

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

Введение	3
1. Постановка задачи	4
2. Алгоритм	5
3. Некоторые примеры задания объектов	6
4. «Обычные» эффекты	7
5. «Специальные» эффекты	8
6. IDE	9
Заключение	10
Список литературы	11

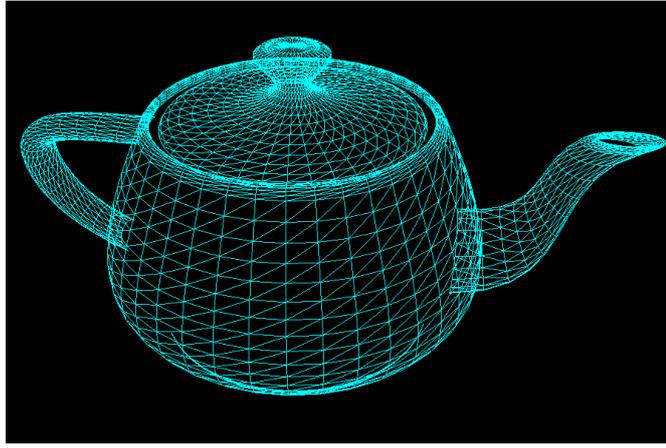


Рис. 1: Пример объекта из треугольников

## Введение

Компьютерная графика — достаточно важный аспект современного мира. У нее много применений [3] и одно из них — получение произвольных реалистичных 3D изображений. Обладая решением данной задачи, так же решается и целый класс задач, посвященных генерации реалистичного видеоизображения и т.п.

В данный момент существует 2 основных алгоритма генерации реалистичных 3D изображений. Первый, самый популярный, использующийся 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. Вторым алгоритмом, Ray Tracing, заключается в том, что для каждого пикселя изображения пускается луч в сторону сцены и ищется пересечение со всеми объектами на сцене. Подробнее см. [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-a;
- 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 ищется 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 объект в сцене задан как функция, для любой точки возвращающая наименьшее расстояние до этого объекта. Например объект сферы, расположенной в точке (0, 0, 0) будет задан функцией (2).

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

Далее при помощи заданной функции идет итерация по лучу, пущенному внутрь сцены. Более простым языком — она вызывается сначала от самого начала луча, потом от точки, расположенной на расстоянии значения функции от начала луча по его направлению. Процесс продолжается до тех пор, пока не будет найдено пересечение, или точка не выйдет за пределы сцены. Более наглядно это можно посмотреть на изображении (рис. 2)

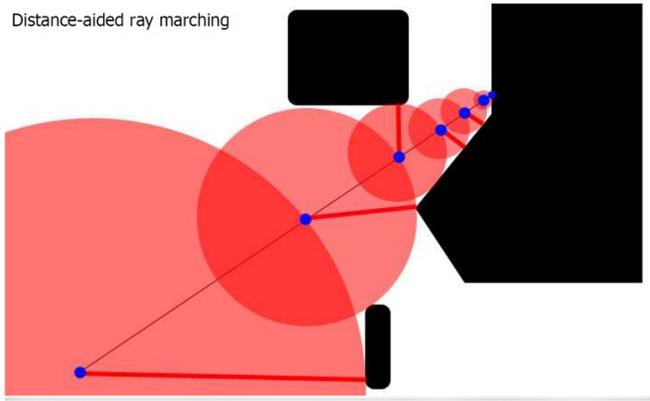


Рис. 2: Трассировка по лучу внутри сцены

### 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).

```
float Torus(vec3 pos, vec2 rad)
{
    return length(vec2(length(pos.xz) - rad.x, pos.y)) - rad.y;
}
```

(3)

Но функция из одного объекта это только самый простейший вариант задания сцены. Чтобы внутри сцены было несколько объектов, может быть применена функция для каждого из объектов, а затем взят минимум из результатов. Такими же очевидными преобразованиями можно взять лишь пересечение 2 объектов, их разность и т.п. На следующем изображении (рис. 3) — пример дома, построенного при помощи алгоритма ray marching. Чтобы добиться такого эффекта, было взято несколько вытянутых кубов и сделано из них нечто вроде завернутого тоннеля. Крыша — те же приплюснутые повернутые кубы. Для представления окон из куба, играющего роль стены, каждые несколько «метров» были вычтены другой куб и сфера.

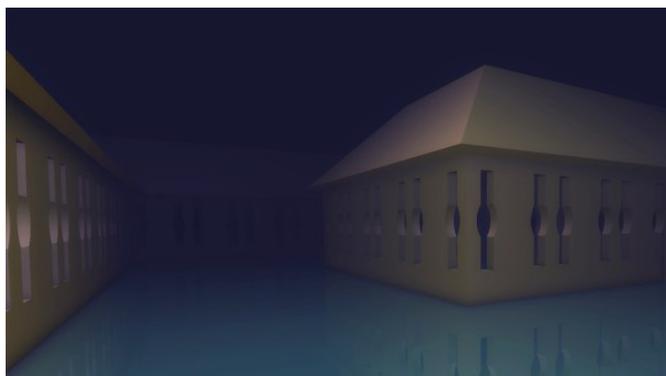


Рис. 3: Объект «дом», полученный различными операциями над простыми объектами

#### 4. «Обычные» эффекты

На 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 можно возвращать не просто цвет объекта, с которым было найдено пересечение. Можно 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, можно сделать мягкие тени, преломления, отражения и все те эффекты, которые применяются в алгоритме ray tracing. А значит при помощи алгоритма Ray Marching может быть построена например данная сцена, построенная при помощи ray tracing. (рис. 4)



Рис. 4: Сцена ray tracing с различными эффектами

## 5. «Специальные» эффекты

Помимо привычных глазу эффектов освещения, теней и прочего, ray marching позволяет делать многое, что остальными алгоритмами получения изображений сделать сложно, или даже вовсе невозможно. Будет приведено несколько примеров, сделанных при помощи реализованной IDE. Первый пример — репликация (4).

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

Данная функция делит все пространство на параллелепипеды с диагональю «с» и располагает объект в середине каждого из этих параллелепипедов. Причем делается это 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)

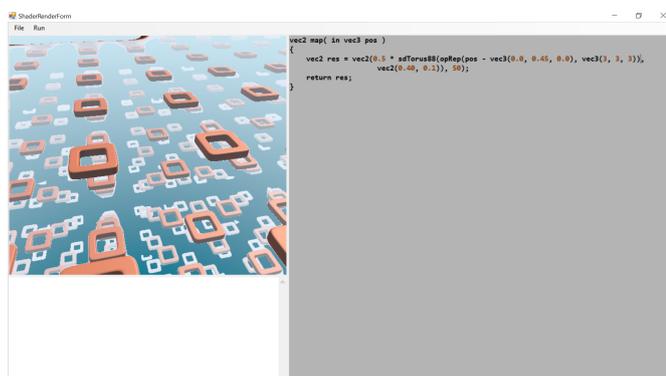


Рис. 5: Результат действия функции «opRep»

Следующий пример — поворот всего объекта по спирали. Код данной функции можно посмотреть на [Git](#), файл «RayMarchBegin.glsl». Пример действия (рис. 6)

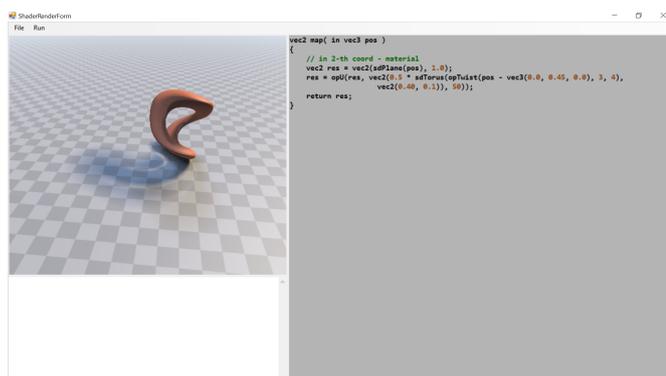


Рис. 6: Результат действия функции «opTwist», примененной к тору.

## 6. IDE

Была реализована IDE для написания своих примеров работы алгоритма RayMarching, а так же небольшая библиотека различных функций. Внутри IDE добавлена поддержка освещения, мягких теней и вращения камеры вокруг сцены. Чтобы написать свой пример, необходимо 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 быстрого действия 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
- 3) Реализована IDE для сцен Ray Marching
- 4) Выполнена оптимизация для выполнения сцен в реальном времени.

Проект можно посмотреть в данной репозитории —  
«<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](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](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](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).