Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Гурьев Василий Александрович

Реализация алгоритма Ray Marching

Курсовая работа

Научный руководитель:
к.т.н., доц Литвинов Ю.В.

Санкт-Петербург
2018

Оглавление

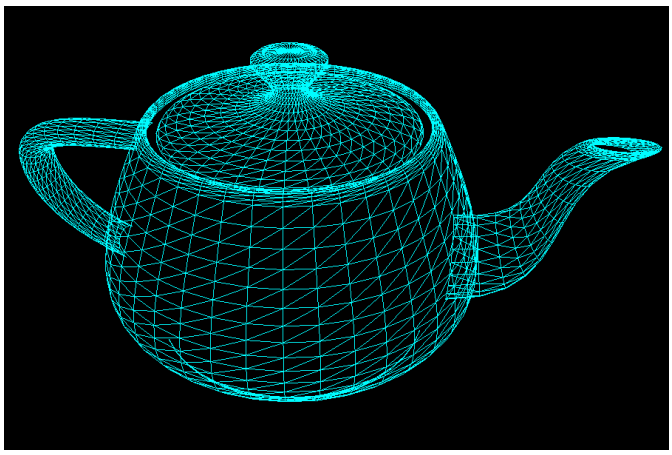

Рис. 1: Пример объекта из треугольников

Введение

Компьютерная графика — достаточно важный аспект современного мира. У нее много применений [3] и одно из них — получение произвольных реалистичных 3D изображений. Ведь если мы обладаем такой возможностью, то мы можем и строить реалистичные видео, а так же решаем и множество других задач. В данный момент существует 2 основных алгоритма генерации реалистичных 3D изображений. Первый, самый популярный, использущейся во всех компьютерных играх, основывается на представлении объектов на сцене как множества треугольников (рис. 1), с которым в последствии работает видеокарта, преобразуя его в изображение. Второй алгоритм, Ray Tracing, заключается в том, что мы для каждого пикселя изображения пускаем луч в сторону сцены и пытаемся найти пересечение со всеми объектами на сцене. Подробнее можно почитать тут [2]. Однако и у одного и у другого метода достаточно много недостатков, у Ray Tracing — очень большое время получения картинки, у метода с треугольниками — высокая сложность рисования 3D-моделей. Поэтому было решено исследовать и реализовать сравнительно новый и не слишком популярный алгоритм, Ray Marching, появившийся около 8 лет назад.

1. Постановка задачи

Итого было поставлено несколько задач:

1) Исследование алгоритма Ray Marching

2) Реализовать данный алгоритм, а так же реализовать библиотеку с различными функциями Ray Marching-а 3) Реализовать IDE для сцен Ray Marching

4) Выполнить оптимизацию для выполнения сцен в реальном времени.

2. Алгоритм

Алгоритм Ray Marching в целом напоминает алгоритм Ray Tracing: из каждой точки изображения в сторону сцены пускается луч, с помощью которого мы ищем пересечение с объектами, находящимися на сцене. Однако при использовании Ray Tracing объекты в сцене заданы аналитически, причем пересечение ищется тоже аналитически. Например для нахождения пересечения луча со сферой необходимо решить следующую систему уравнений (1).

$$\begin{aligned} x^2 + y^2 + z^2 &= R^2 \\ (x - x_0) * x_1 &= (y - y_0) * y_1 = (z - z_0) * z_1 \end{aligned} \tag{1}$$

При использовании же алгоритма Ray Marching объекты в сцене заданы как функция, для любой точки возвращающая наименьшее расстояние до этого объекта. Например объект сферы, расположенной в точке (0, 0, 0) будет задан функцией (2).

$$\begin{aligned} &float\ Sphere(vec3\ position,\ float\ radius) \\ &\{ \\ &\quad return\ length(position) - radios; \\ &\} \end{aligned} \tag{2}$$

Выглядит проще, чем искать пересечение аналитически, верно? Но что же мы делаем с этой функцией далее? Все очень просто — с ее помощью мы итерируемся по нашему лучу, пущенному внутрь сцены. То есть более простым языком — мы вызываем ее сначала от самого начала луча, потом отходим в направлении нашего луча на значение функции и снова ее вызываем, продолжаем процесс до тех пор, пока не найдем пересечение, или не выйдем за пределы нашей сцены. Более наглядно это можно посмотреть на изображении (рис. 2)

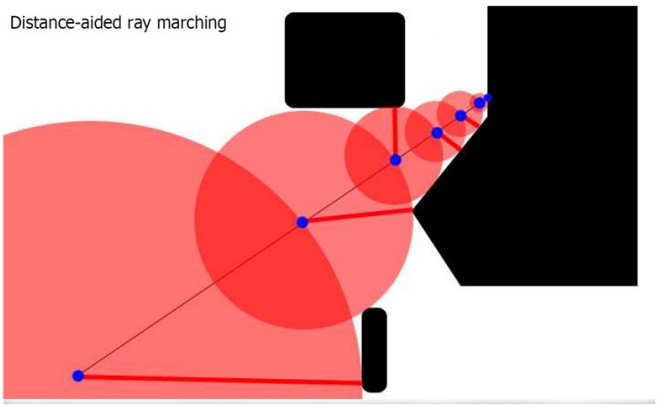

Рис. 2: Трассировка по лучу внутри сцены

3. Некоторые примеры задания объектов

В предыдущем пункте была рассмотрена функция, задающая сферу (2). Теперь рассмотрим как пример функцию, задающую тор, расположенный в точке (0, 0, 0) (3). Она выглядит посложнее сферы, но тоже ничего сложного в ней нет.

$$
\begin{aligned}
&float\ Torus(vec3\ pos,\ vec2\ rad)\\
&\{\\
&\quad return\ length(vec2(length(pos.xz) - rad.x,\ pos.y)) - rad.y;\\
&\}
\end{aligned}
\tag{3}
$$

Но функция из одного объекта это лишь самый простой вариант задания сцены. Если мы хотим, чтобы внутри сцены было несколько объектов, мы можем применить функции для каждого из объектов, а затем взять из результатов минимум. Такими же очевидними преобразованиями мы можем брать лишь пересечение 2 объектов, их разность и т.п. На следующем изображении (рис. 3) — пример дома, построенного мной при помощи алгоритма ray marching, но не в рамках данной работы и достаточно давно. Чтобы добиться такого эффекта, я взял несколько вытянутых кубов и сделал из них нечто вроде завернутого тоннеля. Крыша — те же приплюснутые повернутые кубы. Чтобы сделать окна, пришлось из куба, играющего роль стены, каждые несколько «метров» вычитать другой куб и сферу. И дом — не самое сложное, что

можно построить таким образом!

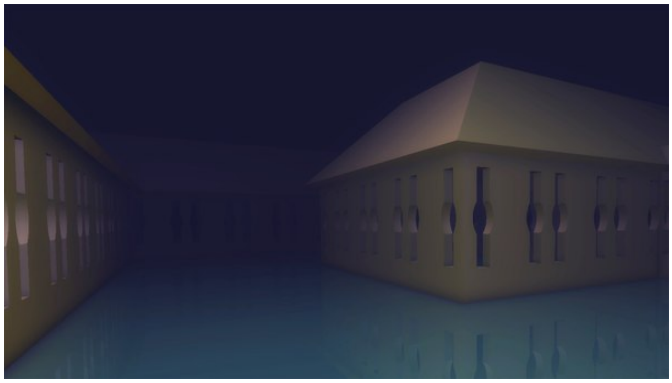

Рис. 3: Объект «дом», полученный различными операциями над простыми объектами

4. «Обычные» эффекты

На построенном мной изображении (рис. 3) вы могли заметить, что у объекта присутствовал цвет, во всей сцене было освещение, отражение и тени. В алгоритме ray marching когда было найдено пересечение с объектом в сцене мы можем возвращать не просто цвет объекта, с которым мы пересеклись, а мы можем посчитать нормаль в точке пересечения и, например с помощью модели освещения Фонга ([1]), вычислить освещенность в данной точке учитывая источники освещения. Так же зная нормаль нам не составит труда сделать мягкие тени, преломления, отражения и все те эффекты, которые применяются в алгоритме ray tracing. А значит при должном усилии наши сцены могут быть гораздо лучше, чем например эта сцена, построенная при помощи ray tracing. (рис. 4)

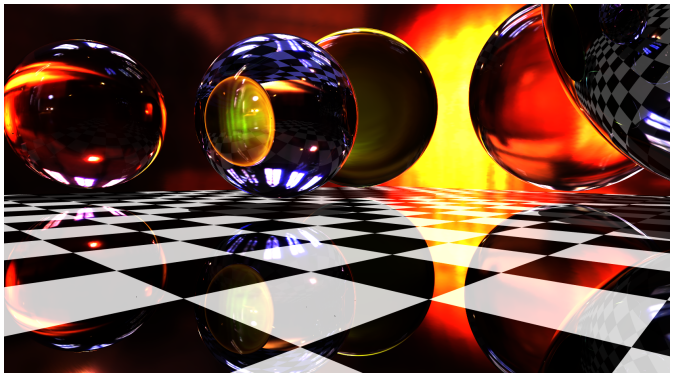

Рис. 4: Сцена ray tracing с различными эффектами

5. «Специальные» эффекты

Помимо обычных и привычных глазу эффектов освещения, теней и прочего, ray marching позволяет делать очень многое, что остальными алгоритмами получения изображений сделать очень сложно, или даже вовсе невозможно. Я приведу лишь несколько примеров, сделанных на реализованной мною IDE (о ней немного ниже). Первый пример — репликация (4).

$$
\begin{aligned}
&vec3\ opRep(vec3\ p,\ vec3\ c)\\
&\{\\
&\quad return\ mod(p,\ c) - 0.5 * c;\\
&\}
\end{aligned}
\tag{4}
$$

Как несложно увидеть из кода — данная функция делит все пространство на параллелепипеды с диагональю «c» и располагает наш объект в середине каждого из этих параллелепипедов. Причем она это делает всего лишь за одну операцию деления по модулю! В любом другом алгоритме сложность подсчетов увеличилось бы многократно, соответственно вместе с этим производительность многократно бы уменьшилась. Пример действия данной функции в IDE можно посмотреть на изображении (рис. 5)

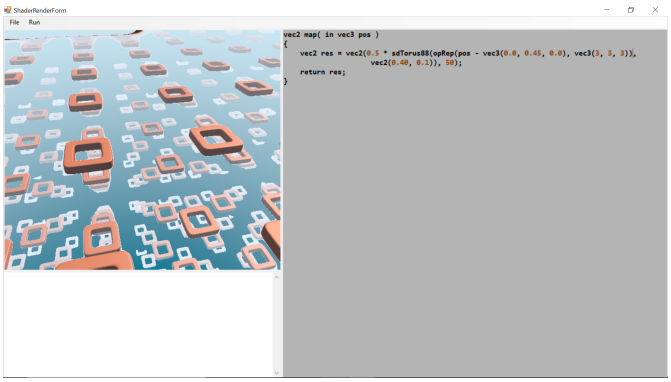

Рис. 5: Результат действия функции «opRep»

Следующий пример — поворот всего объекта по спирали. Код данной функции можно посмотреть на Git, файл «RayMarchBegin.glsl». В данном документе я его приводить не буду, чтобы излишне не загру-

жать. Пример действия (рис. 6)

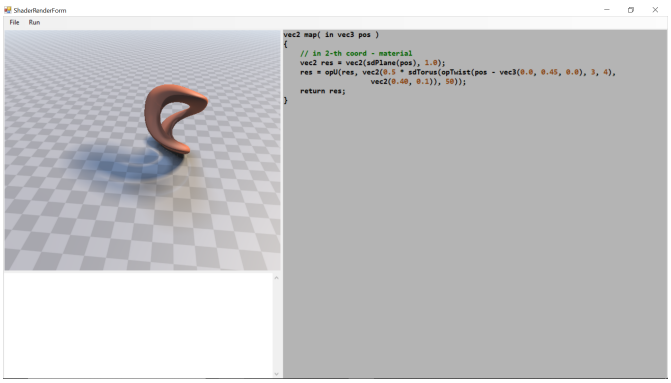

Рис. 6: Результат действия функции «opTwist», примененной к тору.

6. IDE

Мною была реализована IDE для написания своих примеров работы алгоритма RayMarching, а так же небольшая библиотека различных функций. Я добавил поддержку освещения, мягких теней и вращения камеры вокруг сцены. Чтобы написать свой пример, необходимо лишь написать функцию map — функцию, для каждой точки возвращающую наименьшее расстояние до сцены. Список всех реализованных функций можно посмотреть на Git в файле «RayMarchBegin.glsl». В самой IDE реализована подсветка синтаксиса, вывод ошибок в специальном окне, если что-то написано неверно, а так же окно с результатом. Так же для быстродействия данного алгоритма, чтобы сцены могли изменяться в реальном времени, все расчеты выполняются параллельно на графическом процессоре с помощью пиксельных шейдеров [4]. Примеры работы IDE можно посмотреть выше (рис. 6), (рис.5). По технической части — все окна реализованы на C#, библиотека функций и сам алгоритм реализованы на языке glsl, специальном C-подобном шейдерном языке, исполняемом на графическом процессоре.

Заключение

Итого было сделано:
1) Исследован алгоритм Ray Marching
2) Реализован данный алгоритм, а так же реализована библиотека с различными функциями Ray Marching-а 3) Реализована IDE для сцен Ray Marching
4) Выполнена оптимизация для выполнения сцен в реальном времени.

Проект находится тут — «https://github.com/chudovas/RayMarchIDE».

Список литературы

[1] Wikipedia. Затенение по Фонгу // Википедия, свободная энциклопедия. — 2016. — URL: https://ru.wikipedia.org/wiki/Затенение_по_Фонгу (дата обращения: 29.05.2018).

[2] Wikipedia. Ray Tracing // Википедия, свободная энциклопедия. — 2018. — URL: https://ru.wikipedia.org/wiki/Трассировка_лучей (дата обращения: 29.05.2018).

[3] Wikipedia. Компьютерная графика // Википедия, свободная энциклопедия. — 2018. — URL: https://ru.wikipedia.org/wiki/Компьютерная_графика#Основные_области_применения (дата обращения: 29.05.2018).

[4] Wikipedia. Шейдера // Википедия, свободная энциклопедия. — 2018. — URL: https://ru.wikipedia.org/wiki/Шейдер (дата обращения: 29.05.2018).