Выбор языка программирования

Существует множество языков программирования, многие из которых обладают достаточно высокой эффективностью. Выбор осуществлялся по двум критериям: наличию необходимой для выполнения задачи функциональности и наличию опыта разработки на данном языке.

Для разработки данной программы был выбран язык C++. Данный выбор обусловлен следующими факторами.

1) Этот язык обладает высокой эффективностью и гибкостью в работе с памятью []. ToDo: ссылка! Билли, нам нужна ссылка!

2) C++ позволяет применять технологию объектно-ориентированного программирования.

3) C++ предоставляет механизмы для работы с нативными потоками, что позволит распараллелить вычисления.

3.2 Выбор средств разработки

В качестве фреймворка выбран Qt, так как он содержит всю необходимую функциональность и имеет подробную документацию.

В качестве среды разработки была выбрана Visual Studio 2022 Preview. Некоторые факторы, по которым была выбрана данная среда.

1) Поддерживает разработку на C++ / Qt;

2) Включает такую функциональность, как отладчик, поддержка точек остановки, профилировщик;

3) Данная среда бесплатна для студентов.

3.3 Реализация алгоритмов

В листингах 3.1–3.5 представлена реализация алгоритмов нахождения пересечения луча с объектами сцены: сферой, цилиндром, параллелепипедом.

Листинг 3.1. Алгоритм нахождения пересечения луча со сферой

```
1   Hit3d Sphere::hit(Ray3d ray) const
2   {
3       Hit3d hit;
4       float x0 = ray.point.x - _transform.position.x;
5       float y0 = ray.point.y - _transform.position.y;
6       float z0 = ray.point.z - _transform.position.z;
7       float l = ray.direction.x;
8       float m = ray.direction.y;
9       float n = ray.direction.z;
10
11      float a = (l * l + m * m + n * n);
12      float k = (l * x0 + m * y0 + n * z0);
13      float c = (x0 * x0 + y0 * y0 + z0 * z0 - _radius * _radius);
14      float D1 = k * k - a * c;
15      if (D1 < 0)
16          hit.distance = -1.0;
17      else
18      {
19          float t1 = (-k - sqrt(D1)) / a;
20          float t2 = (-k + sqrt(D1)) / a;
21
22          float t = fmin(t1, t2);
23          if (t < 0)
24              t = fmax(t1, t2);
25
26          hit.distance = t;
27          hit.point = ray.point + ray.direction * t;
28          hit.normal = (hit.point -
                _transform.position).normalized();
29          hit.color = _color;
30      }
31
32      return hit;
33  }
```

Листинг 3.2. Алгоритм нахождения пересечения луча с цилиндром (начало)

```
1   Hit3d Cylinder::hit(Ray3d ray) const {
2       Hit3d hit;
3       hit.distance = -1.0f;
4       _transform.rayToLocal(ray);
5       float x0 = ray.point.x;
6       float y0 = ray.point.y;
7       float z0 = ray.point.z;
8       float l = ray.direction.x;
9       float m = ray.direction.y;
10      float n = ray.direction.z;
11      if (fabs(n) > EPS) {
12          float z = _height / 2;
13          float t = (z - z0) / n;
14          float x = l * t + x0;
15          float y = m * t + y0;
16          if (t > 0 && x * x + y * y <= _radius * _radius) {
17              hit.distance = t;
18              hit.point = Vector3d(x, y, z);
19              hit.normal = Vector3d::forward();
20          }
21          z = -z;
22          t = (z - z0) / n;
23          x = l * t + x0;
24          y = m * t + y0;
25          if (t > 0 && x * x + y * y <= _radius * _radius) {
26              if ((hit.distance > 0 && t < hit.distance) ||
                    hit.distance < 0) {
27                  hit.distance = t;
28                  hit.point = Vector3d(x, y, z);
29                  hit.normal = Vector3d::back();
30              }
31          }
32      }
33      //...
```

Листинг 3.3. Алгоритм нахождения пересечения луча с цилиндром (продолжение)

```
1         //...
2         float a = (l * l + m * m);
3         float k = (l * x0 + m * y0);
4         float c = (x0 * x0 + y0 * y0 - _radius * _radius);
5         float D1 = k * k - a * c;
6         if (D1 >= 0) {
7             float t1 = (-k - sqrt(D1)) / a;
8             float t2 = (-k + sqrt(D1)) / a;
9             float t = fmin(t1, t2);
10            if (t < 0)
11            t = fmax(t1, t2);
12            if (t > 0 && fabs(n * t + z0) < _height / 2) {
13                if ((hit.distance > 0 && t < hit.distance) ||
                      hit.distance < 0) {
14                    hit.distance = t;
15                    hit.point = ray.point + ray.direction * t;
16                    hit.normal = Vector3d(l * t + x0,
17                          m * t + y0, 0).normalized();
18                }
19            }
20        }
21        if (hit.distance > 0) {
22            _transform.hitToGlobal(hit);
23            hit.color = _color;
24        }
25
26        return hit;
27    }
```

Листинг 3.4. Алгоритм нахождения пересечения луча с параллелепипедом (начало)

```
1     Hit3d Box::hit(Ray3d ray) const {
2         Hit3d hit;
3         hit.distance = -1.0f;
4         _transform.rayToLocal(ray);
5         float p[3] = { ray.point.x, ray.point.y, ray.point.z };
6         float d[3] = { ray.direction.x, ray.direction.y,
              ray.direction.z };
7         float tn = std::numeric_limits<float>::min();
8         float tf = std::numeric_limits<float>::max();
9         for (int i = 0; i < 3; i++)
10            if (fabs(d[i]) < 1e-12) {
11                if (p[i] < _b1[i] || p[i] > _b2[i])
12                return hit;
13            }
14            else {
15                float t1 = (_b1[i] - p[i]) / d[i];
16                float t2 = (_b2[i] - p[i]) / d[i];
17                if (t1 > t2)
18                std::swap(t1, t2);
19                tn = fmax(tn, t1);
20                tf = fmin(tf, t2);
21                if (tn > tf || tf < 0)
22                    return hit;
23            }
24
25            //...
```

Листинг 3.5. Алгоритм нахождения пересечения луча с параллелепипедом (продолжение)

```
1            //...
2
3            if (0 < tn && tn < tf) {
4                hit.distance = tn;
5                hit.point = ray.point + ray.direction * tn;
6                if (fabs(hit.point.x - _b1[0]) < 1e-3) hit.normal =
                     Vector3d::left();
7                else if (fabs(hit.point.x - _b2[0]) < EPS) hit.normal
                     = Vector3d::right();
8                else if (fabs(hit.point.y - _b1[1]) < EPS) hit.normal
                     = Vector3d::down();
9                else if (fabs(hit.point.y - _b2[1]) < EPS) hit.normal
                     = Vector3d::up();
10               else if (fabs(hit.point.z - _b1[2]) < EPS) hit.normal
                     = Vector3d::back();
11               else if (fabs(hit.point.z - _b2[2]) < EPS) hit.normal
                     = Vector3d::forward();
12
13               _transform.hitToGlobal(hit);
14               hit.color = _color;
15           }
16
17           return hit;
18       }
```

3.4 Тестирование

В таблицах [ссылки] приведены тесты для функий перевода точки из глобальной системы координат в локальную и из глобальной в локальную. В таблицах [ссылки] приведены тесты для функций, нахождения пересечения луча с объектами сцены: сферой, цилиндром, параллелепипедом.

Таблица 3.1. Тестирование алгоритма перевода точки между глобальной и локальной системами координат

Глобальные координаты	Transform тела	Локальные координаты
1 1 1	0 0 0, 0 0 0	1 1 1

Таблица 3.2. Тестирование алгоритма нахождения пересечения луча со сферой

Начало луча	Направление луча	Transform тела	Радиус сферы	Точка пересечения
1 1 1	1 1 1	0 0 0, 0 0 0	10	1 1 1

Таблица 3.3. Тестирование алгоритма нахождения пересечения луча с цилиндром

Начало луча	Направление луча	Transform тела	Радиус цилиндра	Высота цилиндра	Точка пересечения
0 0 0	1 1 1	0 0 0, 0 0 0	50	200	1 1 1

Таблица 3.4. Тестирование алгоритма нахождения пересечения луча с параллелепипедом

Начало луча	Направление луча	Transform тела	Размер параллелепипеда	Точка пересечения
1 1 1	0 0 0	0 0 0, 0 0 0	100 100 100	1 1 1

3.5 Реализация многопоточности

ToDo: ссылки

Параллельные вычисления были реализованы с помощью стандартного класса std::thread [вставить ссылку]. В качестве примитивов синхронизации были использованы классы группы std::atomic [тоже ссылку].

Обработка пользовательского интерфейса была вынесена в отдельный поток, что позволило сохранить отзывчивость интерфейса во время вычислений.

3.6 Формат описания сцены

todo: поведение? в 1-ой главе описать сцену, объекты и их поведение (и что это такое)

Описание сцены и поведения её объектов может быть выполненно с помощью текстового файла, составленного по определенным правилам. Каждая строка файла может содержать одну команду и необходимые для неё аргументы. Команды могут добавлять на сцену объекты, менять их состояние, а также создавать ключевые карды для анимации объектов сцены. Доступен следующий список команд.

todo:

1. /camera <w> <h> <d> — добавляет на сцену камеру, с разрешением экрана w на h и отношением ширины к расстоянию до экранной плоскости d (w, h — целые числа, d — вещественное число).

2. /amb_light <i> — добавляет на сцену рассеянное освещение, интенсивностью i (i — вещественное число).

3. /dir_light <i> <x> <y> <z> — добавляет на сцену источник направленного по вектору (x, y, z) света, с интенсивностью i (i, x, y, z — вещественные числа).

4. /robot <name> <px> <py> <pz> <rx> <ry> <rz> — добавляет на сцену в точке (px, py, pz) модель робота, повёрнутую относительно осей координат на величины rx, ry, rz; <name> — строка — используется для указания этого робота в качестве цели для других команд (<px>, ..., <rz> — вещественные числа).

5. /human <name> <px> <py> <pz> <rx> <ry> <rz> — добавляет на сцену модель человека, аргументы используются аналогично команде /robot.

6. /paint <name> <r> <g> <b> — изменяет цвет объекта name на (r, g, b), где r, g, b — целые числа от 0 до 255.

7. /anim <name> — обозначает, что объект name будет анимирован.

8. /key <t> <name> [<px> <py> <pz> <rx> <ry> <rz>] — обозначает кадр номер t как ключевой в анимации объекта name. Если указаны параметры [px, ..., rz], то в данном кадре объект будет находиться в точке (px, py, pz) и будет повёрнут относительно осей координат на величины rx, ry, rz в градусах (t — неотрицательное целое число, <px>, ..., <rz> — вещественные числа). Если параметры не указаны, на кадре t объект будет представлен в том же состоянии, что было указано при его создании. Примечание: части тела робота и человека могут быть анимированны отдельно, для чего нужно использовать указания на части объекта: name.head, name.body, name.lhand и name.rhand. Смещение и вращение для частей тела указываются относительно их точки крепления к объекту, а не глобальной системы координат.

ToDo: поправить обозначения (перенести)в списке; ценры поворота ге-то обсудить – не факт, что в этой главе

В листинге 3.6 описана сцена, состоящая из поверхности пола, робота, человека и источника света. Робот анимирован: сначала он поворачивается в сторону человека, а затем машет ему руками.

Листинг 3.6. Пример файла описания сцены

```
1         /camera 800 600 0.5
2
3         /box gr 2500 50 2500 0 -130 200 0 0 0
4         /paint gr 0 200 0
5
6         /robot r1 0 0 200 0 180 0
7
8         /human h1 300 0 150 0 90 0
9
10        /amb_light 0.2
11        /dir_light 0.75 0 -0.2 1
12
13        /anim r1
14        /key 0 r1
15        /key 5 r1 0 0 200 0 135 0
16
17        /anim r1.lhand
18        /key 0 r1.lhand
19        /key 5 r1.lhand
20        /key 15 r1.lhand 0 0 0 120 45 0
21
22        /anim r1.rhand
23        /key 0 r1.rhand
24        /key 5 r1.rhand
25        /key 15 r1.rhand 0 0 0 120 -45 0
```

4. Исследовательский раздел

4.1 Измерение быстродействия

Важным критерием качества разработанного ПО является быстродействие, которое можно оценить, замерив количество генерируемых програмой кадров в секунду (FPS). Были проведены замеры времени отрисовки кадра для сцены, содержащей один источник света и разное число роботов. Результаты представлены в таблице 4.1.

Таблица 4.1. Результаты замеров времени (на моделях роботов)

Количество роботов	1	1	1	1
FPS	1	1	1	1

Время, затрачиваемое на отрисовку объекта, зависит от модели этого объекта. Для сравнения, в таблице 4.2 представлены результаты замеров времени для сцены, в которой все модели роботов были заменены на модели сфер.

Таблица 4.2. Результаты замеров времени (на моделях сфер)

Количество сфер	1	1	1	1
FPS	1	1	1	1

4.2 Результат работы программы

На рисунке [todo] можно видеть изображение сцены, получаемое в ходе работы программы.

Заключение

Цель поставленной работы достигнута. Разработано ПО генерирующее динамическое изображение трёхмерной сцены с анимированным роботом Ф2. Все поставленные задачи были выполнены.

1. Был описан состав сцены.
2. Выбран алгоритм удаления невидимых линий и поверхностей.
3. Выбрана модель освещения.
4. Выбран способ анимации.
5. Разработана модель описания сцены и анимации.
6. Разработан алгоритм получения изображения.
7. Реализовать и протестировать ПО с функциональностью получения динамического изображения трёхмерной сцены.
8. Выполнен анализ быстродействия программы.

Список источников

ToDo:

1. Дёмин А.Ю., Основы компьютерной графики: учебное пособие – Томск: Изд-во Томского политехнического университета, 2011. – 191 с.

2. Авдеева С.М., Куров А.В. Алгоритмы трехмерной машинной графики: учебное пособие. - М.: Издательство МГТУ им. Н.Э. Баумана, 1996. - 60 с., ил.

3. Шикин Е.В., Боресков А.В. Компьютерная графика. Динамика, реалистические изображения. – М.: Диалог-МИФИ, 1995. – 288 с.

4. Роджерс Д., Алгоритмические основы машинной графики: пер. с англ.— М.: Мир, 1989.— 512 с.: ил.

5. Hart, John C. (June 1995), "Sphere Tracing: A Geometric Method for the Antialiased Ray Tracing of Implicit Surfaces"

6. Brown, Margery Experimental Animation Techniques. Olympia, Washington: Evergreen State Collage (2003). Дата обращения: 11 ноября 2005. Архивировано 7 марта 2008 года.

7. Ken A. Priebe (2006), “The art of stop-motion animation”