Лицей НИУ ВШЭ

Рендеринг трехмерных объектов методом ray marching

Ученик группы 9ФМ Чубий Савва Андреевич

Москва, 2020

1 Цель использования метода

Данный метод используется для получения двумерной проекции трехмерных объектов на плоскоси с последующим отображением плоскости на экране.

2 Signed Distance Function

Для использования ray marching-а необходимо знать *sdf* для объектов сцены. *sdf* (Signed Distane Function, знаковая функция расстояния) — функция, возвращающая минимальное расстояние от данной точки в пространстве до поверхности объекта, который требуется изобразить.

- $sdf(P) > 0$, если точка P расположена вне объекта.
- $sdf(P) = 0$, если P лежит на поверхности.
- $sdf(P) < 0$, если P находится внутри объекта.

Для большего понимания выведем формулу $sdf(P)$ для шара.

1. Пусть центр шара расположен в точке C, а его радиус равен r.
2. Минимальное расстояние от P до поверхности шара равно длине перпендикуляра, опущенного из P на поверхность шара. Назовем этот перпендикуляр PH
3. $CP = PH + CH \;\Rightarrow\; sdf(P) = CP - PH = |P - C| - r \;\Rightarrow\; sdf(f) = |P - C| - r$

Такую функцию можно найти не только для шара, но и для куба, конуса, капсулы, тора, цилиндра и некоторых других фигур. Также *sdf* можно преобразовывать: округлить края, отразить или выгнуть фигуру, повторить её бесконечное количество раз и т.д.

3 Принцип работы

Нам дана точка C (от Camera) — расположение камеры, и единичные векторы $\vec{f}$ (от front), $\vec{r}$ (от right), $\vec{u}$ (от up), указывающие вперед, вправо и вверх соответственно, относительно направления камеры. ($\vec{f} \perp \vec{r}, \vec{r} \perp \vec{u}, \vec{u} \perp \vec{f}$)

Перед камерой расположенная плоскость, на которой отмечен прямоугольник. Плоскость и $\vec{f}$ перпендикулярны, две смежные стороны прямоугольника параллельны $\vec{u}$ и $\vec{r}$ соответственно, а луч сонаправленный $\vec{f}$, начинающийся в C, пересекает прямоугольник в его центре. Этот прямоугольник как бы является нашим монитором.

Теперь мы хотим получить цвет каждого пикселя. Для этого пропустим через точку на прямоугольнике, соответствующую данному пикселю луч, начинающийся в C. Луч будет сонаправлен вектору $uv_x \cdot \vec{r} + uv_y \cdot \vec{u} + \vec{f}$, где $\vec{uv}$ — двумерный вектор, описывающий координату пикселя. ($\vec{uv} = (0, 0)$ в центре экрана.) Если луч попал в какой-либо объект, красим пиксель в цвет этого объекта, иначе пиксель будет цвета заднего фона.

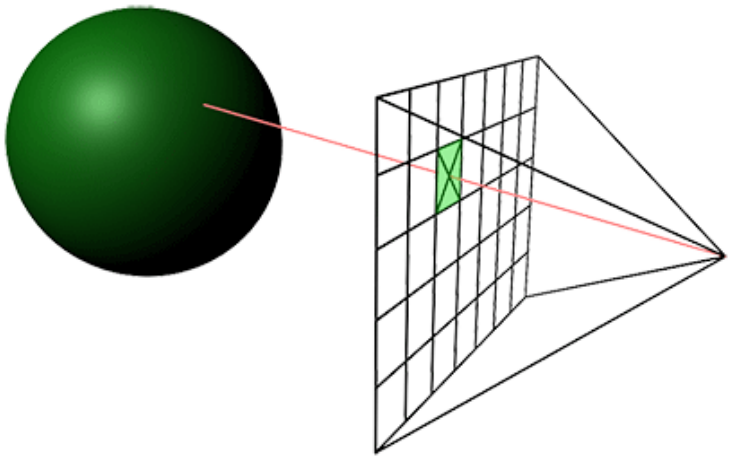

Рис. 1: Получение цвета пикселя

Осталось понять, каким образом проверяется пересечение луча и объекта, заданного sdf. Важно заметить, что именно эта часть отличает ray marching от похожих методов, например, ray tracing-а.

1. Рассматривается луч, заданный точкой начала $ro = C$ и направлением $\vec{rd} = normalize(uv_x \cdot \vec{r} + uv_y \cdot \vec{u} + \vec{f})$.

2. Работа алгоритма состоит из нескольких итераций, количество которых задается заранее. Чем больше итераций — тем лучше изображение и медленнее программа.

3. Будем "шагать" по лучу, пока не доберемся до объекта. Один шаг — одна итерация.

4. Пусть $t_0 = 0$, где t_i — расстояние, пройденное по лучу к i-ой итерации.

5. Рассмотрим i-ую итерацию.

   5.1. Найдем точку, в которой мы находимся: $P_i = \vec{ro} + t_i \cdot \vec{rd}$

   5.2. Вычислим минимальное расстояние от неё до объектов сцены $h_i = sdf\ (P_i)$

   5.3. Если это расстояние меньше некоторого порога, то считаем, что добрались до объекта и завершаем алгоритм. Порог задается заранее. Чем он меньше, тем лучше изображение.

   5.4. Если мы продвинулись по лучу дальше некоторого расстояния, выбранного заранее, говорим, что ни с чем не пересеклись и завершаем алгоритм. Ограничение дальности прорисовки устанавливается для ускорения работы программы.

   5.5. "Шагаем" по лучу $t_{i+1} = t_i + h_i$. Т.к. h_i минимальное расстояние, то мы точно не "перешагнем" через какой-то объект.

6. Будем считать, что пересеклись с объектом, который ближе всего к P_{last}. При хорошо подобранных параметрах до этой части мы будем доходить только в тех случаях, когда луч почти пересекает объект, поэтому разумно считать, что мы его пересекли.

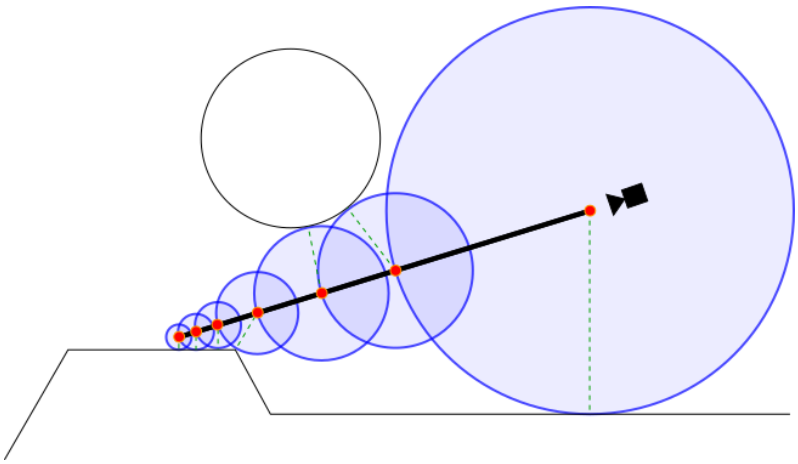

Рис. 2: Принцип работы ray marching-a

4 Программная реализация

Покажем реализацию данного алгоритма на языке программирования glsl. Полные коды программ, используемых для генерации некоторых изображений ниже, приводиться не будут по причине больших размеров и простоты их написания.

```
float marching(in vec3 ro, in vec3 rd) {
    float t = 0.;
    for (float i = 0.; i < MARCHING_ITERATIONS; ++i) {
        vec3 P = ro + rd*t;

        float h = sdf(P);
        if (h < RAY_INTERSECTION_TRASHOLD)
            break;
        if (t > ZFAR)
            break;
        t += h;
    }
    if (t > ZFAR) t = inf;
    return t;
}
```

Рис. 3: Изображения, полученные методом ray marching-a (без освещения)

5 Освещение

Как видно на рисунке 3 трехмерные объекты выглядят нереалистично. Проблема в том, что они не освещены, но это легко исправить. Например, можно использовать модель освещения Фонга. Однако для этого необходимо знать нормаль к поверхности фигур. Её легко вычислить, учитывая, что мы знаем $sdf(P)$.

$$\vec{n} = normalize\left(\frac{d \cdot sdf}{dx}, \frac{d \cdot sdf}{dy}, \frac{d \cdot sdf}{dz}\right)$$

, где $\vec{n}$ — вектор нормали к поверхности. С освещением объекты выглядят значительно лучше. (См. рисунок 4.)

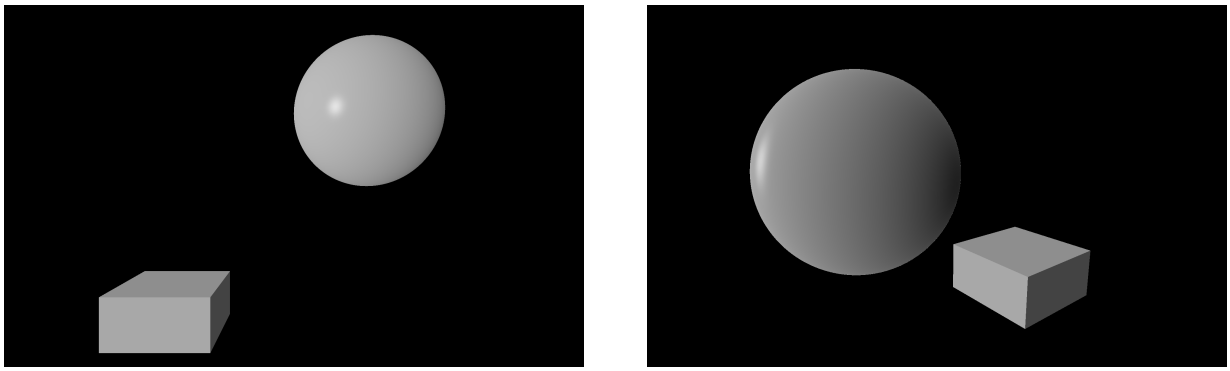

Рис. 4: Изображения, полученные методом ray marching-a (с освещением)

6 Преимущества и недостатки данного метода

6.1 Преимущества

- Достаточно быстрый для того, чтобы работать в реальном времени (при малом количестве объектов для отрисовки).
- Найти *sdf* достаточно легко даже для сложных фигур, например, оболочки Мандельброта или проекций трехмерных объектов.
- Над *sdf* просто производить различные преобразования: зеркально отражать масштабировать, изгибать и т.д.
- *sdf* позволяет быстро считать нормаль к плоскости, что полезно для создания освещения.

6.2 Недостатки

- Скорость работы алгоритма, а точнее скорость поиска значения *sdf*, быстро понижается при добавлении большого количества объектов, что, например, не позволяет использовать только ray marching при создании компьютерных игр.

7 Галерея

Здесь показаны несколько изображений, сгенерированных программами, написанными мной с использованием мотода ray marching-a.

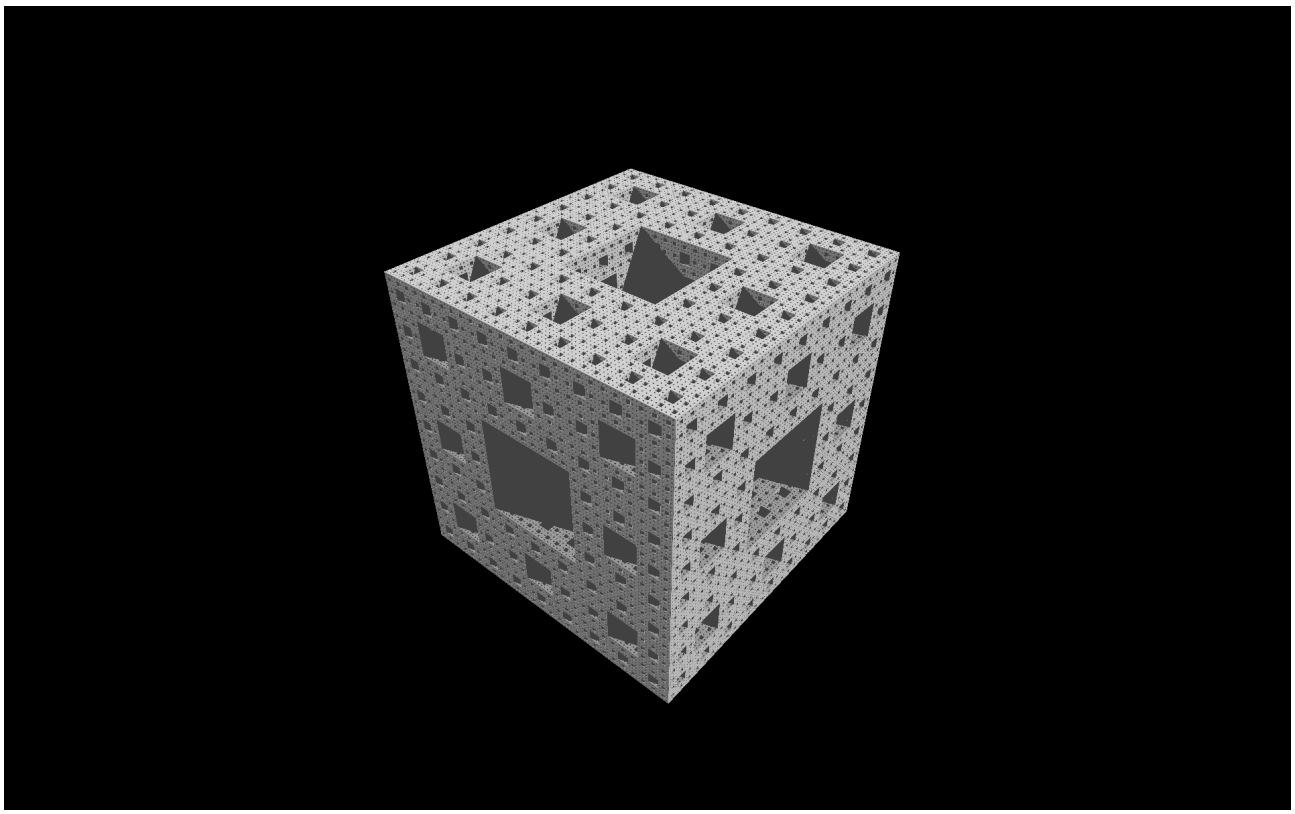

Рис. 5: Губка Менгера (вид снаружи)

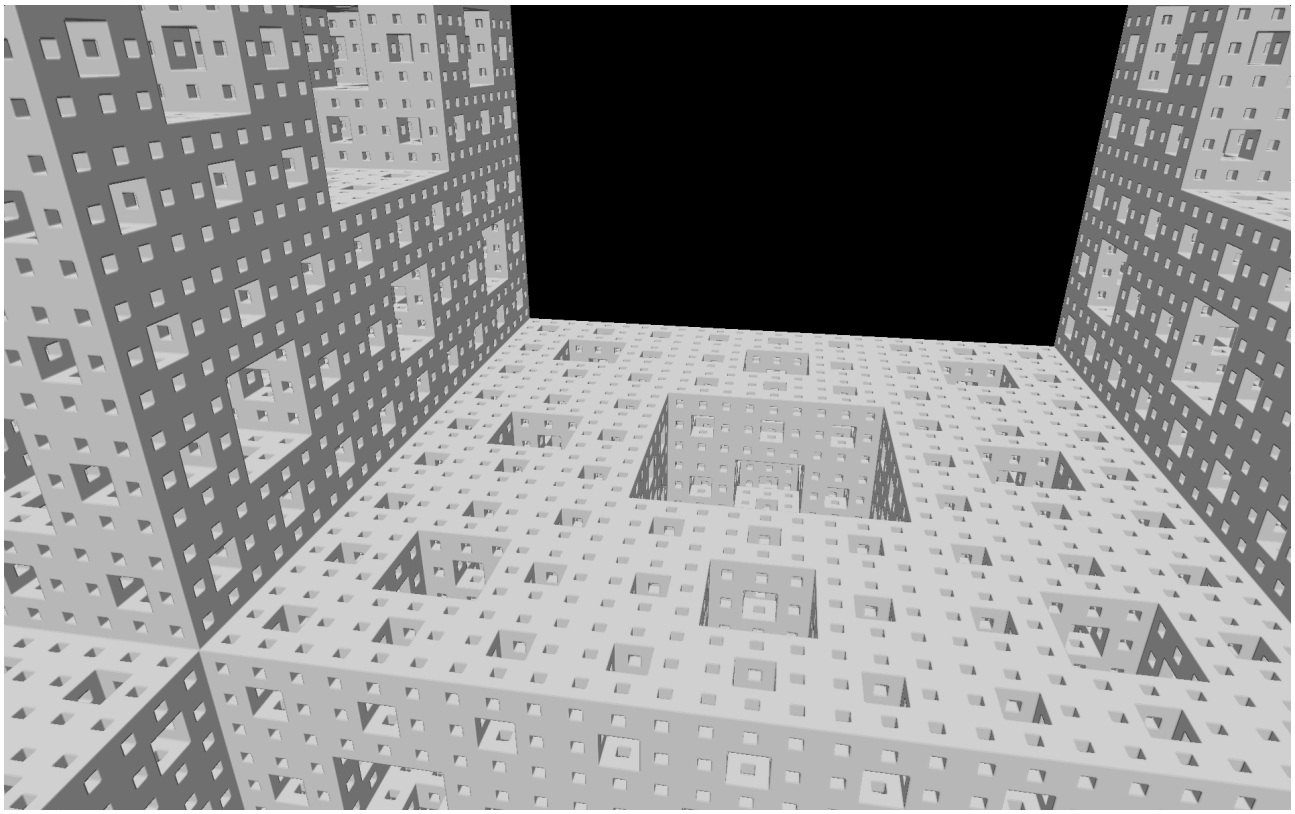

Рис. 6: Губка Менгера (вид изнутри)

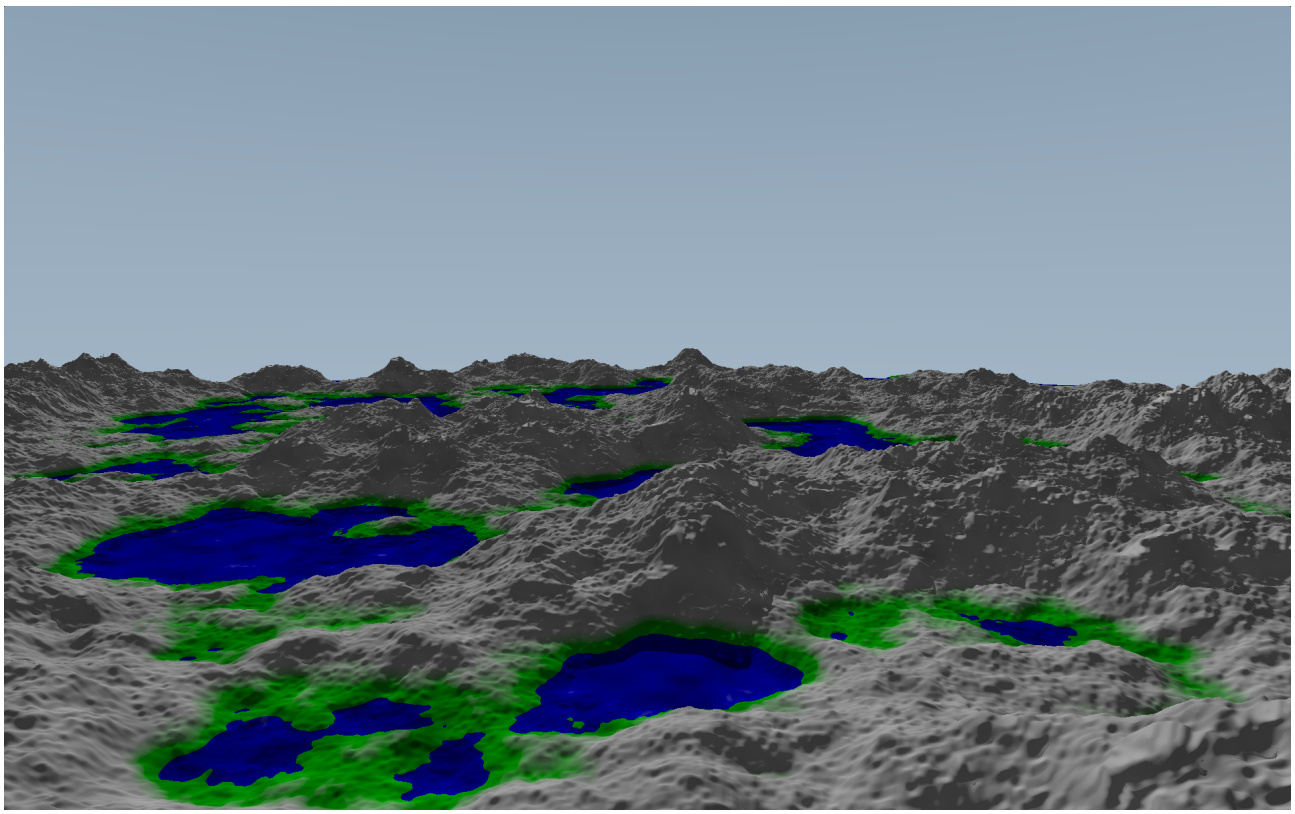

Рис. 7: Горный ландшафт (на основе шума Перлина)

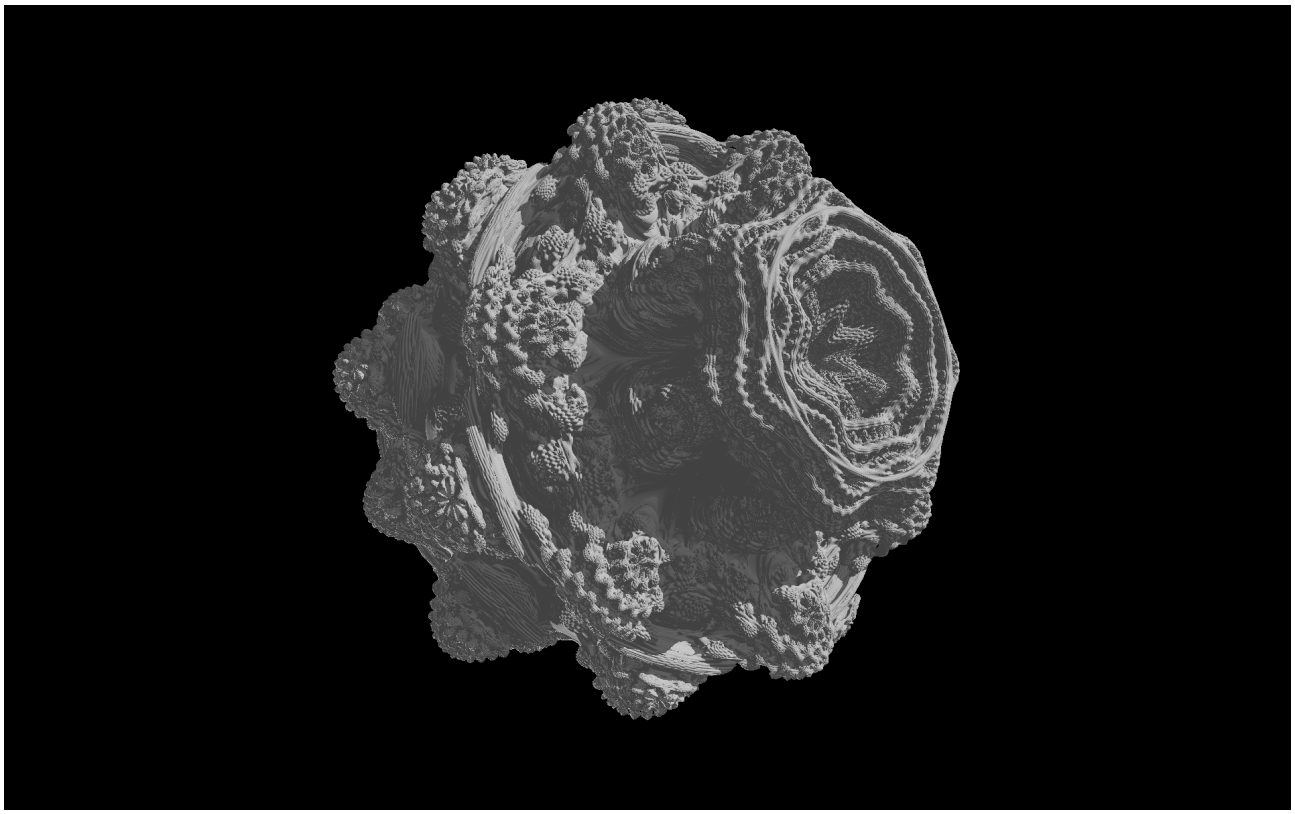

Рис. 8: Оболочка Мандельброта