Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет
имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ
и информационные технологии»

РАСЧЁТНО - ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту на тему:

Визуализация взаимодействия частиц при столкновении (взрыв)

Студент _____ Сиденко А. Г.
(Подпись, дата)

Руководитель курсового проекта _____ Кострицкий А. С.
(Подпись, дата)

Москва 2019

РЕФЕРАТ

Отчет содержит 43 стр., 21 рис., 9 источн., 1 прил.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На протяжении десятилетий взрывы были самыми динамичными и визуально привлекательными спецэффектами в кино и видеоиграх. Они стали настолько заметными в боевиках и приключенческих фильмах, что кажется необычным, когда его нет в фильме. Каким был бы фильм "Звездные войны"без финального взрыва Звезды Смерти?

Традиционно взрывные эффекты создаются перед камерой, а не в компьютере. Либо строится уменьшенная модель и взрывается перед высокоскоростными камерами, либо используются настоящие взрывчатые вещества.

Правда в кинематографе, важна эффектность, поэтому все заранее планируется в мельчайших подробностях.

Если нужно взорвать настоящее здание, пусть даже специально для этого построенное, его предварительно готовят: подпиливают рамы так, чтобы нужная часть их осталась на месте, а нужная – разлетелась на части. Также это делается для того, чтобы здание развалилось при минимально необходимом заряде. Все части, что должны развалиться, делаются из легких материалов с тем, чтобы если какие-то куски случайно упадут на голову кинематографистам, ущерб был бы минимален.

Если при этом в кадре должны быть персонажи, с людьми скрупулезно отрабатываются все их передвижения, фиксируется время, необходимое для того, чтобы они достигли безопасного расстояния до того, как произойдет главный взрыв. [1]

Существует множество веских причин для использования компьютеров для создания взрывных эффектов вместо более традиционных практических методов. Основной мотивацией, конечно, забота о безопасности актеров. Когда взрыв происходит полностью внутри компьютера, нет никаких шансов на то, что кто-то случайно попадет в зону взрыва. Также компьютерные взрывы дешевле и быстрее, чем точное масштабирование и размещение детонаторов, а также огнестойкость существующих

конструкций или создание специальных миниатюр. Когда режиссеры снимают практический взрыв, они настраиваются на несколько дней, чтобы получить несколько ракурсов на один единственный взрыв. С помощью компьютерных эффектов режиссеры могут посмотреть на промежуточный результат и попросить что-то изменить, чтобы более точно отразить свое творческое видение.

Целью проекта является создание максимально приближенной модели взрыва большого числа частиц, при столкновении с телом, имеющим больший размер с использованием графического редактора систем частиц. Моделирование основано на физическом явлении взрыва взрыва и возникающих побочных эффектов в заданном пространстве за заданное время и взаимодействующих с окружающей средой.

Для достижения поставленной цели необходимо решить следующие задачи:

а) Определить понятие системы частиц.

б) Создать движок для работы с частицами.

в) Изучить физическое явление - взрыв.

г) Смоделировать взрыва большого числа частиц, при столкновении с телом

1 Аналитический раздел

Целью работы является создание максимально приближенной модели взрыва большого числа частиц. Объекты в сцене представлены в виде твердых частиц, которые приводятся в движение различными силами.

1.1 Анализ предметной области

Система частиц – широко используемый в компьютерной графике метод представления объектов, не имеющих четки геометрических границ. Облака, туманности, дым, взрыв, снег – все эти объекты моделируются с помощью систем частиц. Системы частиц создают объем частиц с индивидуальными свойствами, а не поверхность с текстурой. При таком подходе возможно изменять внешний вид каждой отдельной частицы с течением времени и, таким образом, создавать визуализации, которые реалистично отображают огонь, дым, воду, взрывы и с некоторыми изменениями даже волосы и мех. Они также используются для представления неестественных явлений, таких как магические эффекты. [2]

Уже в начале 1980-х годов Уильям Т. Ривз использовал системы частиц для моделирования огня, поглощающего планету, в фильме «Звездный путь II: Гнев Хана». Ривз считается изобретателем систем частиц, и в 1983 году он опубликовал статью, объясняющую, как это было сделано. Ривз продолжил развивать эту идею и с тех пор вносил больший вклад в область систем частиц. Разработка продолжалась, и сегодня большинство игровых движков содержат систему частиц. [3]

Эта техника имеет много полезных приложений, будь то большие взрывы, пыль, пожары или большие толпы людей, спецэффекты необходимы во многих фильмах, анимациях, съемках или компьютерных играх. Фильмы, показывающие массивные взрывы и разрушенные здания, могут имитировать эти эффекты, а не создавать их по-настоящему. В

этой работе мы сосредоточимся на визуализации взрыва большого числа частиц, при столкновении с телом, имеющим больший размер.

Взрыв в реальности состоит из мелких частиц, которые движутся независимо друг от друга, однако, поскольку частицы будут взаимодействовать в основном одинаковыми силами, они будут вести себя одинаково.

Визуализация взрыва частиц может быть описана по разному в зависимости от источника силы, порождающий взрыв. В данной работе, источником является шарообразное тело, врезающееся в систему более мелких частиц.

На сцене располагается неподвижная, в начальный момент, группа частиц, расположенных вместе, шарообразное подвижное тело, точечный источник света.

В начальный момент времени, есть возможность скорректировать размеры тела, поменять расположение камеры.

После нажатия на кнопку, произойдёт симуляция.

На рисунке 1.1 изображена **функциональная модель**, отображающая структуру и функции системы.

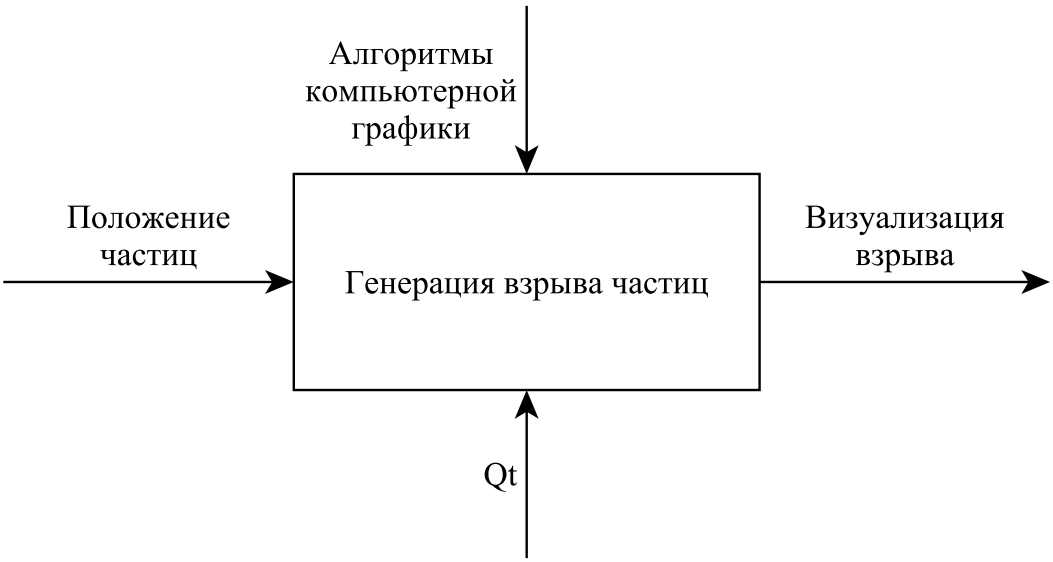

Рисунок 1.1 — Функциональная модель процесса визуализации взрыва частиц

На вход принимаются частицы с различными параметрами (положение, масса, размер) и положение шарообразного врезающегося тела, с идентичными параметрами. На выходе получаем визуализацию взрыва мелких частиц при столкновении с телом.

В рамках данного исследования существует возможность, менять параметры частиц.

Ставится задача получения визуализации взрыва в режиме реального времени.

1.2 Обзор и анализ существующих решений, обоснование необходимости разработки

В настоящее время при 3D моделировании объекты часто создают в основном только двумя способами. [4]

а) Либо с помощью плоских полигонов — тем самым будет создана полая модель без внутреннего наполнения.

б) Либо с помощью частиц, которые полностью заполняют внутренности модели, где каждая частица представляется как материальная точка с дополнительными атрибутами, такими как скорость, цвет, ориентация в пространстве, угловая скорость и т. п.

Ввиду того, что полигональные модели пусты по своей природе, очень трудно моделировать их поведение в 3D мире, как, например, всплески воды. Если же воду моделировать через систему чпстиц, то всё становится гораздо проще, так как вся вода от поверхности океана и до дна состоит из атомов, которые можно рассматривать, как набор отдельных частиц.

Далее рассматриваются алгоритмы удаления невидимых поверхностей. [5]

а) Алгоритм, использующий z-буфер

б) Алгоритм разбиения области Варнока

в) Метод трассировки лучей

Алгоритм удаления поверхностей с Z-буфером.

Алгоритм предложен Эдом Кэтмулом и представляет собой обобщение буфера кадра. Обычный буфер кадра хранит коды цвета для каждого пиксела в пространстве изображения. Идея алгоритма состоит в том, чтобы для каждого пиксела дополнительно хранить еще и координату Z или глубину. При занесении очередного пиксела в буфер кадра значение его Z-координаты сравнивается с Z-координатой пиксела, который уже находится в буфере. Если Z-координата нового пиксела больше, чем координата старого, т.е. он ближе к наблюдателю, то атрибуты нового пиксела и его Z-координата заносятся в буфер, если нет, то ни чего не делается. Таким образом, после выполнения этих операций в каждой точке экрана будет находиться пиксель, соответствующий грани, находящейся ближе всего к картинной плоскости в данной точке.

Этот алгоритм наиболее простой из всех алгоритмов удаления невидимых поверхностей, но требует большого объема памяти. Время

работы алгоритма не зависит от сложности сцены. Многоугольники, составляющие сцену, могут обрабатываться в произвольном порядке.

Основной недостаток алгоритма с Z-буфером - дополнительные затраты памяти. Другие недостатки алгоритма с Z-буфером заключаются в том, что так как пикселы в буфер заносятся в произвольном порядке, то возникают трудности с реализацией эффектов прозрачности или просвечивания и устранения лестничного эффекта.

Алгоритм разбиения области Варнока.

Алгоритм работает в пространстве изображения и анализирует область на экране дисплея (окно) на наличие в них видимых элементов. Если в окне нет изображения, то оно просто закрашивается фоном. Если же в окне имеется элемент, то проверяется достаточно ли он прост для визуализации. Если объект сложный, то окно разбивается на более мелкие, для каждого из которых выполняется тест на отсутствие и/или простоту изображения. Рекурсивный процесс разбиения может продолжаться до тех пор пока не будет достигнут предел разрешения экрана (1 пиксель).

Можно выделить 4 случая взаимного расположения окна и многоугольника (рис. 1.2):

а) многоугольник целиком вне окна,

б) многоугольник целиком внутри окна,

в) многоугольник пересекает окно,

г) многоугольник охватывает окно.

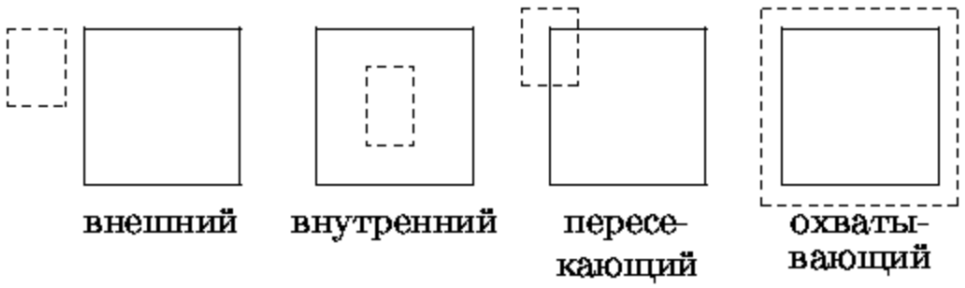

Рисунок 1.2 — Соотношения между окном экрана (сплошная рамка) и многоугольником (штриховая рамка)

В четырех случаях можно сразу принять решение о правилах закраски области экрана:

а) все многоугольники сцены - внешние по отношению к окну. В этом случае окно закрашивается фоном;

б) имеется всего один внутренний или пересекающий многоугольник. В этом случае все окно закрашивается фоном и затем часть окна, соответствующая внутреннему или пересекающему окну закрашивается цветом многоугольника;

в) имеется единственный охватывающий многоугольник. В этом случае окно закрашивается его цветом.

г) имеется несколько различных многоугольников и хотя бы один из них охватывающий. Если при этом охватывающий многоугольник расположен ближе остальных к наблюдателю, то окно закрашивается его цветом.

В любых других случаях процесс разбиения окна продолжается.

Алгоритм работает с многоугольниками.

Алгоритм трассировки лучей.

Суть метода заключается в том, что для каждой точки на проектируемой плоскости строится луч, начало которого совпадает с положением наблюдателя, а угол луча по отношению к направлению наблюдения определяется характеристиками наблюдателя (максимальный угол зрения относительно горизонтальной и вертикальной плоскостей) и точкой плоскости проецирования (картинная плоскость), для которой строится луч.

После построения луча определяется его пересечение со всеми гранями, из которых состоит сцена и выбирается та грань, расстояние от которой до наблюдателя минимально.

Далее в данной точке картинной плоскости рисуется пиксель цветом, определяемым в точке пересечения луча и выбранной грани.

Этот метод является довольно простым и позволяет совместить определение видимости с расчётом цвета соответствующего пикселя. Ещё одним преимуществом метода является то, что сцена может состоять не из треугольников, а быть задана набором геометрических примитивов (например, поверхностями второго порядка). В этом случае пересечение луча с геометрическим примитивом вычисляется аналитически (точное решение задачи построения геометрического примитива), в отличие от разбивания произвольного геометрического примитива (например, сферы) на треугольники и решения более простой задачи пересечения луча с треугольником.

Могут генерироваться новые лучи внутри сцены для корректного отображения отражений, преломлений, затенений – алгоритм обратной трассировки лучей более эффективен.

Может быть модифицирован для отображения общего освещения сцены.

Тем не менее для реализации метода требуется довольно большое количество вычислений и следовательно большие временные затраты.

Актуальность темы исследования обусловлена тем, что за последние несколько лет технология виртуальной реальности совершила огромный скачок в развитии и расширении сфер применения. Если раньше эта технология в основном применялась в военной промышленности и компьютерных играх, то сейчас виртуальная реальность проникает практически во все сферы деятельности человека: медицину, образование, архитектуру, рекламу и прочее. Эта технология имеет огромный потенциал и поэтому она так активно развивается.

1.3 Выбор, обоснование метода моделирования и алгоритма

На основании предложенных методов моделирования, в данной работе будет использоваться система частиц (а именно сфер).

Для удаления невидимых поверхностей будет реализован алгоритм обратной трассировки лучей, несмотря на вычислительную сложность, алгоритм предоставляет возможность работать непосредственно со сферическими поверхностями и отображать освещение.

1.4 Вывод

В данной работе ставится задача визуализации взрыва системы частиц при столкновении с шарообразным телом, используя алгоритм обратной трассировки, как наиболее оптимальный для поставленной задачи.

На сцене располагается неподвижная, в начальный момент, группа частиц, расположенных вместе, шарообразное подвижное тело, точечный источник света.

В начальный момент времени, есть возможность скорректировать размеры тела, поменять расположение камеры.

После нажатия на кнопку, произойдёт симуляция.

2 Конструкторский раздел

После запуска программы загружается модель объекта из файла в формате txt, выполняется генерация начальной картинки, устанавливаются начальные параметры (поворот камеры, размеры объектов), и после нажатия кнопки «Старт» начинается визуализация взрыва. Во время работы программы пользователь может изменять положение камеры, рассматривая модель с разных точек.

2.1 Математические основы метода математического моделирования

Для удаления невидимых поверхностей выбран алгоритм обратной трассировки лучей. А для реализации взрыва, была выбрана система частицы сферической формы.

2.1.1 Трассировка лучей

Во-первых, требуется задать точку обзора — это место, в котором располагается глаз, оно обычно называется положением камеры. Обозначим положение камеры $O(O_x, O_y, O_z)$.

Во-вторых, в начальный момент ориентация камеры направлена вдоль положительной оси Z, положительная ось Y направлена вниз, а положительная ось X – вправо, представлено на рисунке 2.1.

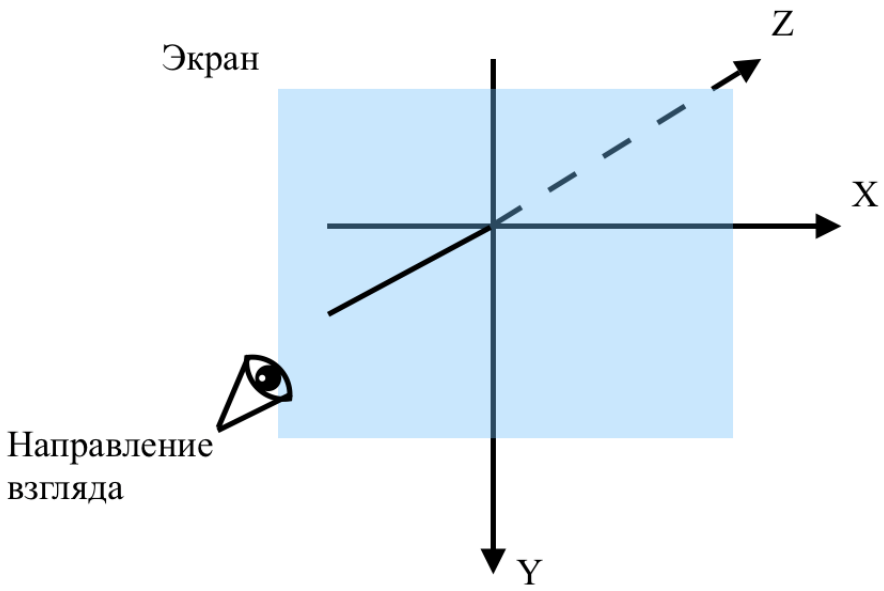

Рисунок 2.1 — Экранная система координат с позицией камеры в начальный момент

Экран - прямоугольник, называется окном просмотра. В сущности, на холсте отображается всё то, что видим через окно просмотра.

Далее необходимо определить для каждого пикселя окна просмотра (V) какого же он цвета.

В реальном мире свет исходит из источника света (солнца, лампочки и т.д.), отражается от нескольких объектов и наконец достигает наших глаз. Необходимо трассировать лучи «в обратном порядке» – следует начать с луча, находящегося на камере, проходящего через точку в окне просмотра и двигаясь, пока он не столкнётся с каким-нибудь объектом в сцене. Этот объект будет «виден» из камеры через эту точку окна просмотра. То есть в качестве первого приближения следует взять цвет этого объекта как «цвет света, прошедшего через эту точку». Алгоритм представлен на рисунке 2.2.

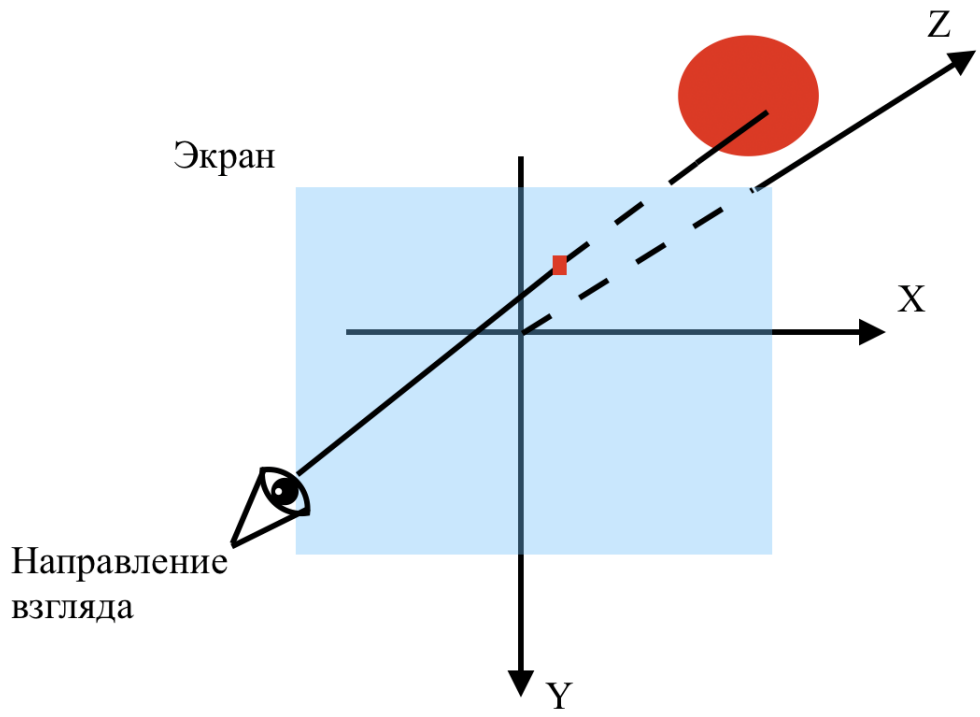

Рисунок 2.2 — 1 шаг алгоритма трассировки лучей

Наилучшим способом представления лучей для поставленной цели будет использование параметрического уравнения. Луч проходит через O, и его направление (из O в V), поэтому любую точку P луча можно представить как $P = O + t \cdot (V - O)$, где t – произвольное действительное число.

Обозначим направление луча за $\vec{D} = \overrightarrow{(V - O)}$. Тогда уравнение примет вид: $P = O + t\vec{D}$.

Следующим шагом необходимо рассмотреть объекты сцены, с которыми лучи сталкиваются. В сцене присутствует плоскость и сферы.

Для отыскания точки пересечения луча с произвольной поверхностью необходимо знать аналитические уравнения, определяющие оба эти объекта в трехмерном пространстве. Точка пересечения удовлетворяет всем уравнениям, так как принадлежит и лучу, и поверхности. Поэтому, сводя уравнения в систему и находя ее решения, мы получаем координаты этой точки.

Пересечение со сферой.

Пусть из точки O выпущен луч в направлении $\vec{d}$, как показано на рисунке 2.3.

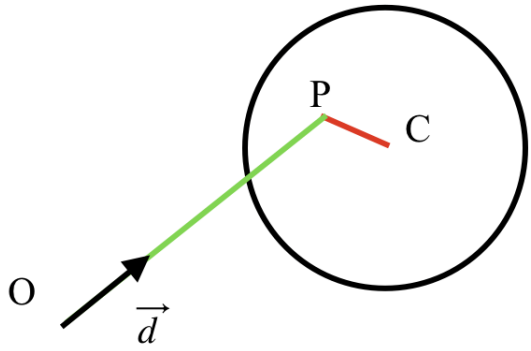

Рисунок 2.3 — Пересечение луча со сферой

Точкой пересечения луча и сферы является P – решение системы 2.1.

$$\begin{cases} P = O + t\vec{d} \\ (P - C)^2 = r^2 \end{cases} \tag{2.1}$$

$$(O + t\vec{d} - C)^2 = r^2$$

Обозначим $\vec{s} = C - A$, тогда

$$(t\vec{d} - \vec{s})^2 = r^2$$

$$t^2\vec{d}^2 - 2t \cdot (\vec{d}, \vec{s}) + \vec{s}^2 = r^2$$

, где (d,s) — скалярное произведение $\vec{d}$ и$\vec{s}$, равное сумме попарных произведений их координат.

Получилось квадратное уравнение.

$$D = (\vec{d}, \vec{s})^2 - \vec{d}^2(\vec{s}^2 - r^2)$$

Если $D < 0$, то луч проходит мимо сферы, если же $D \leq 0$, то уравнение имеет действительные корни:

$$t_{1,2} = \frac{(\vec{d}, \vec{s}) \pm sqrt(D)}{\vec{d}^2}$$

Наименьшее положительное значение t, если оно существует, дает ответ задачи. Если же положительного значения нет, то луч сферу не пересекает.

Если точка пересечения найдена, то нас будет интересовать также нормаль к поверхности в этой точке, так как эта нормаль определяет направление отраженного луча. Формула 2.2.

$$\vec{n} = \frac{P - O}{|P - O|} \tag{2.2}$$

Пересечение с плоскостью.

Плоскость задается общим уравнением: $(P, n) = r$, где n – вектор нормали с плоскости, имеющий единичную длину, а r – расстояние от начала координат до плоскости (с точностью до знака). См. рисунок 2.4.

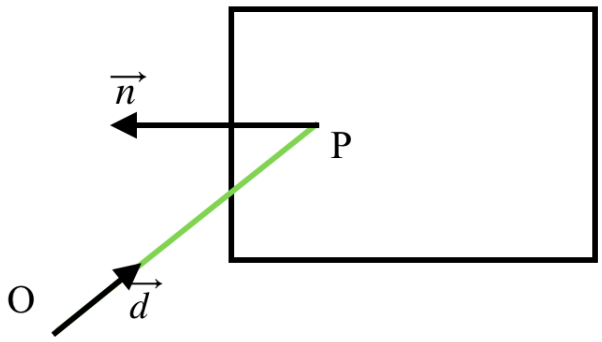

Рисунок 2.4 — Пересечение луча с плоскостью

Точкой пересечения луча и плоскости является P – решение системы 2.3.

$$\begin{cases} P = O + t\vec{d} \\ (P, \vec{n}) = r \end{cases} \tag{2.3}$$

Тогда:

$$t = \frac{r - (O, \vec{n})}{(\vec{d}, \vec{n})}$$

Если $(d,n) = 0$, то луч параллелен плоскости, т.е. не пересекает ее. Если же $(d,n) \neq 0$, то вычисляем t. Тогда если его значение положительно, то луч пересекает плоскость.

Источники освещения

Свет должен откуда-то поступать.

Точечный источник испускает свет из фиксированной точки в пространстве, называемой его позицией. Свет испускается равномерно во всех направлениях; именно поэтому его также называют всенаправленным освещением. Следовательно, точечный источник полностью характеризуется его позицией и яркостью.

Тогда $\vec{l}$ – направление из точки P в сцене к источнику освещения Q. Этот вектор, называемый световым вектором, просто равен $\overrightarrow{Q-P}$.

Для вычисления освещённости точки нам просто нужно вычислить количество света, вносимое каждым источником и сложить их, чтобы получить одно число, представляющее общее количество полученного точкой освещения. Затем необходимо вычислить новое значение цвета пикселя, используя цветовую модель HSV (англ. Hue, Saturation, Value — тон, насыщенность, значение).

Диффузное рассеяние

Когда луч света падает на матовый объект, то из-за неровностей его поверхности, он отражает луч в сцену равномерно во всех направлениях, то есть получается «рассеянное» («диффузное») отражение.

С другой стороны, количество отражённого света зависит от угла между лучом света и поверхностью. Интуитивно это понятно – энергия, переносимая лучом, в зависимости от угла должна распределиться по меньшей или большей поверхности, то есть энергия на единицу площади, отражённая в сцену, будет соответственно выше или ниже:

Чтобы выразить это математически, используется вектор нормали. Вектор нормали, или просто «нормаль» — это вектор, перпендикулярный

поверхности в какой-то точке. Также он является единичным вектором, то есть его длина равна 1. См. формулу 2.2 или уравнение плоскости.

Итак, луч света с направлением $\vec{l}$ и яркостью I падает на поверхность с нормалью $\vec{n}$. Какая часть I отражается обратно на сцену. Схема диффузного отражения изображена на рисунке 2.5.

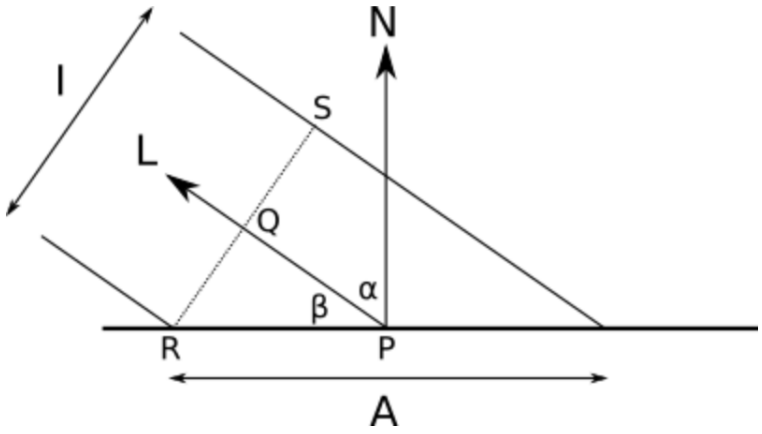

Рисунок 2.5 — Диффузное отражение

Поскольку технически луч света не имеет ширины, поэтому считается, что всё происходит на бесконечно малом плоском участке поверхности. Даже если это поверхность сферы, то рассматриваемая область настолько бесконечно мала, что она почти плоская относительно размера сферы, так же как Земля выглядит плоской при малых масштабах.

Луч света с яркостью I падает на поверхность в точке P под углом β. Нормаль в точке P равна $\vec{n}$, а энергия, переносимая лучом, распределяется по A. Нам нужно вычислить $\frac{I}{A}$.

Будем считать SR «шириной» луча. По определению, она перпендикулярна $\vec{l}$, который также является направлением PQ. Поэтому PQ и QR образуют прямой угол.

Давайте рассмотрим треугольник PQR.

$$QR = \frac{I}{2}$$

$$PR = \frac{A}{2}$$

Тогда

$$cos(\alpha) = \frac{QR}{PR} = \frac{I}{A}$$

α – угол между $\vec{n}$ и $\vec{l}$. То есть

$$cos(\alpha) = \frac{(\vec{n}, \vec{l})}{|\vec{n}||\vec{L}|}$$

Тени

Тени появляются там, где есть свет, но его лучи не могут достичь объекта, потому что на их пути есть другой объект.

Рассматривается луч, который проходит из точки до источника освещения. Пересекает ли этот луч другой объект? Если нет, то между точкой и источником ничего нет, то есть мы можем вычислить освещённость от этого источника и прибавить его к общей освещённости. Если пересекает, то мы игнорируем этот источник. См. рисунок 2.6.

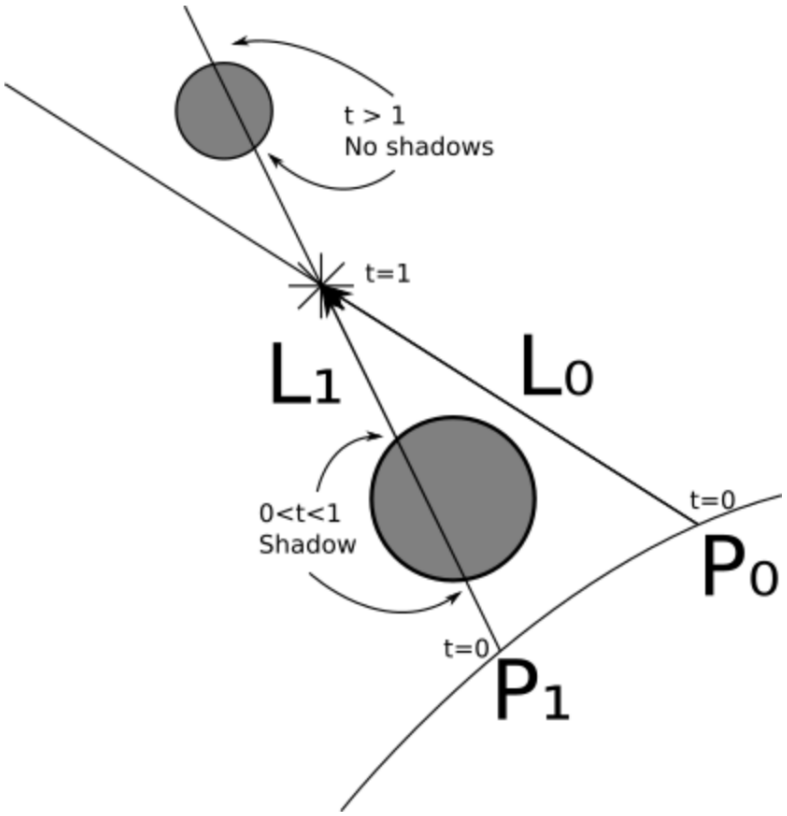

Рисунок 2.6 — Тени

Существует один пограничный случай, который нам нужно рассмотреть. Если искать пересечения, начиная с $t = 0$ то мы, вероятнее всего, найдём саму P, другими словами, каждый объект будет отбрасывать тени на самого себя.

Простейший способ справиться с этим — использовать в качестве нижней границы значений t вместо 0 использовать малое значение ϵ.

2.1.2 Моделирование взрыва

Взрыв частиц состоит из взаимодействия между каждой парой частиц. Приведенное ниже столкновение является описанием события 3D-столкновения и обеспечивает сохранение импульса и энергии.

Рассматриваются 2 частицы $m1$, $m2$ с радиусами $r1$, $r2$, центры которых находятся в точках с координатами ($x1$,$y1$,$z1$), ($x2$,$y2$,$z2$) и двигаются со скоростями ($vx1$,$vy1$,$vz1$), ($vx2$,$vy2$,$vz2$).

Алгоритм.

а) Вычислить относительное расстояние и относительную скорость.

б) Если расстояние между шарами больше суммы радиусов или если относительная скорость = 0 (частицы не взаимодействуют) $\rightarrow$ выход из подпрограммы.

в) Для моделирования необходимо перейти в систему координат, одна из осей которой направлена вдоль вектора скорости 2 шара (он покоится), а начало координат совпадает с центром первой частицы. См. рисунок 2.7.

Для этого необходимо сдвинуть систему координат так, чтобы шар 1 находился в начале координат, а затем изменить скорость 1 шара на относительную (так как 2 шар покоится в новой системе отсчета). Для того чтобы ось Z была направлена вдоль вектора v_2, необходимо сделать два поворота: первый – на угол φ вокруг оси Z, второй – на угол θ вокруг оси Y.

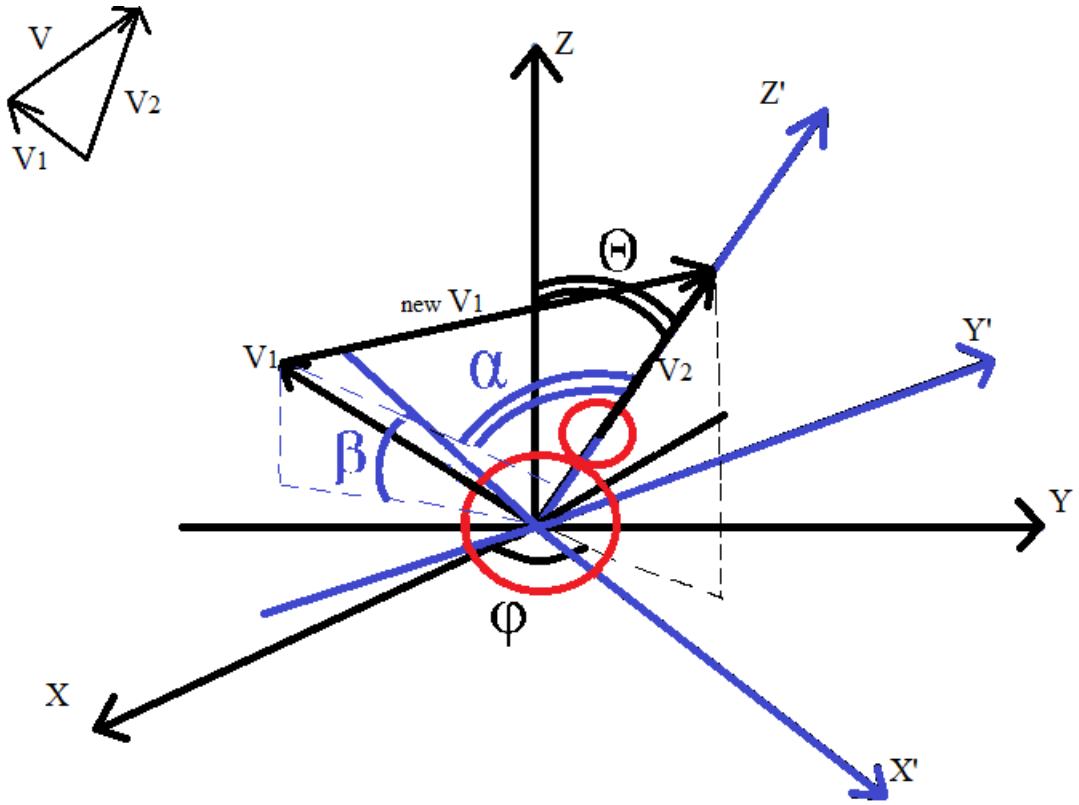

Рисунок 2.7

Матрица первого поворота

$$\begin{pmatrix} cos(\varphi) & sin(\varphi) & 0 \\ -sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

Матрица второго поворота

$$\begin{pmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{pmatrix} \tag{2.5}$$

Пересчет координат вектора при переходе от системы XYZ к системе X',Y',Z' дается произведением матриц 2.4, 2.5.

$$\begin{pmatrix} cos(\theta)cos(\varphi) & cos(\theta)sin(\varphi) & -sin(\theta) \\ -sin(\varphi) & cos(\varphi) & 0 \\ sin(\theta)cos(\varphi) & sin(\theta)sin(\varphi) & cos(\theta) \end{pmatrix} \tag{2.6}$$

Обратный переход от системы X',Y',Z' к системе XYZ производится с помощью транспонированной матрицы.

$$\begin{pmatrix} cos(\theta)cos(\varphi) & -sin(\varphi) & sin(\theta)cos(\varphi) \\ cos(\theta)sin(\varphi) & cos(\varphi) & sin(\theta)sin(\varphi) \\ -sin(\theta) & 0 & cos(\theta) \end{pmatrix} \tag{2.7}$$

г) Проверить сталкиваются ли шары (пересечение векторов скоростей) $\rightarrow$ если нет, выход из-под программы.

д) Нахождение новых скоростей, решением системы уравнений 2.8.

$$\begin{gathered} m_1 v_{1,x} = m_1 v'_{1,x} + m_2 v'_{2,z} tg(\alpha) cos(\beta) \\ m_1 v_{1,y} = m_1 v'_{1,y} + m_2 v'_{2,z} tg(\alpha) sin(\beta) \\ m_1 v_{1,z} = m_1 v'_{1,z} + m_2 v'_{2,z} \\ \frac{m_1(v_{x,1}^2+v_{y,1}^2+v_{z,1}^2)}{2} = \frac{m1(v_{x,1}'^2+v_{y,1}'^2+v_{z,1}'^2)}{2} + \frac{m2v_{z,2}'^2(1+tg^2(\alpha))}{2} \end{gathered} \tag{2.8}$$

е) Обновить скорости и повернуть векторы скорости назад и добавить начальный вектор скорости шара 2, чтобы получить исходную систему координат.

2.2 Разработка и обоснование используемых типов и структур данных

В данной работе происходит взаимодействие с моделью, структура которой представлена на рисунке 2.8. Состоит из вектора частиц (система частиц, в которой каждая частица обладает определенными параметрами), земля – плоскость, на которой происходит взаимодействие частиц и частица, рассматриваемая отдельно, в начальный момент имеющая скорость.

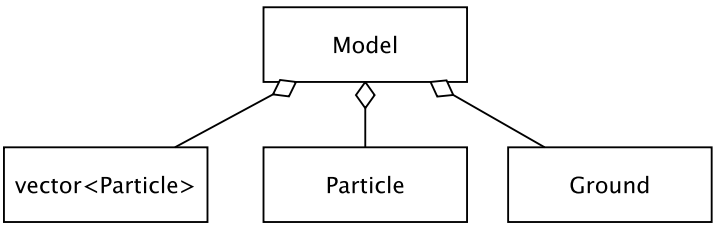

Рисунок 2.8 — Структура модели

Камера в данной работе содержит структуру матрицу, для удобства взаимодействия. В ней реализованы методы, умножения матрицы на матрицу, вектора на матрицу для работы с преобразования координат.

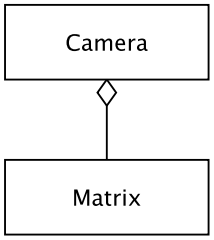

Рисунок 2.9 — Структура камеры

2.3 **Разработка структуры программного комплекса**

Алгоритм работы программы.

1. Загрузка модели из txt файла.

2. Установка первоначальных параметров для модели сцены и камеры.

3. Вычисление координат и положения объекта (для различных этапов взрыва).

4. Преобразование координат (повороты моделей, масштабирование и перенос) относительно камеры.

5. Генерация и отображение модели.

6. Считывание новых параметров.

7. Возвращение к п.3.

2.4 **Вывод**

Необходимо реализовать алгоритмы трассировки лучей для удаления невидимых поверхностей и работы с освещением (диффузное отражение и тени). Далее необходимо разработать алгоритм визуализации взрыва с опорой на физические законы, такие как закон сохранения энергии и закон сохранения импульса.

Необходимо также разработать взаимодействие с камерой и объектом (перемещение, масштабирование, поворот).

3 Технологический раздел

В соответствии с выбранной задачей – визуализацией взрыва. Необходимо выбрать средства реализации, создать интерфейс и структуры программного обеспечения, описать ограничения и порядок работы программы.

3.1 Выбор и обоснование языка программирования

Для выполнения проекта был выбран язык программирования C++/QT.

C++ – компилируемый, статически типизированный язык программирования общего назначения [6].

C++ сочетает свойства высокоуровневого и низкоуровневого языка. В сравнении с его предшественником, языком C, присутствует поддержка объектно-ориентированного программирования.

Являясь одним из самых популярных языков программирования, C++ широко используется для разработки программного обеспечения [7].

Qt – кроссплатформенная библиотека разработки GUI на C++ [8].

Библиотека Qt является безусловным лидером среди имеющихся средств разработки кроссплатформенных приложений на языке C++. Qt – полностью объектно-ориентированная библиотека.

Все элементы в окне программы, кнопки, переключатели - это отдельные виджеты. Именно виджеты являются основой построения графических интерфейсов. С точки зрения программы, они воспринимают все события и генерируют сигналы, которые в свою очередь вызывают слоты – специальные функции, реагирующие на различные события в окне.

Qt прекрасно документирована, благодаря чему всегда можно найти о ней любую интересующую информацию.

Для написания программного кода использовался Qt Creator 4.8.1. Основная задача Qt Creator – упростить разработку приложения с помощью фреймворка Qt на различных платформах.

3.2 **Интерфейс пользователя**

Взаимодействие пользователя с приложением осуществляется через интерфейс, представленный на рисунке 3.1.

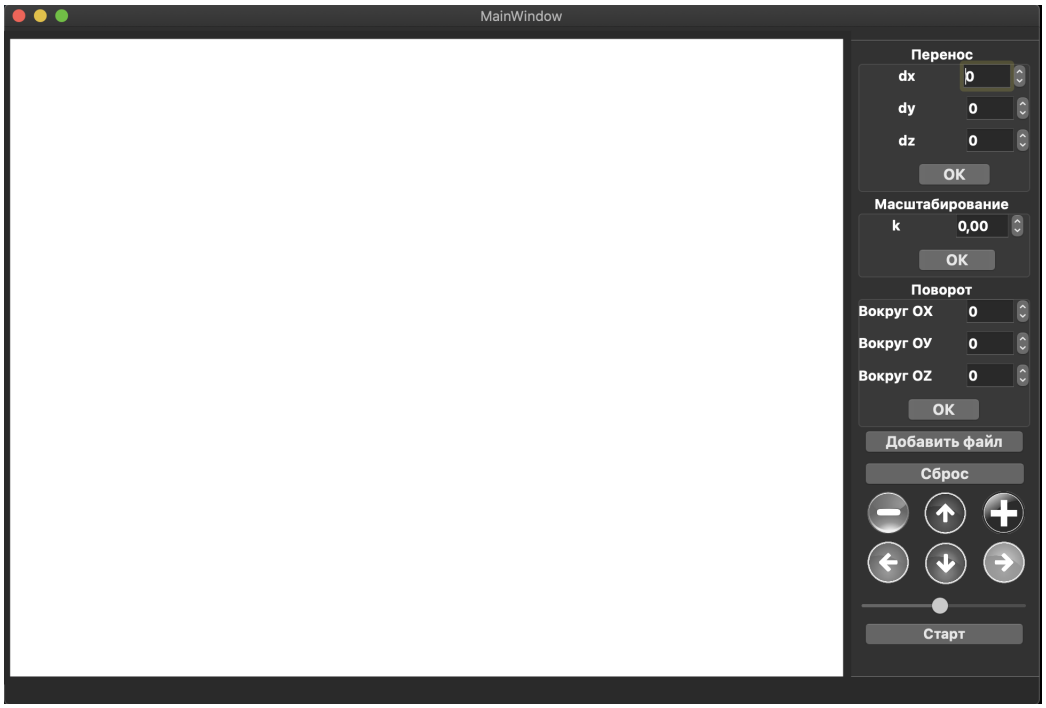

Рисунок 3.1 — Интерфейс пользователя

а) Доступны поля для ввода данных, при желании менять расположение и вид объектов.

б) Кнопки ($\rightarrow, \leftarrow, \uparrow, \downarrow, +, -$) предназначены для взаимодействия с камерой.

в) Кнопка «Сброс» – сбрасывает все настройки в начальное состояние.

г) Кнопка «Добавить файл» – для загрузки данных.

д) Слайдер – для задания скорости частицы в начале.

е) Кнопка «Старт» – запускает визуализацию взрыва.

3.3 Хранение и обмен данными в системе

Данные считываются из файла, содержащего в себе:

1-я строка – количество частиц

2 -я строка и последующие строки содержать в себе информацию о частицах: положение, радиус и вес (5 значений через пробел).

Последняя строчка рассматривается, как частица, которая в начальный момент получит скорость движения.

Из каждого файла считывается модель, состоящая из вектора частиц, позиции земли (плоскости ограничивающей взаимодействие). Класс частицы представлен в листинге 3.1.

Листинг 3.1 — Класс частицы

```
1  class Particle
2  {
3  public:
4      Particle() {}
5      Particle(Point_3d point, int r, int m);
6      Particle(Point_3d point, int r);
7      Particle(Point_3d point);
8      ~Particle() {}
9
10     void set_p(const Point_3d p);
11     Point_3d get_p() const;
12
13     void set_v(const Point_3d v);
14     Point_3d get_v() const;
15
16     void set_m(const double m);
17     double get_m() const;
18
19     void set_r(const int r);
20     int get_r() const;
21
22     void collision(Particle &p);
23     void update(int time);
24
25 private:
```

```
26      Point_3d point;
27      Point_3d speed;
28      int m;
29      int radius;
30  };
```

3.4 **Разработка программы**

В данной работе, используется объектно-ориентированный подход программирования и паттерны проектирования[9].

Для связание интерфейса и реализации используются паттерны фасад и команда. См. рисунки 3.2, 3.3.

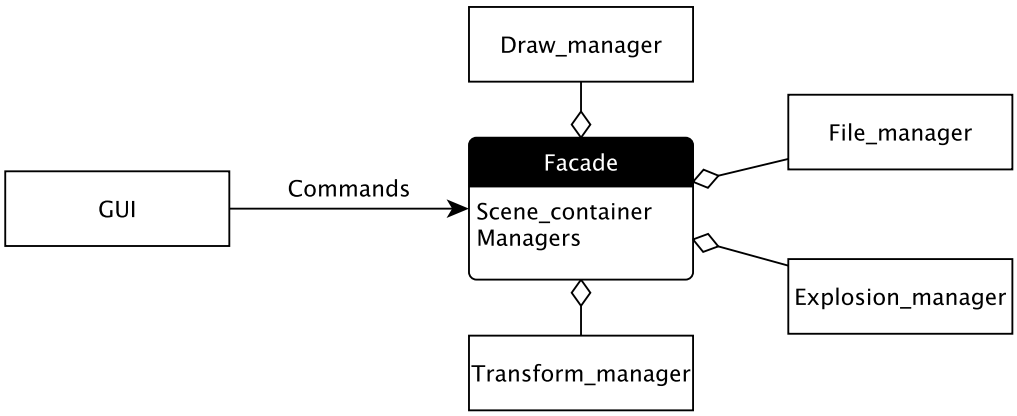

Рисунок 3.2 — Паттерн фасад

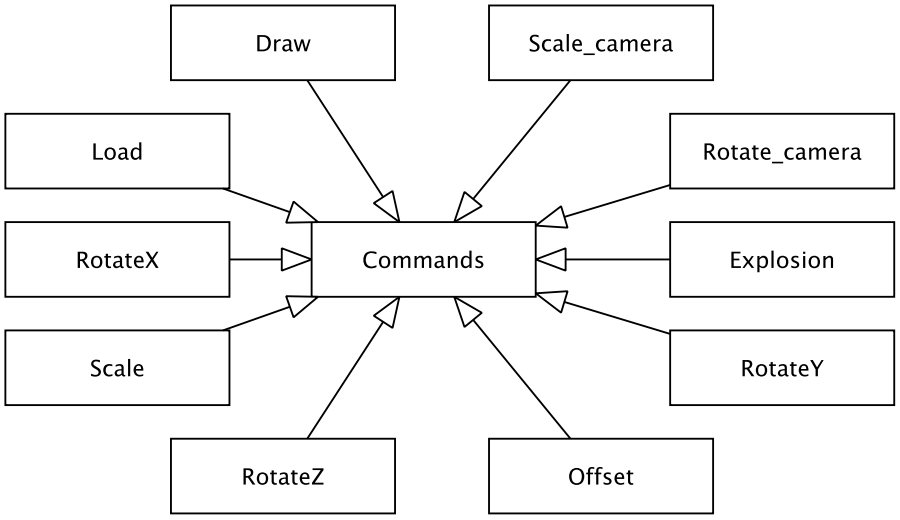

Рисунок 3.3 — Паттер команда

Структура сцены в данной работе представлена на рисунке 3.4.

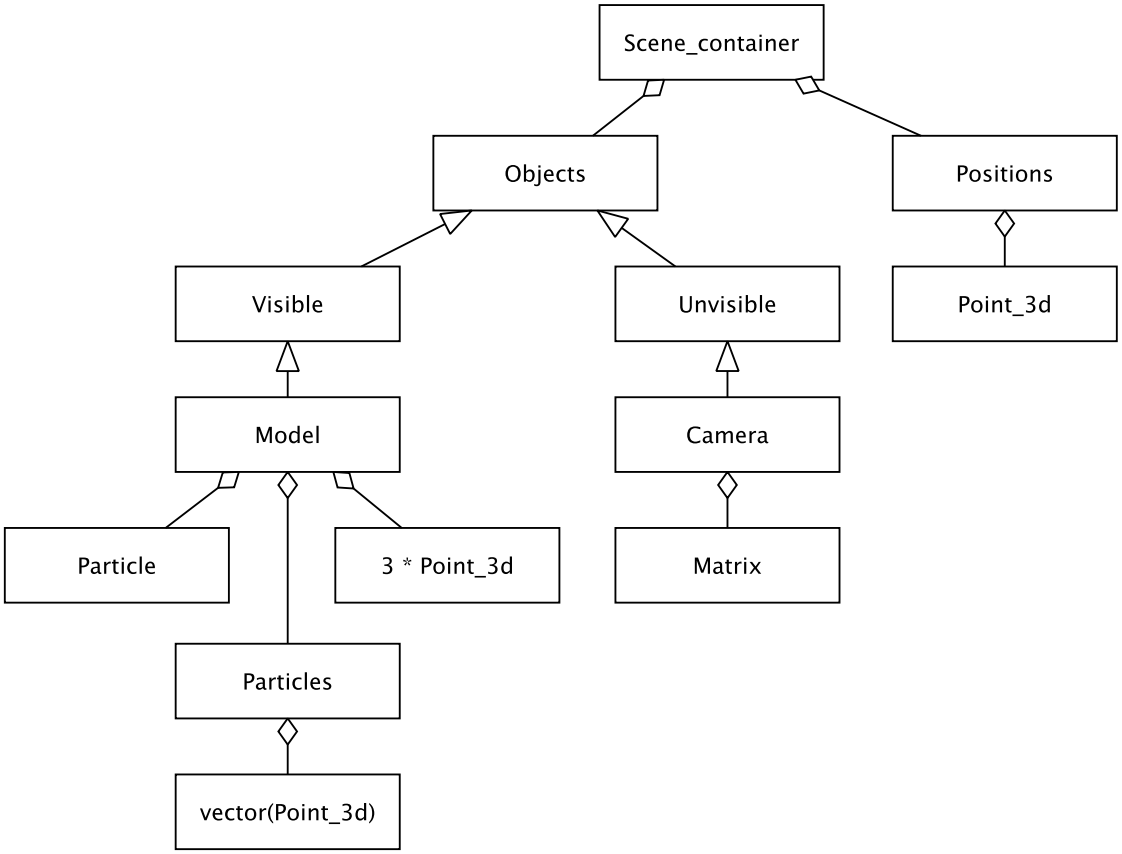

Рисунок 3.4 — Контейнер сцены

Для создания сложной модели используется паттерн строитель. Для последовательного обхода объектов на сцене – паттерн итератор.

3.5 **Требования к аппаратуре**

Так как в данной работе используется трассировка лучей, в которой необходимо пробегаться по всем пикселям экрана, стоит необходимость уменьшения временных затрат на вычисления.

Таким образом, существует необходимость использования параллельных вычислений. Так как параметры каждого пикселя экрана вычисляются независимо от других, то для того, чтобы вычислить все параллельно, достаточно просто указать какие пиксели какому потоку вычислять.

Теоретически, для нахождения значения каждого из пикслей можно создать свой поток. Но ОС не позволит создать такое число потоков, ресурсов на их создание не хватит и создание потока также занимает определенный промежуток времени.

Алгоритм.

а) Разделить экран по оси X на равные участки N.

б) Создать N потоков с указанием вычисляемых пикселей.

в) Запустить потоки.

Проведено исследование с замером времени необходимого на отрисовку экрана для разного количества потоков. Количество повторов эксперимента – 15.

Количество потоков	1	2	4	8	16
Среднее время визуализации экрана, микросекунды	11518775.92	6013675	3298333.53	2843617.94	2886129.94

Как видно из эксперимента, наиболее быстрая отрисовка при количестве потоков равным 8, что равно количеству логических ядер процессора.

Следовательно, для наиболее быстрой работе, количество логических ядер процессора на персональном компьютере должно быть больше 8.

3.6 **Требования к программному обеспечению**

С помощью Qt можно создавать графические приложения на различных операционных системах, не переписывая исходный код. Qt поддерживается на различных 32-битных и 64-битных платформах.

Qt – кроссплатформенная библиотека разработки GUI на C++, а значит нет привязки к платформе, возможно использование любого ПО.

Qt Creator доступен для следующих операционных систем:

1. Windows 7 или более поздняя версия
2. Ubuntu Linux 16.04 (64-разрядная версия) или более поздняя версия
3. macOS 10.12 или более поздней версии

3.7 **Порядок работы**

Алгоритм программы.

1. Визуализация начального экрана и интерфейса.
2. Ввод и чтение из пользовательского файла.
3. Если ошибок не возникло, загрузка новой модели.
4. Визуализация модели с учетом матрицы камеры.

   1) Алгоритм трассировки для каждого пикселя экрана.

   2) Определение затененности для каждого пикселя экрана.

   3) Применение диффузного отражения для каждого пикселя экрана.

   4) Отрисовка пикселей.
5. При нажатой кнопке взрыва.

   1) Пересчет скорости каждого объекта по законам физики.

   2) Пересчет положения каждого объекта с учетом скорости.
6. Переход к пункту 4, если не загружен новый файл, иначе пункт 2.

3.8 **Обращение к программе**

Программа компилируется и запускается с помощью Qt Creator.

Взаимодействие пользователя с приложением осуществляется через интерфейс, представленный на рисунке 3.1.

3.9 Входные и выходные данные

На вход поступают данные из файла, определенного вида. Также можно изменять данные с помощью графического интерфейса или менять направление камеры.

На выходе получаем визуализацию столкновения частиц.

3.10 Сообщения системы

Система уведомляет пользователя в случае некорректных данных.

а) Отсутствует файл при загрузке.

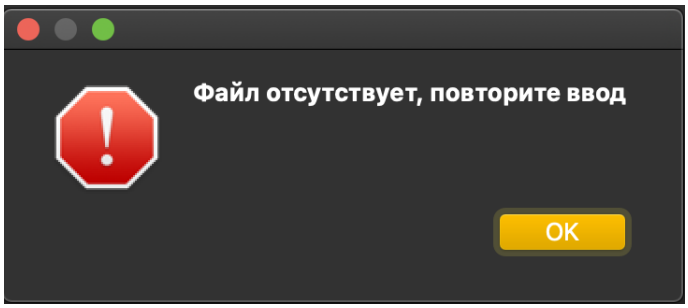

Рисунок 3.5 — Отсутствует файл

б) Некорректный файл (данные в нем).

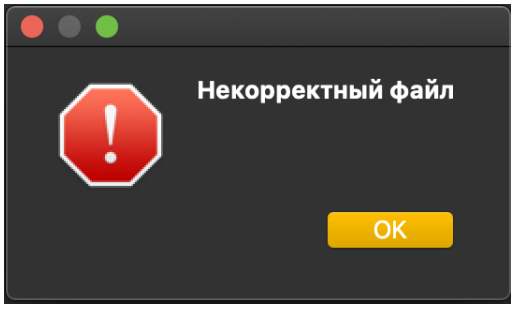

Рисунок 3.6 — Некорректные данные в файле

в) Некорректные значения в полях ввода интерфейса.

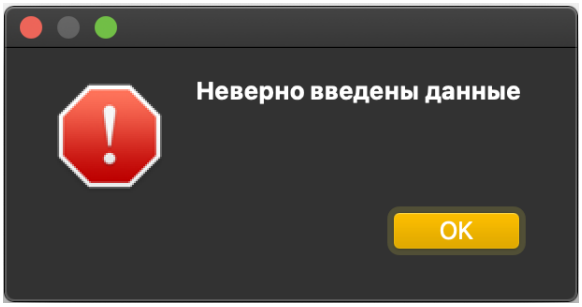

Рисунок 3.7 — Некорректные вводимые данные

3.11 Вывод

В данном разделе подготовлен и отлажен исходный код приложения, приведены этапы работы приложения, способы хранения данных. Представлены и протестированы различные случаи работы приложения, как корректные так и некорректные.

4 Экспериментальный раздел

Проводится анализ реализованного приложения, проверяется корректность работы, строятся графики зависимости времени генерации картинки от количества частиц и количества потоков.

4.1 Исследование характеристик программы

В работе для ускорения вычислений используется параллельной программирование. Необходимо исследовать оптимальное число потоков необходимых для наиболее быстрой генерации изображения.

Из графика, представленного на рисунке 4.1 видно, что при генерации частиц, количество которых не превышает 50, использование 4 поток наиболее эффективно, например на 15% чем 8 потоков и в 4 раза чем при использовании 1 потока.

Однако при росте числа частиц, ситуация меняется. Использование 8 и 16 потоков становится эффективнее, причем 8 потоков быстрее чем 16 на 3%.

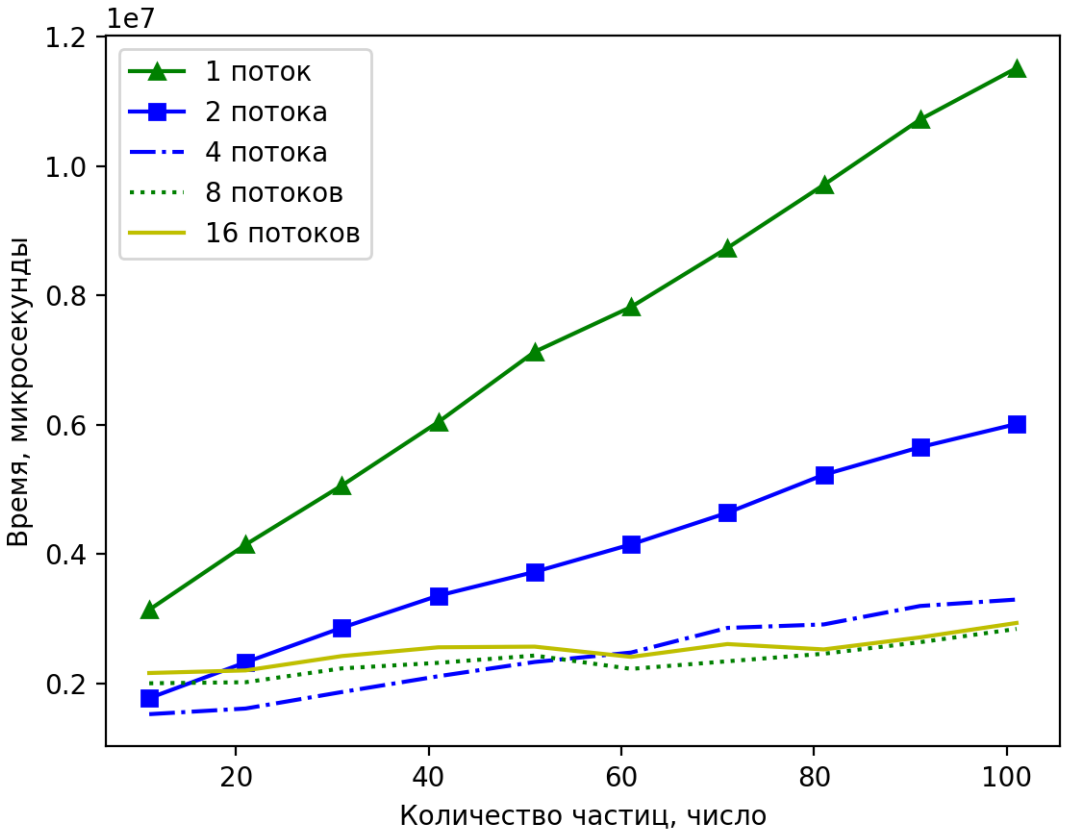

Рисунок 4.1 — График зависимости времени генерации изображения от количества частиц для различного числа потоков

Стоит заметить, что 8 потоков соответствует количеству логических ядер процессора.

Таким образом, получается что количество потоков будет на прямую зависеть от параметров компьютера, на котором запущена визуализация.

4.2 **Примеры использования программы**

Примеры работы программы представлены на рисунках 4.2, 4.3. Визуализация одного и того же процесса в разные моменты времени.

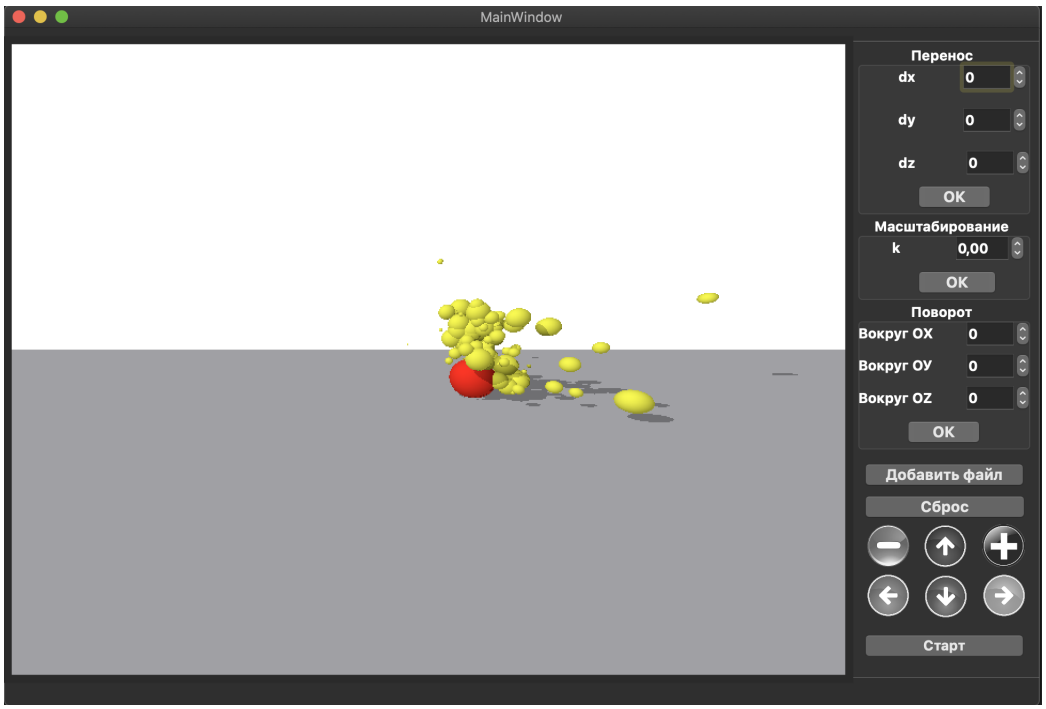

Рисунок 4.2 — Пример 1

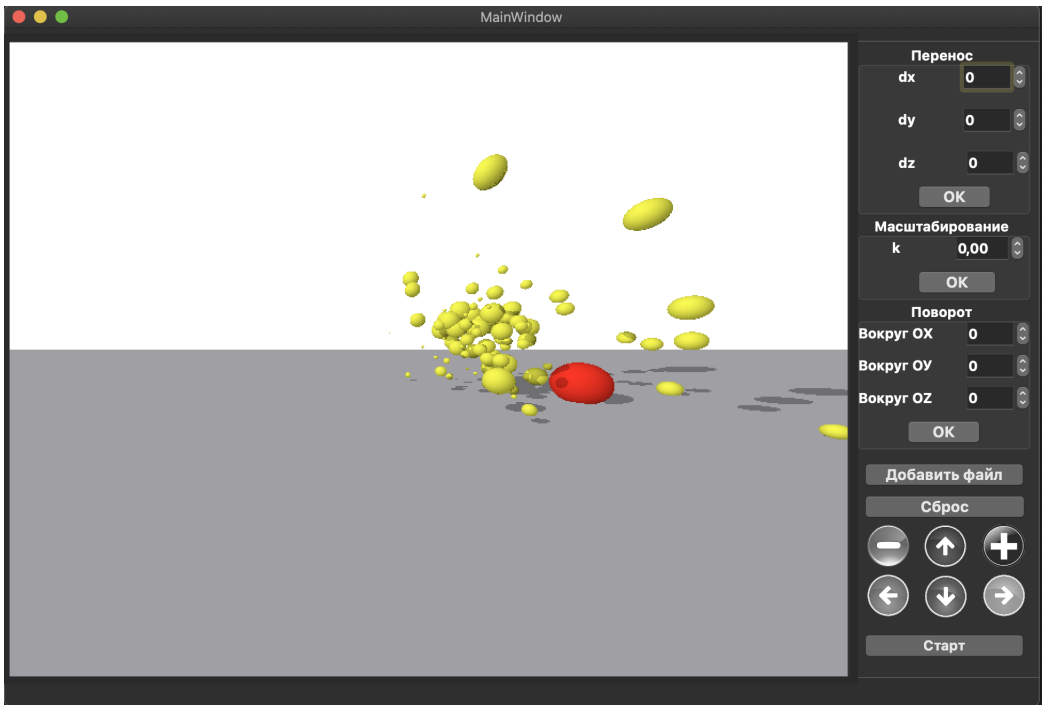

Рисунок 4.3 — Пример 2

4.3 Вывод

Проведено исследование, зависимости количества частиц от времени визуализации для различного числа потоков. Получены выводы,

что оптимальное число используемых потоков, при количестве частиц большем 50, равно количеству логических ядер процессора. А так как, в данной работе ставится задача визуализации взаимодействия большого числа частиц, то использовано 8 потоков.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Взрывы в кино: что за кадром? — Режим доступа: http://www.mir3d.ru/vfx/950/ (дата обращения: 10.10.2019).

2. Простая система частиц? — Режим доступа: http://grafika.me/node/496 (дата обращения: 16.11.2019).

3. Reeves William T. Particle Systems – A Technique for Modeling a Class of Fuzzy Objects. — Computer Graphics, том 17, №3, с. 359-376, 1983.

4. И. Е. Жигалов И. А. Новиков. Программирование трехмерной компьютерной графики. — Изд-во ВлГУ, Владимир, 2016.

5. Удаление скрытых линий и поверхностей. — Режим доступа: http://algolist.ru/graphics/delinvis.php (дата обращения: 16.11.2019).

6. C++ documentation. — Режим доступа: https://en.cppreference.com/w/ (дата обращения: 27.11.2019).

7. C++ в современном мире. — Режим доступа: https://habr.com/ru/company/pvs-studio/blog/259777/ (дата обращения: 27.11.2019).

8. Qt documentation. — Режим доступа: https://doc.qt.io (дата обращения: 27.11.2019).

9. Паттерны. — Режим доступа: https://refactoring.guru/ru/design-patterns (дата обращения: 29.11.2019).

ПРИЛОЖЕНИЕ А

КАРТИНКИ