# Содержание

# Введение

В настоящее время компьютерная графика имеет широкое применение в различных сферах. В частности, визуализация объектов с помощью электронно-вычислительных машин используется в киноиндустрии, разработке компьютерных игр и моделировании физических процессов.

Особое внимание уделяется визуализации деформируемых тел, которые могут менять свою форму, внутреннюю структуру, объем и площадь поверхности под действием внешних сил. Одним из таких объектов является детская игрушка «Лизун».

Целью данной курсовой работы является разработка программного обеспечения, моделирующего детскую игрушку «Лизун».

Задачи работы:

- выбрать модель тела и алгоритмы, необходимые для реализации программы моделирования детской игрушки «Лизун»;
- разработать соответствующее программное обеспечение;
- измерить среднее время работы реализации алгоритма удаления невидимых линий и поверхностей;
- закрепить знания и навыки, приобретенные в ходе изучения курса компьютерной графики.

# 1 Аналитическая часть

## 1.1 Описание объектов сцены

Сцена состоит из следующих объектов: камера, источник света, пол, имеющий структуру, и слайм.

Камера представляет собой невидимый объект, содержащий в себе информацию о координатах положения камеры в пространстве и векторе направления взгляда и использующийся для получения изображения на дисплее.

Источник света представляет собой материальную точку, которая испускает лучи света во все стороны. Данный объект хранит в себе информацию о координатах положения в пространстве источника света и цвете.

Пол представляет собой плоскость, заданную уравнением $z = 0$. Предполагается, что все объекты находятся над полом или на нем.

Слайм - это моделируемая игрушка «Лизун». В нашей программе слайм представляет собой вязкоупругое тело. При отсутствии внешних сил слайм имеет форму шара.

## 1.2 Выбор модели представления объекта

Для задания трехмерных моделей выделяют три формы: каркасную, поверхностную и объемную.

Каркасная модель является простейшим видом. В ней задается информация о вершинах и ребрах объекта. Однако, ввиду своей простоты, данный вид обладает серьезным недостатком: не всегда корректно передается представление об объекте.

Поверхностная модель предполагает хранение информации не только о вершинах и ребрах объекта, но и о его поверхности. Поверхность может быть описана как аналитически, так и с помощью задания участков поверхности как поверхностей того или иного рода. Недостатком данной модели является невозможность определения, с какой стороны поверхности находится

материал объекта.

Объемная модель отличается от поверхностной лишь тем, что в ней мы храним информацию о расположении материала объекта, указывая направление вектора внутренней нормали

Для данного проекта была выбрана поверхностная модель, так как каркасная модель не всегда правильно передает представление об объекте, а объемная модель является избыточной в рамках решаемой задачи.

## 1.3 Выбор модели представления поверхности объекта

Поверхность объекта может быть описана как аналитически, так и с помощью полигональной сетки. В аналитическом методе поверхность объекта рассматривается как множество точек, координаты которых удовлетворяют заданному уравнению, в то время как полигональная сетка представляет собой совокупность вершин, ребер и полигонов, соединенных таким образом, что каждое ребро принадлежит не более, чем двум многоугольникам [1].

Существует несколько способов описания полигональных сеток:

1 список вершин;

2 список ребер;

3 список граней.

Первый способ подразумевает хранение информации о сетке в виде перечня координат вершин, в котором каждая вершина записывается один раз в виде тройки координат. При этом каждый многоугольник задается множеством порядковых номеров вершин в списке.

Во втором способе базовую информацию дает список вершин, но многоугольники описываются ссылками на список ребер, в котором каждое из них упоминается только один раз. В свою очередь, ребра представляются в виде пары граничных вершин и перечня смежных многоугольников, число которых не превышает двух.

В третьем способе хранятся список вершин и список граней. В каждой грани хранится информация о трех вершинах.

Для данного проекта была выбрана модель полигональной сетки, ввиду простоты реализации и отсутствия серьезных проблем при описании сложных объектов. Кроме того, хранение информации о сетке будет осуществляться с помощью списка граней, так как данный способ предоставляет полную информацию о гранях, которая может быть крайне полезна при реализации алгоритмов удаления невидимых ребер и поверхностей.

## 1.4 Выбор метода физического моделирования объекта

Для моделирования процессов, протекающих при деформации слайма, необходимо выбрать соответствующий метод описания физических свойств тела.

Существует три метода моделирования деформируемых тел [2]:

1 метод конечных элементов;

2 метод граничных элементов;

3 метод с использованием модели масс с пружинами.

### 1.4.1 Метод конечных элементов

В данном методе объект разбивается на конечные элементы, соединяющиеся между собой в узлах. Каждый конечный элемент должен быть достаточно простым, чтобы имелась возможность легко определить напряжения в любой его части по заданным перемещениям узлов. По точкам составляется система уравнений, которая решается относительно перемещений рассматриваемых точек. Данная модель имеет высокую точность, однако требует знания характеристик материала объекта, необходимых для рассчета соотношений для каждого элемента [2].

### 1.4.2 Метод граничных элементов

Метод граничных элементов является интересной альтернативой метода конечных элементов, так как все преобразования осуществляются над точками поверхности объекта, что дает прирост к скорости вычислений. Основная сложность реализации данного метода заключается в преобразовании интегральной формы уравнения движения тела в поверхностный интеграл [2].

### 1.4.3 Модель масс с пружинами

Система масс с пружинами подразумевает хранение объекта, как совокупности точечных масс, соединенных между собой сетью невесомых пружин [2].

Для описания динамики движения слайма была выбрана модель масс с пружинами, так как она позволяет выполнять простые вычисления и дает приемлемую точность моделирования.

## 1.5 Выбор алгоритма удаления невидимых ребер и поверхностей

Рассмотрим следующие алгоритмы удаления невидимых ребер и поверхностей:

- алгоритм Робертса;
- алгоритм Варнока;
- алгоритм, использующий z-буфер;
- алгоритм обратной трассировки лучей.

### 1.5.1 Алгоритм Робертса

Алгоритм Робертса работает в объектном пространстве, осуществляя удаление невидимых линий выпуклых тел. Выполнение данного алгоритма можно разделить на 4 этапа [3].

1 Подготовка исходных данных.

2 Удаление ребер, экранируемых самим телом.

3 Удаление ребер, экранируемых другими телами.

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

На первом этапе для каждого тела формируется матрица, каждый столбец которой содержит коэффициенты уравнения плоскости:

$$ax + bx + cz + d = 0, \tag{1.1}$$

проходящей через соответствующую грань [3]. Данная матрица имеет вид:

$$V = \begin{pmatrix} a_1 & a_2 & \cdots & a_n \\ b_1 & b_2 & \cdots & b_n \\ c_1 & c_2 & \cdots & c_n \\ d_1 & d_2 & \cdots & d_n \end{pmatrix},$$

где $a_i$, $b_i$, $c_i$ и $d_i$ - коэффициенты уравнения (1.1) $i$-ой плоскости.

Матрица тела должна быть сформирована корректно. Это значит, что любая точка, лежащая внутри тела, должна лежать по положительную сторону от каждой грани тела. Если для какой-либо грани это условие не выполняется, соответствующий столбец матрицы умножается на -1 [3].

На втором этапе указывается расположение наблюдателя и направление его взгляда. Как правило, наблюдатель располагается в бесконечности на положительной полуоси z и смотрит в сторону начала координат. В однородных

координатах вектор такого направления равен [3]:

$$E = [0, 0, -1, 0].$$

Для определения невидимых граней нужно умножить вектор $E$ на матрицу $V$. Отрицательные компоненты результирующего вектора соответствуют невидимым ребрам тела [3].

На третьем этапе работы алгоритма для определения невидимых линий строится луч, соединяющий точку наблюдения с точкой на ребре. Точка невидима, если луч проходит через тело [3].

На четвертом этапе ведется поиск ребер, образовавшихся в результате взаимного протыкания тел. Вновь полученные ребра проверяются на экранирование телами [3].

Алгоритм Робертса дает высокую точность вычислений, однако сложен в реализации [3].

### 1.5.2 Алгоритм Варнока

Данный алгоритм работает в пространстве изображений. Суть алгоритма заключается в рассматривании окна и решении вопроса 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 подокна не станет достаточно простым для визуализации или его размер не достигнет требуемого предела разрешения [3].

Конкретная реализация алгоритма Варнока зависит от метода разбиения окна и от критериев, используемых для того, чтобы решить, является ли содержимое окна достаточно простым [3]. В оригинальной версии алгоритма Варнока каждое окно разбивается на четыре одинаковых подокна. Окно является пустым, когда все многоугольники сцены являются внешними по отношению к этому окну. Пределом разбиения является размер окна в 1 пиксель. Тогда определяется глубина каждого из рассматриваемых многоугольников в этой точке и изображается точка многоугольника, наиболее близкая к пользователю.

Алгоритм Варнока прост в реализации, однако возможны большие затраты по времени, если рассматривается область с большим количеством информации [3].

### 1.5.3 Алгоритм, использующий z-буфер

Данный алгоритм работает в пространстве изображений. В нем используются буфер кадра и z-буфер. Буфер кадра используется для запоминания интенсивности каждого пиксела в пространстве изображения, в то время как z- буфер — это отдельный буфер глубины, используемый для запоминания глубины (координаты z) каждого видимого пиксела в пространстве изображения [3].

В процессе работы значение z каждого нового пикселя, 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 пиксел заносится в этот буфер, и производится корректировка z-буфера новым значением z. Если же сравнение дает противоположный результат, то никаких действий не производится [3].

Главное преимущество алгоритма — простота. Кроме того, этот алгоритм решает задачу об удалении невидимых поверхностей и делает тривиальной визуализацию пересечений сложных поверхностей. Однако, несмотря на свою простоту, алгоритм требует использование большого объема памяти. Кроме того, недостаток алгоритма заключается еще в трудоемкости реализации эффектов прозрачности [3].

### 1.5.4 Алгоритм обратной трассировки лучей

Основная идея, лежащая в основе трассировки лучей, заключается в том, что наблюдатель видит любой объект посредством испускаемого неким источником света, который падает на объект и затем каким-то путем доходит до наблюдателя согласно законам оптики. Однако не все лучи света доходят

до наблюдателя, что делает процесс отслеживания этих лучей вычислительно неэффективным. Поэтому лучше отслеживать лучи в обратном направлении, то есть от наблюдателя к объекту, причем испуская лучи от точки наблюдателя через каждый пиксел экрана [4].

Для данной работы был выбран алгоритм обратной трассировки лучей, так как он, несмотря на малую производительность из-за большого количества вычислений, позволяет получить реалистичное изображение с учетом оптических явлений, таких как отражение и преломление.

## 1.6 Выбор модели освещения

Построение реалистических изображений включает в себя как физические, так и психологические процессы. В частности, при визуализации сцены учитываются законы оптики: отражение, преломление, поглощение света и т. д. Поэтому необходимо выбрать модель освещения для построения изображения.

### 1.6.1 Модель Ламберта

Модель Ламберта описывает диффузное освещение [3]. Модель показана на рисунке 1.1.

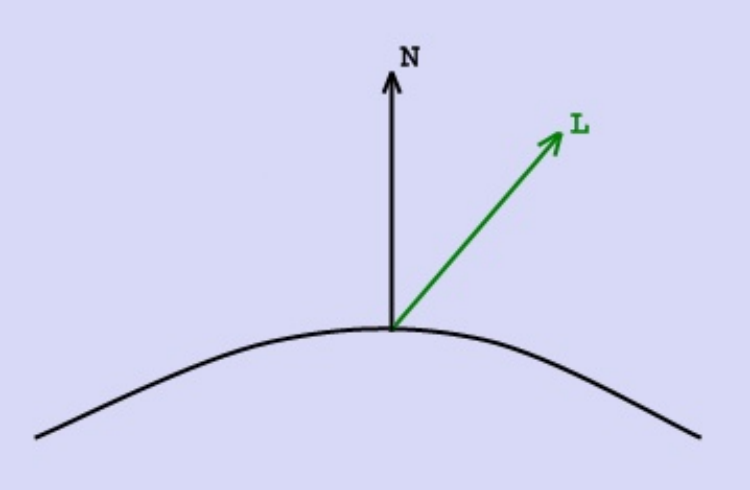


Рис. 1.1 – Модель освещения Ламберта

Свет точечного источника диффузно отражается от идеальной поверх-

ности по закону косинусов Ламберта: интенсивность отраженного света пропорциональна косинусу угла между направлением света L и нормалью к поверхности N [3]:

$$I = I_l k_d cos\theta, \tag{1.2}$$

где $I$ - интенсивность отраженного света,

$I_l$ - интенсивность точечного источника в направлении L,

$k_d$ - коэффициент диффузного отражения,

$\theta$ - угол между направлением света L и нормалью к поверхности n, $0 \leq \theta \leq \frac{\pi}{2}$

### 1.6.2 Модель Фонга

Модель Фонга — это классическая модель освещения, представляющая собой совокупность диффузной и зеркальной составляющих [3]. Модель представлена на рисунке 1.2.

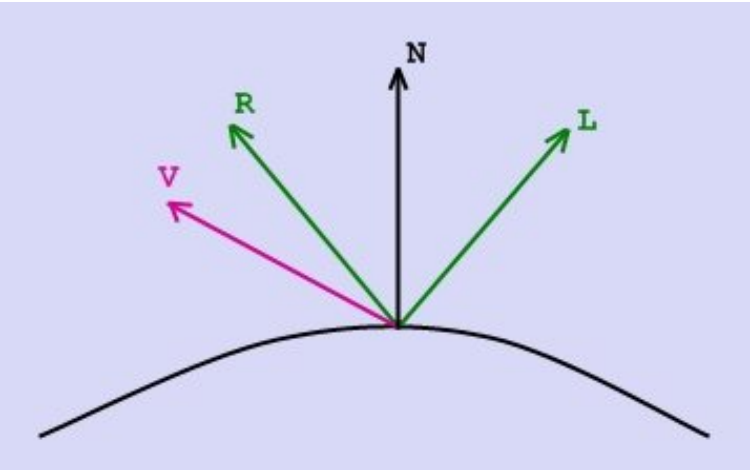


Рис. 1.2 – Модель освещения Фонга

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.

Зеркальная модели составляющая описывается выражением [3]:

$$I = I_l w(i, \lambda) cos^n \alpha, \tag{1.3}$$

где $I$ - интенсивность отраженного света,

$I_l$ - интенсивность точечного источника в направлении L,

$w(i, \lambda)$ - кривая отражения, представляющая отношение зеркально отраженного света к падающему как функцию угла падения $i$ и длины волны $\lambda$;

$n$ - степень, аппроксимирующая пространственное распределение зеркально отраженного света;

$\alpha$ - гол между вектором направления отраженного луча R и вектором направления на наблюдателя V.

Функция $w(i, \lambda)$ довольно сложна, поэтому обычно ее заменяют константой, которая либо выбирается из эстетических соображений, либо определяется экспериментально.

### 1.6.3 Глобальная модель освещения Уиттеда

Модель освещения Уиттеда показана на рисунке 1.3


Рис. 1.3 – Модель освещения Уиттеда

В данной модели луч V, падающий на поверхность в точке Q, отражается в направлении r и, если поверхность прозрачна, преломляется в направлении p. Тогда наблюдаемая интенсивность $I$ выражается формулой [3]:

$$I = k_a I_a + k_d \sum_j I_{l_j}(N; L_j) + k_s \sum_j I_{l_j}(S; R_j)^n + k_s I_s + k_t I_t, \qquad (1.4)$$

где $k_a$ - коэффициент рассеянного отражения,

$k_d$ - коэффициент диффузного отражения,

$k_s$ - коэффициент зеркального отражения,

$k_t$ - коэффициент пропускания,

$I_t$ - интенсивность света, падающего в точку Q по направлению p;

$I_s$ - интенсивность света, падающего в направлении r и зеркально отраженного к наблюдателю в точке Q;

$N$ - нормаль к поверхности в точке Q,

$L_j$ - направление на $j$-ый источник света,

$S$ и $R$ - локальные векторы наблюдения отражения,

$n$ - степень пространственного распределения Фонга.

Для вычисления интенсивности преломленного света можно применить закон Бугера-Ламберта-Бера для учета поглощения энергии при прохождении луча света через вещество [5]:

$$I_t = I_0 e^{-k_\lambda l}, \tag{1.5}$$

где $I_0$ - интенсивность света на входе в вещество,

$l$ - толщина слоя вещества, через который прошел свет;

$k_\lambda$ - показатель поглощения.

Для данной работы была выбрана модель освещения Уиттеда, так как она позволяет учесть такие оптические явления, как отражение и преломление света, что позволит нам получать более реалистичные изображения, чем модели Ламберта и Фонга.

# Вывод

В данном разделе были рассмотрены методы и алгоритмы, необходимые для написания программного обеспечения. Для описания объекта была выбрана модель полигональной сетки. Для представления поверхности слайма был выбран список граней, при этом информация о сетке будет храниться с помощью списка граней. Для физического моделирования слайма была выбрана модель масс с пружинами. В качестве алгоритма удаления невидимых

ребер и поверхностей была выбрана обратная трассировка лучей. Для учета освещения была выбрана глобальная модель Уиттеда.

# 2 Конструкторская часть

## 2.1 Общие сведения

Положение объектов сцены в пространстве описывается с помощью мировой системы координат, при этом ось Oz должная быть направлена вверх.

Общий алгоритм работы программы можно описать следующим образом:

1 Задать объекты сцены (камера, пол, источник света, слайм);

2 Для каждого кадра:

2.1. Если пользователь изменил характеристики объектов сцены, установить новые характеристики объектов;

2.2. Если пользователь пытается захватить точку слайма, найти и запомнить ее;

2.3. Определить силы, действующие на точки слайма;

2.4. Изменить скорости точек слайма в соответствии с действующими на них силами;

2.5. Переместить точки слайма, в соответствии с их скоростями;

2.6. Если пользователь захватил точку, вернуть ее в исходное положение;

2.7. Вывести изображение на экран.

## 2.2 Разработка структур данных, необходимых для хранения информации о слайме

Моделируемый объект можно представить в виде множества точек, соединенных между собой пружинами. На рисунке 2.1 приведена диаграмма структур данных, необходимых для хранения информации о слайме.

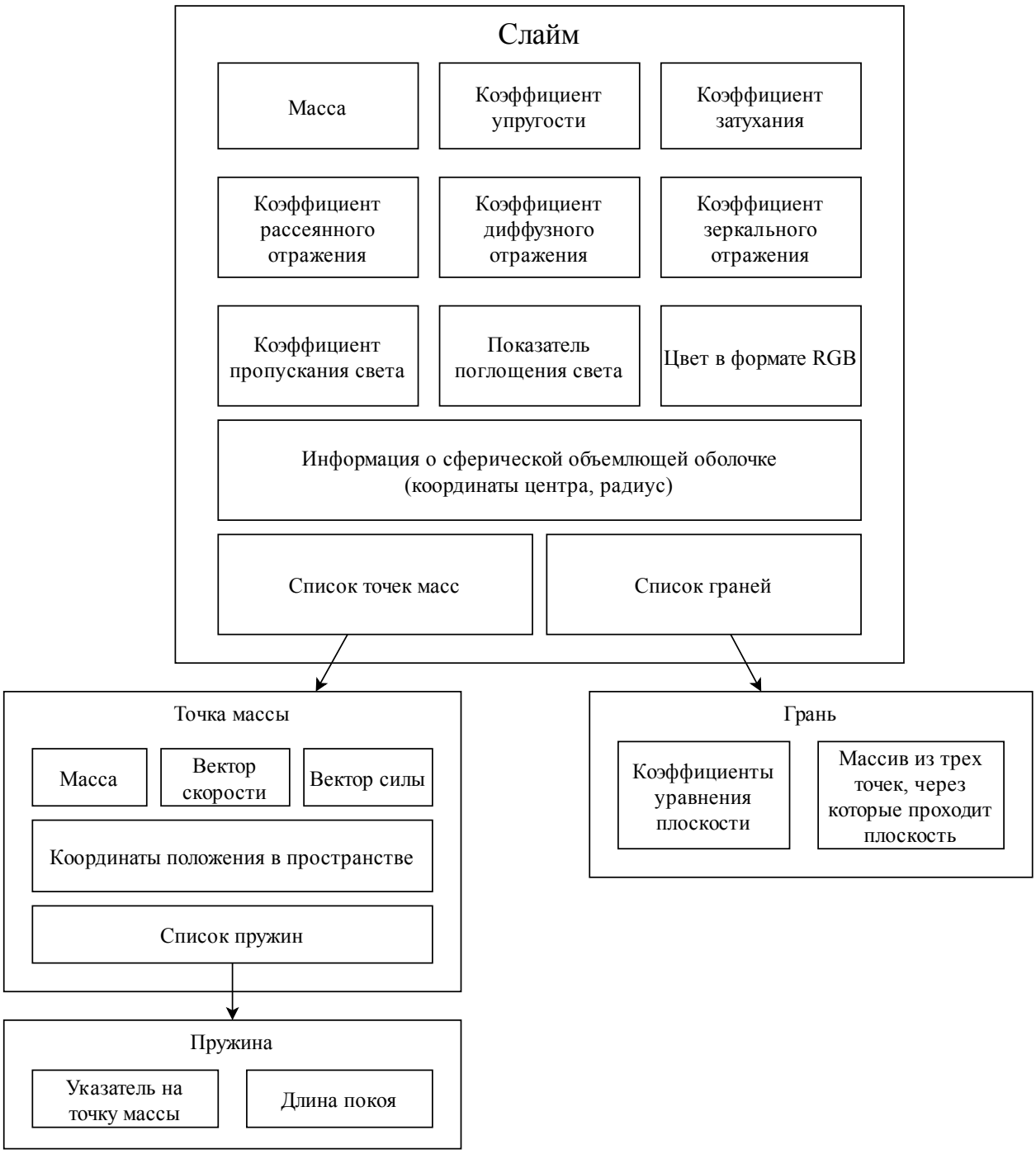


Рис. 2.1 – Структура хранения информации о слайме

Слайм содержит в себе данные для визуализации и физического моделирования, а также линейные односвязные списки точек масс и граней.

Точка массы описывает элемент поверхности слайма. Данная структура содержит в себе значение массы, вектора скорости и силы для физических расчетов, координаты точки и линейный односвязный список пружин. Сами пружины представляют собой пары, которые содержат в себе указатель на точку массы, с которой есть соединение, и значение длины покоя, при которой

сила упругости пружины равна нулю.

Грань нужна для корректной работы реализации алгоритма обратной трассировки. Она содержит в себе коэффициенты уравнения (1.1) плоскости и массив из трех точек, через которые проходит грань. Ребра, соединяющие эти точки, ограничивают плоскость, и поэтому грань имеет форму треугольника.

## 2.3 Разработка алгоритма генерации слайма

Для получения точек масс слайма производится рекурсивное деление исходных треугольных граней тела на новые треугольники посредством бисекции, то есть деления сторон треугольников пополам. До разбиения граней объем представляет собой икосаэдр. Таким образом, поверхность слайма будет представлять собой трехмерную фигуру с треугольными гранями. После разбиения, полученные точки масс соединяются друг с другом пружинами.

Пример разбиения граней приведен на рисунке 2.2.

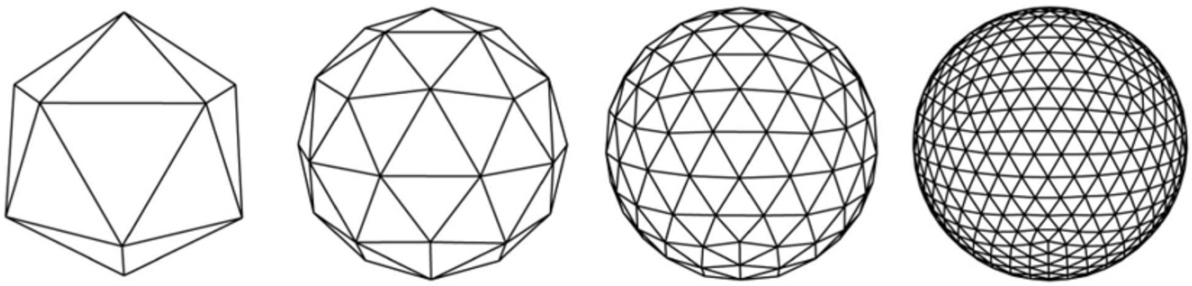

Рис. 2.2 – Деление граней икосаэдра посредством бисекции

## 2.4 Разработка алгоритма расчета физических параметров слайма

На рисунке 2.3 представлена схема алгоритма расчета физических параметров слайма.

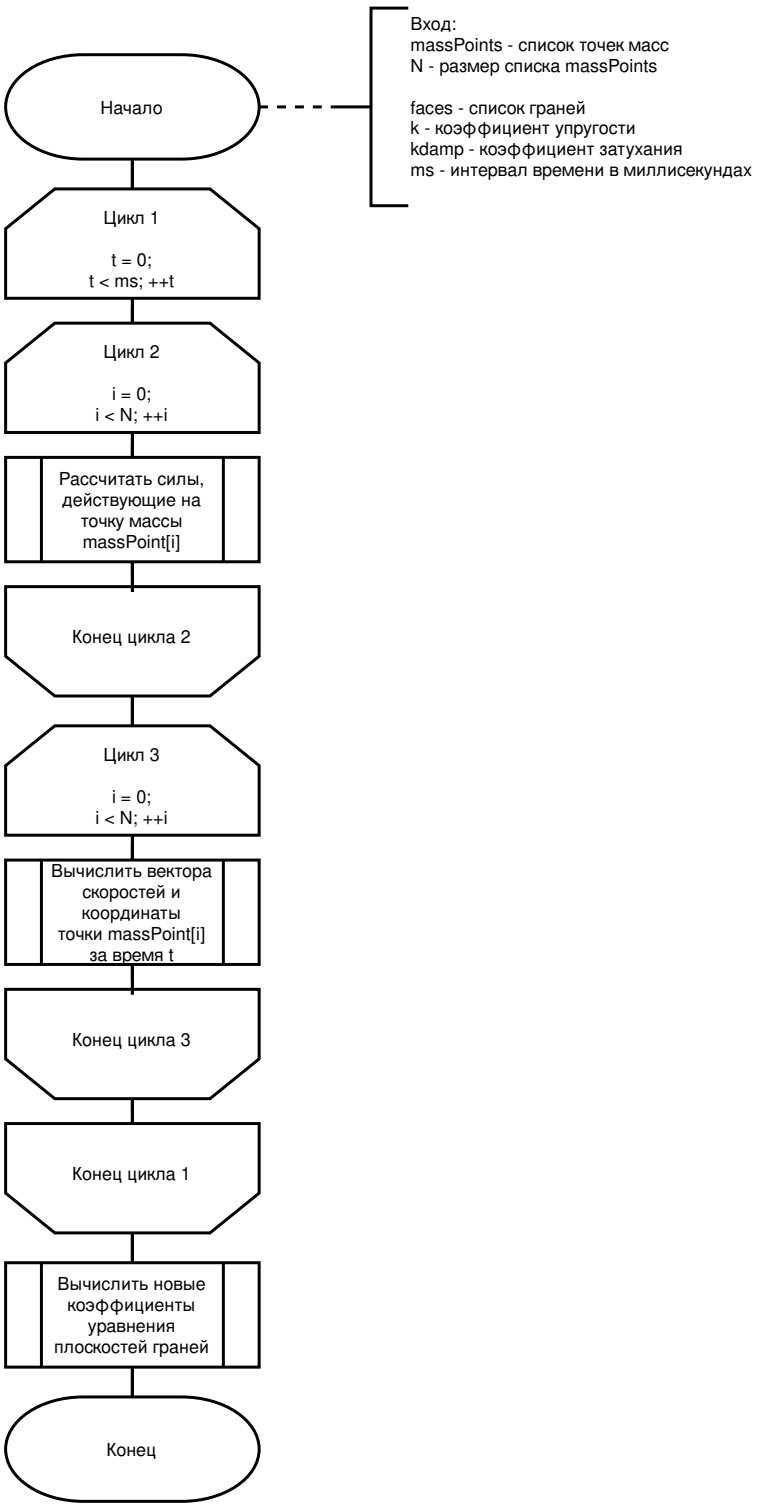


Рис. 2.3 – Схема алгоритма расчета физических параметров слайма

### 2.4.1 Вычисление векторов сил, действующих на точки масс

На каждую точку массы действуют три силы:

- сила тяжести;
- сила упругости со стороны пружины;
- сила затухания со стороны демпфера;

Вектор силы тяжести вычисляется по формуле:

$$F_{\text{тяж}_i} = m_i g, \tag{2.1}$$

где $m_i$ - масса $i$-ой точки,

$g$ - вектор ускорения свободного падения, $|g| = 9.81\frac{\text{Н}}{\text{м}}$.

Сила упругости определяется по закону Гука [2]:

$$F_{\text{упр}_{ij}} = -k(|x_{ij}| - l_{ij})\frac{x_{ij}}{|x_{ij}|}, \tag{2.2}$$

где $k$ - жесткость пружины, соединяющей точки масс $i$ и $j$;

$x_{ij}$ - разница радиус-векторов точек масс $j$ и $i$ соответственно,

$l_{ij}$ - расстояние между точками масс $i$ и $j$, при котором сила упругости равна нулю.

Затухающая сила вычисляется по формуле [2]:

$$F_{\text{сопр}_{ij}} = -k_d\frac{(v_{ij}; x_{ij})}{(x_{ij}; x_{ij})}\frac{x_{ij}}{|x_{ij}|}, \tag{2.3}$$

где $k_d$ - коэффициент затухания,

$x_{ij}$ - разница радиус-векторов точек масс $j$ и $i$ соответственно,

$v_{ij}$ - разница векторов скокростей точек масс $j$ и $i$ соответственно.

В итоге, используя формулы (2.1), (2.2) и (2.3), можно вычислить равнодействующую всех сил, приложенных к $i$-ой точку массы:

$$F_i = F_{\text{тяж}_i} + \sum_{j}^{K_i}(F_{\text{упр}_{ij}} + F_{\text{сопр}_{ij}}), \tag{2.4}$$

где $K_i$ - количество пружин, которые имеет $i$-ая точка массы.

### 2.4.2 Вычисление коэффициента упругости пружины

Из нашей модели, используемой для моделирования слайма, следует, что вся масса объекта распределяется по его поверхности, а не по объему. Из этого следует, что если всем пружинам, отвечающим за упругие взаимодействия между точками, присвоить одинаковые значения коэффициента упругости, то моделирование тела будет не совсем верным по эстетическим соображениям.

Для решения данной проблемы автор предлагает следующую идею: представить пружины как совокупности последовательно соединенных элементарных пружин одинаковой длины покоя $d_e$ и коэффициента упругости $k_e$. Тогда количество элементарных пружин будет равно:

$$n = \frac{d}{d_e}, \tag{2.5}$$

где $d$ - длина покоя пружины, соединяющей точки масс.

Наконец, коэффициент упругости системы последовательно соединенных пружин будет равен:

$$k = \frac{k_e}{n} = \frac{k_e d_e}{d}, \tag{2.6}$$

### 2.4.3 Вычисление новых координат точек масс

Вектор ускорения точки вычисляется по второму закону Ньютона:

$$a_i = \frac{F_i}{m_i}, \tag{2.7}$$

где $m_i$ - масса $i$-ой точки,

$F_i$ - равнодействующая всех сил, приложенных к $i$-ой точке.

В соответствии с полученным ускорением, вычисляется вектор скорости $i$-ой точки массы по формуле:

$$v_i = v_{0_i} + a_i t, \tag{2.8}$$

где $v_{0_i}$ - начальная скорость $i$-ой точки,

$t$ - время, в течение которого изменяется скорость под действием сил, $t = 1$ мс.

Новые координаты точки вычисляются по формуле:

$$P_i(x, y, z) = P_{0_i}(x_0, y_0, z_0) + v_{0_i}t + a_i t^2, \tag{2.9}$$

где $P_{0_i}(x_0, y_0, z_0)$ - радиус-вектор начального положения $i$-ой точки в пространстве,

$v_{0_i}$ - начальная скорость $i$-ой точки,

$a_i$ - ускорение $i$-ой точки,

$t$ - время, в течение которого изменяется скорость под действием сил.

## 2.5 Разработка рекурсивного алгоритма обратной трассировки лучей

На рисунке 2.4 приведена схема рекурсивного алгоритма обратной трассировки лучей.

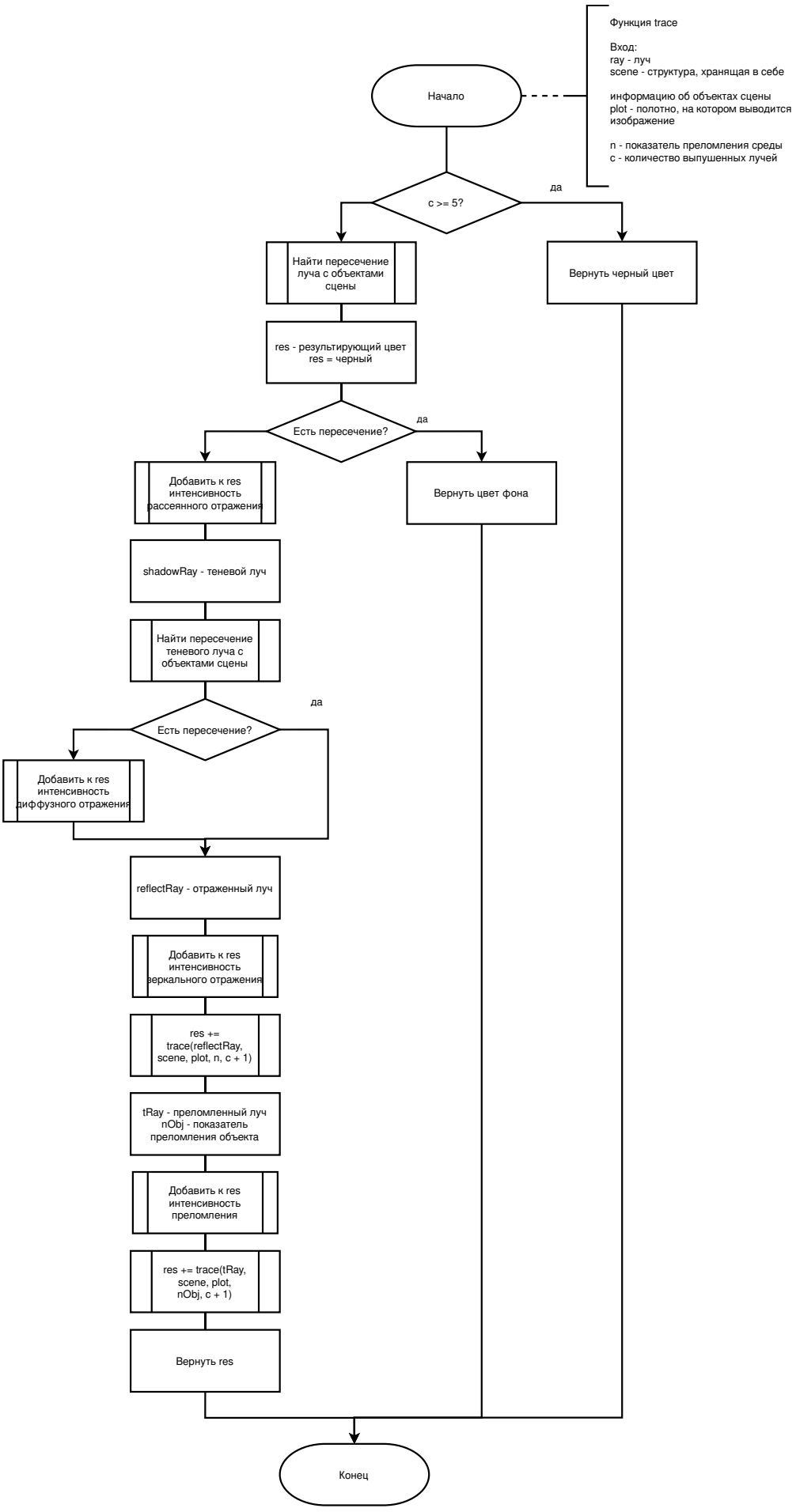


Рис. 2.4 – Схема алгоритма обратной трассировки лучей

## 2.6 Вычисление вектора отраженного луча

Вектор отраженного от плоской поверхности луча можно найти по формуле [6]:

$$v_r = v - 2(n; v)n, \tag{2.10}$$

где $v$ - вектор падающего на поверхность луча,

$n$ - нормаль к поверхности.

## 2.7 Вычисление вектора преломленного луча

Вектор преломленного луча определяется по формуле [6]:

$$v_t = \frac{n_0}{n}v - (\frac{n_0}{n}(N; v) + cos\theta_t)N, \tag{2.11}$$

где $v$ - вектор падающего на поверхность луча,

$N$ - нормаль к поверхности,

$n_0$ - показатель преломления среды, из которой падает луч $v$,

$n$ - показатель преломления среды, в которую падает луч $v$,

$\theta_t$ = угол между нормалью $n$ и преломленным лучом.

Угол $\theta_t$ вычисляется по закону Снеллиуса [6]:

$$\theta_t = \frac{n_0 sin(\theta)}{n} \tag{2.12}$$

где $\theta$ - угол между нормалью $n$ и вектором луча $v$.

## 2.8 Разработка алгоритма захвата точки слайма пользователем

Программа моделирования слайма должна обеспечивать управление объектом. В частности, пользователь должен иметь возможность растягивать и вдавливать тело.

На рисунке 2.5 приведена схема алгоритма захвата точки слайма пользователем.

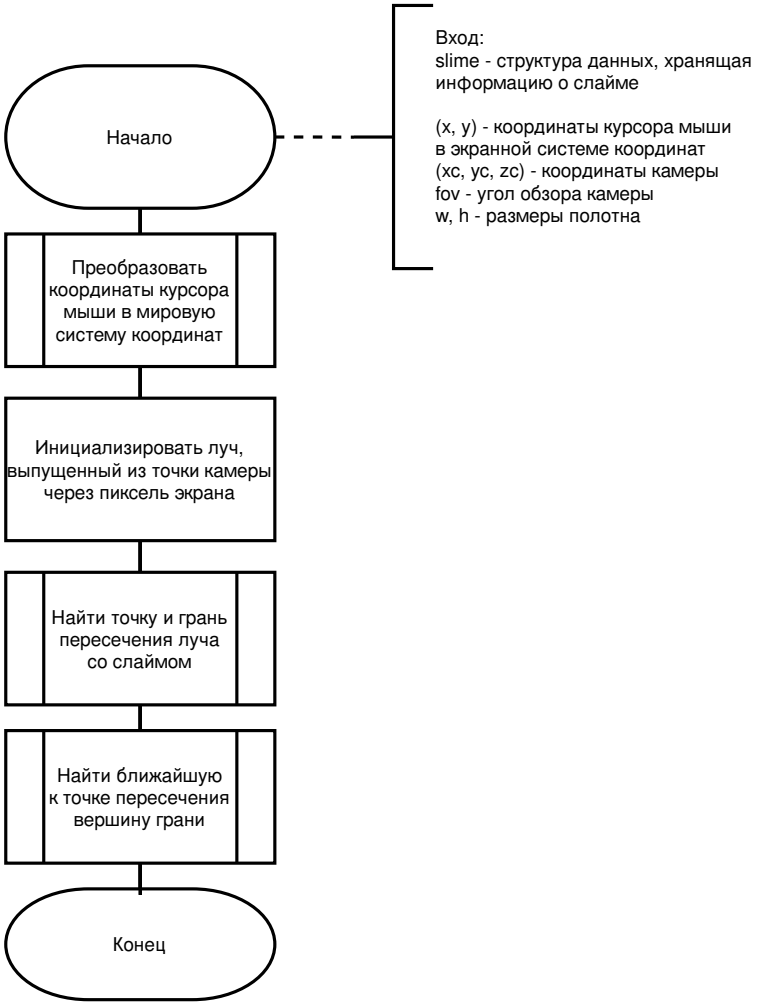


Рис. 2.5 – Схема алгоритма захвата точки слайма пользователем

## 2.9 Вычисление точки пересечения луча с треугольной гранью

Любую точку, лежащую на луче, можно описать следующим выражением:

$$P(x, y, z) = P_0(x_0, y_0, z_0) + tl(x_l, y_l, z_l), \tag{2.13}$$

где $P_0(x_0, y_0, z_0)$ - радиус-вектор начальной точки луча,

$l(x_l, y_l, z_l)$ - вектор направления луча,

$t \in \mathbb{R}, t \geq 0$.

Точку пересечения луча с треугольной гранью слайма можно найти с помощью барицентрического теста [7]. Суть данного метода заключается в том, что любую точку пространства можно перевести в барицентрическую систему координат, связанную с треугольником.

Пусть даны треугольная грань с вершинами $v_1$, $v_2$ и $v_3$. Тогда точка пересечения луча, описанного выражением (2.13), с гранью может быть определена с помощью решения:

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{(S; E_1)} \begin{bmatrix} (Q; E_2) \\ (S; T) \\ (Q; l) \end{bmatrix}, \tag{2.14}$$

где $(u, v)$ - барицентрические координаты точки пересечения,

$E_1 = v_2 - v_1$,

$E_2 = v_3 - v_1$,

$T = P_0 - v_1$,

$S = [l; E_2]$,

$Q = [T; E_1]$.

Если $u \in [0, 1]$, $v \in [0, 1]$ и $t \geq 0$, то найденная точка пересечения лежит внутри треугольника.

## 2.10 Вычисление точки пересечения луча со сферой

Для ускорения работы реализации алгоритма обратной трассировки лучей используется сферическая объемлющая оболочка слайма. Следовательно, встает вопрос об определении признака пересечения полупрямой с поверхностью шара.

Пусть луч задан выражением (2.13), а оболочка имеет центр в точке $C(x_c, y_c, z_c)$ и радиус $R$, то есть описывается уравнением:

$$(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = R^2. \tag{2.15}$$

Тогда полупрямая пересекает сферу, если имеет хотя бы одно неотрицательное решение следующее уравнение:

$$(x_0 + tx_l - x_c)^2 + (y_0 + ty_l - y_c)^2 + (z_0 + tz_l - z_c)^2 = R^2. \tag{2.16}$$

После раскрытия скобок и приведения подобных слагаемых получаем выражение:

$$at^2 + bt + c = 0, \tag{2.17}$$

где $a = x_l^2 + y_l^2 + z_l^2$,
$b = 2(x_l(x_0 - x_c) + y_l(y_0 - y_c) + z_l(z_0 - z_c))$,
$c = (x_0 - x_c)^2 + (y_0 - y_c)^2 + (z_0 - z_c)^2 - R^2$.

Полученное квадратное уравнение имеет хотя бы одно решение, если его дискриминант неотрицателен:

$$D = b^2 - 4ac. \tag{2.18}$$

Тогда корни вычисляются по формуле:

$$t_{1,2} = \frac{-b \pm \sqrt{D}}{2a}. \tag{2.19}$$

Если хотя бы одно найденное решение неотрицательно, то луч пересекает сферу.

# Вывод

В данном разделе были разработаны алгоритмы, необходимые для разработки программного обеспечения.

# 3 Технологическая часть

## 3.1 Требования к программе

К программе предъявляются следующие требования:

- программа должна предоставлять пользователю возможность взаимодействовать со слаймом;
- оконный интерфейс программы должен предоставлять пользователю возможность изменять такие параметры слайма, как цвет, коэффициент пропускания и показатель поглощения;
- должна быть возможность перезагрузки сцены;

## 3.2 Выбор инструментов разработки

Для выполнения данной работы был выбран высокоуровневый язык программирования C++. Данный выбор обусловлен следующими факторами:

- C++ поддерживает технологию объектно-ориентированного программирования, которая позволяет легко модифицировать программу и эффективно организовывать взаимодействия между объектами;
- C++ обладает очень высокими показателями вычислительной производительности, что также обеспечивает достойную скорость исполнения кода;
- для C++ было создано множество библиотек, которые могут быть использованы, в частности, для математических расчетов.

В качестве среды разработки была выбрана Visual Studio Code по следующим причинам:

- данная среда разработки бесплатна;

- в данной среде присутствуют встроенный отладчик, инструменты для работы с Git и средства навигации по коду, автодополнения кода и контекстной подсказки;
- имеется возможность расширить любой функционал, включая компиляцию и отладку.

Для создания графического пользовательского интерфейса был выбран фреймворк QT. Данный выбор обуславливается быстрой многоуровневой разработкой интерфейса и наличием классов для работы с изображениями. Соответственно, для сборки проекта была выбрана утилита qmake.

Для параллелизации работы алгоритма обратной трассировки лучей и вычисления векторов сил и новых координат точек масс моделируемого объекта была выбрана библиотека pthread. Данный выбор обуславливается возможностью ручной настройки потоков в соответствии со стандартом POSIX, который позволяет разрабатывать переносимые программы.

## 3.3 Диаграмма классов

На рисунке 3.1 изображена диаграмма классов, использующихся в программе.

В программе используются следующие абстракции:

- MainWindow — класс оконного интерфейса;
- Scene — класс, хранящий в себе информацию об объектах сцены;
- Plot — класс, использующийся для визуализации сцены;
- Grabber — класс, хранящий информацию о захваченной пользователем вершине слайма;
- Object — абстрактный класс объекта сцены;
- Camera — класс, хранящий информацию о камере;
- LightSource — класс точечного источника света;

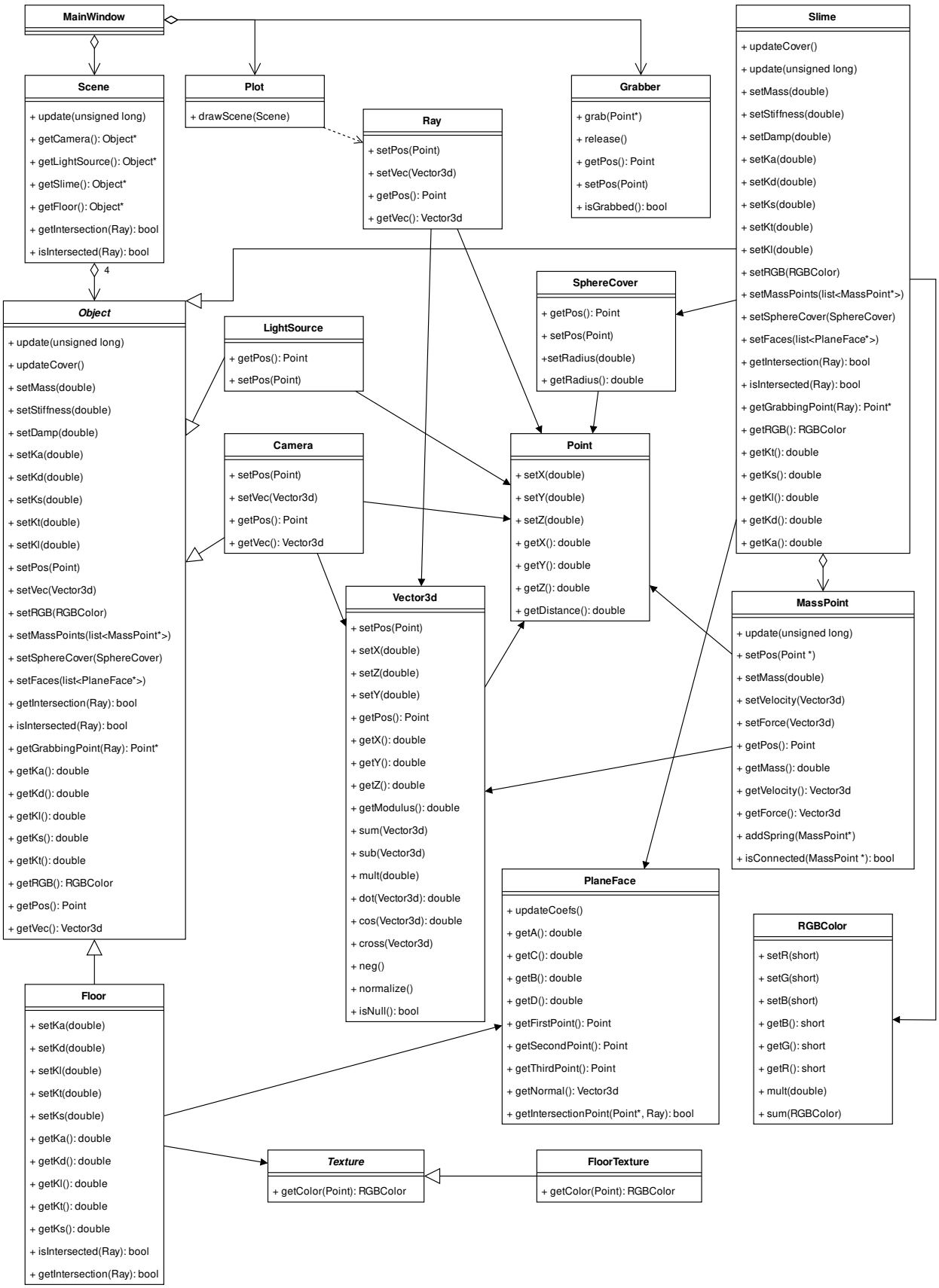


Рис. 3.1 – Диаграмма классов

- Floor — класс пола;

- Slime — класс слайма;
- Point — класс геометрической точки;
- Vector3d — класс трехмерного вектора;
- MassPoint — класс точки массы;
- RGBColor — класс, хранящий информацию о цвете в схеме RGB;
- PlaneFace — класс плоской треугольной грани;
- SphereCover — класс сферической объемлющей оболочки слайма;
- Texture — абстрактный класс текстуры объекта;
- FloorTexture — класс текстуры пола;
- Ray - класс луча.

## 3.4 Интерфейс программы

На рисунке 3.2 показан интерфейс программы.

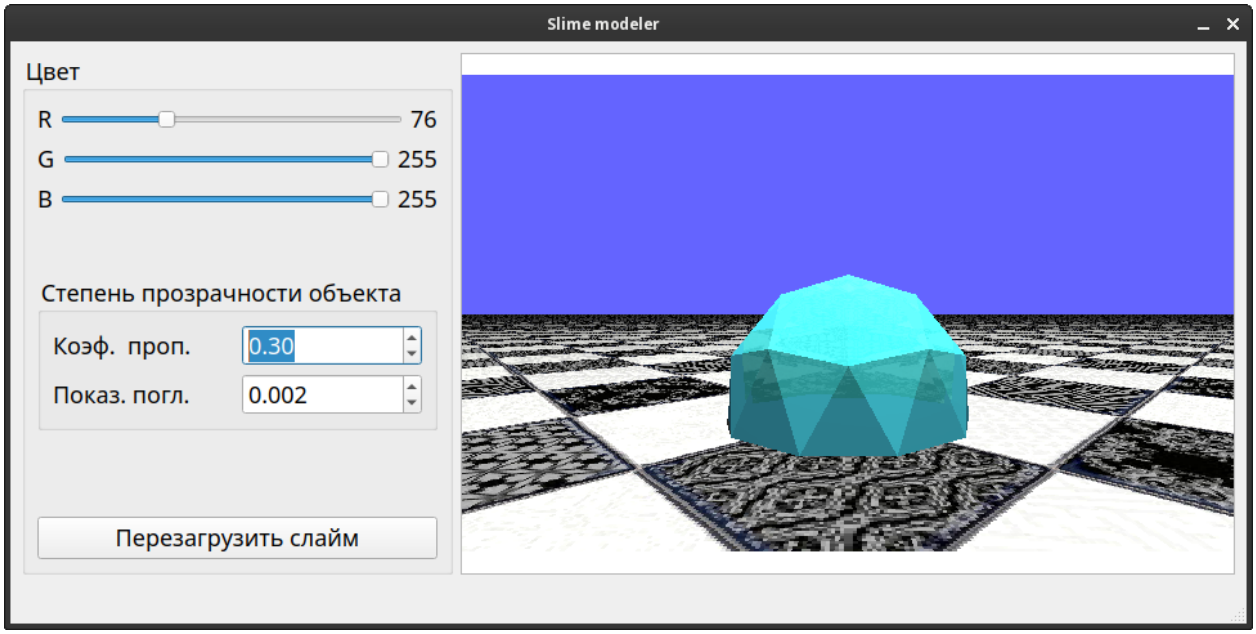


Рис. 3.2 – Интерфейс программы

Цвет слайма устанавливается с помощью трех слайдеров, отвечающих за красный, зеленый и синий компоненты цветовой модели RGB соответственно. Степень прозрачности устанавливается двумя полями для ввода вещественных чисел, отвечающих за коэффициент пропускания и показатель поглощения слайма. Кнопка «Перезагрузить слайм» используется для перезагрузки сцены.

## Вывод

В данном разделе были выбраны инструменты для разработки программного обеспечения. Были созданы графический пользовательский интерфейс и диаграмма классов.

# 4 Исследовательская часть

## 4.1 Цель исследования

Целью исследования является оценка временной эффективности параллельной реализации обратной трассировки лучей в зависимости от количества граней моделируемого объекта.

## 4.2 Технические характеристики электронно-вычислительной машины

Ниже приведены технические характеристики электронной вычислительной машины, на которой было произведено исследование работы программы:

- Fedora Linux 36 (Xfce) x86_64;
- ЦП Intel i7-10510U (8) @ 4.900 ГГц;
- ОЗУ 8 ГБ.

## 4.3 Описание исследования

В рамках данного исследования было изучено, как количество граней слайма влияет на временную эффективность работы параллельной реализации обратной трассировки лучей.

Количество граней слайма было равно 20, 80, 320, 1240, 4960 и 19840.

Коэффициент пропускания и показатель поглощения принимали значения: 0.3 и 0.001 соответственно.

Максимальная глубина рекурсии, реализующее алгоритм обратной трассировки лучей, равен 3.

Во время измерений со стороны пользователя не оказывалось каких-либо воздействий на слайм.

Для измерения времени работы реализации алгоритма был использован заголовочный файл chrono.

## 4.4 Результаты исследования

В таблице 4.1 приведены результаты измерений.

Таблица 4.1 – Результаты исследования

| **Количество граней объекта, шт.** | **Среднее время работы, мс** |
|---|---|
| 20 | 79 |
| 80 | 273 |
| 320 | 1158 |
| 1240 | 4191 |
| 4960 | 21875 |
| 19840 | 102175 |

На рисунке 4.1 приведен график результатов исследования.

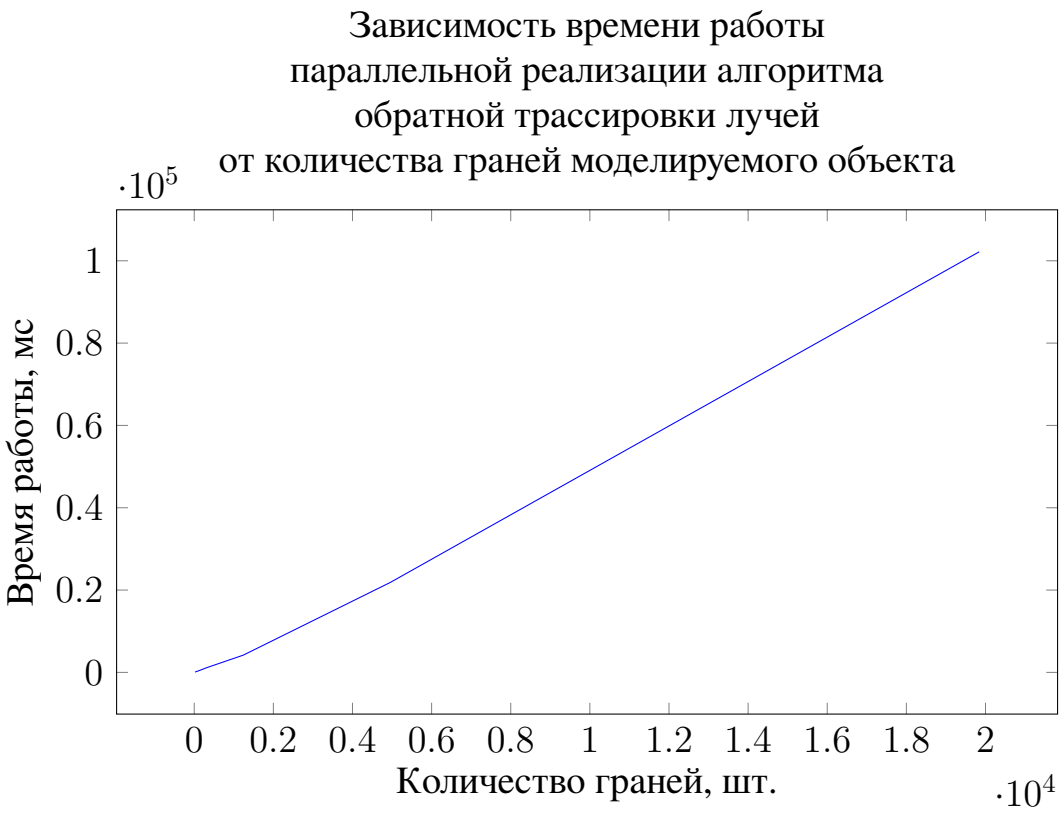


Рис. 4.1 – График результатов исследования

Из полученных результатов видно, что время работы параллельной реализации алгоритма обратной трассировки лучей линейно возрастает с количеством граней моделируемого объекта.

## 4.5 Вывод

В данном разделе было измерено время работы параллельной реализации алгоритма обратной трассировки лучей при разных количествах граней слайма. По полученным результатам можно сделать вывод, что время выполнения реализации алгоритма прямо пропорциональна количеству граней моделируемого объекта.

# Заключение

# Список использованных источников


1. Божко А.Н., Жук Д.М., Маничев В.Б., Компьютерная графика: Учеб. пособие для вузов. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2007. — 392 с.

2. Nealen A., Müller M., Keiser R., Boxerman E., Carlson M., Physically Based Deformable Models in Computer Graphics [Электронный ресурс] — Режим доступа: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.124.4664 (дата обращения 06.07.22)

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

4. Задорожный А.Г., Компьютерная графика: введение в трассировку лучей: учебное пособие — Новосибирск: Изд-во НГТУ, 2021. — 64 с.

5. Сивухин Д. В. Общий курс физики. — Издание 3-е, стереотипное. — М.: Физматлит, МФТИ, 2002. — Т. IV. Оптика. — 792 с.

6. Physically Based Rendering: From Theory to Implementation [Электронный ресурс] — URL: https://pbr-book.org/3ed-2018/contents

7. Фролов В., Фролов А. Пересечение луча и треугольника [Электронный ресурс] — URL: http://www.ray-tracing.ru/articles213.html (дата обращения: 10.07.2022).