Задание 1. Трассировка лучей

Описание задания

Задание посвящено созданию, модификации и визуализации 3D моделей, представленных полигональными мешами и SDF. Вам предлагается реализовать две популярных структуры для хранения произвольных SDF - регулярная сетка и октодерево, а также метод рендера мешей трассировкой лучей с использованием ускоряющих структур. Для задания есть шаблон, где происходит создание окна с выводом на экран массива пикселей, а также описаны структуры данных и функции их чтения/записи из файла. Это задание, как и курс в целом, фокусируется на алгоритмах, поэтому предполагается самостоятельная реализация всех пунктов задания в программе на языке C++, без использования сторонних библиотек. Для многопоточного выполнения программы можно использовать средства стандартной библиотеки или OpenMP. Мы не предполагаем, что программа будет реализована на GPU, но это возможно, реализация рендера на Vulkan принесет немного дополнительных баллов.
Задание состоит из 3 частей: A - рендер меша трассировкой лучей, B - работа с SDF на регулярной сетке, C - работа с SDF октодеревом.
Вопросы по заданию можно писать Альберту Гарифуллину (@SammaelT)

Требования к оформлению

Необходимо написать программу, работающую в двух режимах:

1) Конвертация. Приложение принимает в качестве аргументов входной и выходной файл и параметры конвертации (размер сетки для SDF grid и максимальную глубину для SDF octree). Входной файл - меш в формате obj, выходной - SDF grid или octree. Файл с SDF grid должен иметь расширение .grid, с SDF octree - .octree.
2) Визуализация - программа принимает на вход путь до модели (ее тип задается расширением - obj, grid, octree) и открывает окно, где в реальном времени рендерится заданная модель. Пользователь управляет движением и вращением камеры с помощью клавиатуры и мыши.

При желании можно добавить еще третий режим - рендер одного кадра с заданной камерой как изображения на диск.

Решение нужно прислать в виде zip архива с названием следующего формата:

<номер группы>_<инициалы>_<фамилия>.zip

Выполненное задание вы сдаете в google classroom, оно должно содержать исходный код, инструкцию по сборки (без крайней необходимости не отходите от сборки из шаблона), readme с перечислением выполненных заданий, а также описанием, как запускать программу для проверки каждого из выполненных пунктов. Также можно приложить примеры построенных SDF. Для выполнения задания **запрещается** использование сторонних библиотек, за исключением тех, что уже содержатся в шаблоне.
Задание будет приниматься очно, с 17:00 до 18:00 в понедельник. При необходимости будут назначены дополнительные дни для сдачи.

Часть A - Mesh

A1. Рендер плоскости, заданной уравнением (1 балл)
Первый пункт для знакомства с принципом трассировки лучей. Здесь и далее должно быть освещение по модели Ламберта (см. теоретические сведения). Плоскость задается уравнением ***ax + by + cz + d = 0***. Уравнение плоскости можете выбрать любое, но для дальнейшего выполнения задания удобно взять плоскость “пола” ***y = 0***

A2. Рендер треугольного меша (наивный) (1 балл)
Полигональные сетки, состоящие из треугольников (треугольные меши) - самый распространенный способ представления 3D моделей в компьютерной графике. Каждый такой меш состоит из списка вершин, нормалей и индексов. Тройки индексов задают треугольники, из которых и состоит меш. В шаблоне уже есть структура данных, описывающая меш (некоторые поля оттуда не будут использоваться) и возможность ее загрузки из obj файла, а также примеры моделей.
Наивный способ рендера меша заключается в следующем: пересекаем каждый луч с каждым из треугольников и выбираем самую близкую точку пересечения.
Таким образом, можно визуализировать очень простую модель (например, куб), но из-за линейной зависимости сложности от размера модели на практике этот метод неприменим.

A3. Рендер треугольного меша (классический) (3 балла)
Для рендера настоящих моделей нужно построить BVH, вспомогательную структуру данных, которая позволит искать пересечение луча с мешем на log(N), где N - число треугольников в меше. Про методы построения BVH и его обхода во время рендера можно посмотреть в разделе “Краткие теоретические сведения” и справочных материалах к лекциям.

A4. Тени (1 балл)
Реализация “жестких” теней от точечного или направленного источника света.

A5. Зеркальные отражения (1 балл)
Реализация простых зеркальных отражений (идеально гладкая поверхность, без шероховатости). Некоторые объекты на сцене (например, плоскость, на которой стоит меш) должны быть зеркальными.

Часть B - SDF grid

B1. Рендер модели, представленной SDFGrid (2 балла)
Необходимо реализовать функцию для визуализации модели с помощью алгоритма sphere tracing[1]. Считаем, что данная нам регулярная сетка описывает куб с центром в луче и стороной 2 - область пространства [-1,1]^3. Значения SDF в произвольной точке вычисляется как трилинейная интерполяция 8 ближайших к этой точке значений из регулярной сетки. Для построения изображения необходимо пустить один луч на каждый пиксель картинки, начало каждого из этих лучей будет совпадать с позицией камеры. Sphere tracing позволит определить точку пересечения луча с моделью, после чего нужно определить цвет поверхности в этой точке, используя модель освещения Ламберта. им в нашу визуализацию модель освещения Ламберта - самую простую из возможных. Для этого нам нужно знать нормаль к поверхности там, куда попал луч. Воспользуемся для этого свойством SDF: если SDF(p) = 0 (p - точка на поверхности), то normal = normalize(dSDFp)/dp). Вы можете явно посчитать производную для SDF или воспользоваться методом конечных разностей в окрестности p.
В репозитории с шаблоном есть модель, на которой можно проверить правильность алгоритма рендера. Если все работает правильно, то вы увидите изображение кролика примерно как на рисунке выше.

B2. Построение регулярной сетки по мешу (2 балла)
Необходимо реализовать метод, строящий SDF регулярную сетку заданного разрешения (одинакового по всем осям) по мешу. Этот метод должен корректно работать при следующих ограничениях на входной меш:

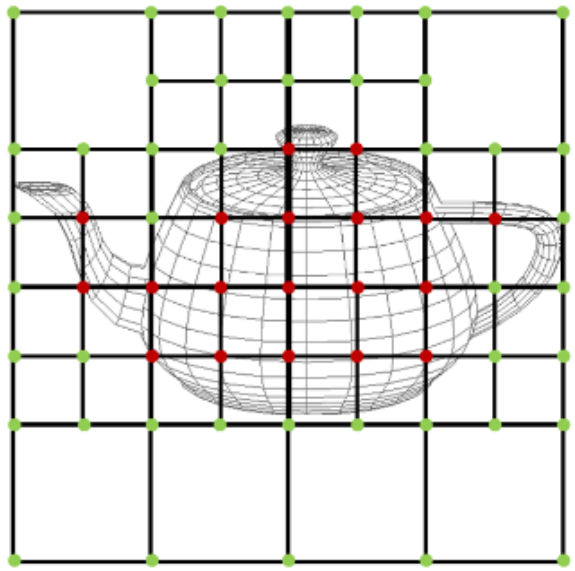
1) Он полностью помещается в куб [-1,1]^3
2) Он watertight, т.е. совокупность треугольников, составляющих меш, однозначно делит пространство на внутренность и наружность модели.
3) для произвольной точки ***p***, корректная SDF может быть вычислена следующим образом - пусть ***(a,b,c)*** - самый близкий к ***p*** треугольник, ***pt*** - ближайшая к ***p*** точка треугольника ***(a,b,c)***. Тогда:
   SDF(p) = sign(dot(cross(a - b, a - c), p - pt))*length(p-pt)

Ваша реализация должна корректно и с разумной скоростью (не более нескольких секунд на модель в разрешении не ниже 512x512) работать на моделях из шаблона при разрешении сетки до 256^3.

B3. Модификация регулярной сетки (2 балла)
Необходимо реализовать алгоритм модификации, который “вырезает” из модели, представленной SDF регулярной сеткой, шар с заданной позицией и радиусом. У пользователя программы должна быть возможность задавать параметры шаров через командную строку и видеть на экране изображение изменившейся модели. Ситуации, когда заданный шар частично или полностью находится за границами сетки, должны корректно обрабатываться. Модификация должна занимать меньше времени, чем построение сетки с нуля.

Часть C - SDF octree

C1. Рендер модели, представленной SDF Octree (2 балла)
В отличие от регулярной сетки, октодерево позволяет гораздо компактнее хранить SDF и также пропускать большие регионы, где нет поверхности. В задании используется структура данных, называемая sparse voxel octree[2]. В каждом узле такого октодерева хранятся 8 значений дистанций в углах (см. рис.). Таким образом, если мы знаем в каком узле находится точка, то для подсчета SDF в ней достаточно только значений в этом узле. Для рендера SDF Octree нужно реализовать алгоритм обхода октодерева [3], а в каждом непустом узле на пути луча искать пересечение с поверхностью методом sphere tracing.



C2. Построение регулярной SDF Octree по мешу (3 балла)
Задание аналогично **B2**, однако дополнительно требуется, чтобы полученное октодерево было разреженным - общее число узлов значительно меньше, чем в регулярной сетке. Параметром, отвечающим за точность, является глубина. Октодерево глубины **d** должно представлять модель с той же точностью, что и регулярная сетка размера **2^d**.

C3. Модификация SDF Octree (4 балла)
Аналогично **B3.** Дополнительно требуется удаление лишних углов октодерева там, где уже нет поверхности. Например, при крайнем случае - вычитании из сцены шара очень большого (10^6 например) радиуса - октодерево должно редуцировать до одного узла.

Часть D - дополнительные пункты

D1. Ambient Occlusion, SDF grid (1 балл) + SDF Octree (1 балл)
Реализовать фоновое затенение (ambient occlusion) по алгоритму, описанному в [4]

D2. Быстрое построение SDF grid (3 балла)
Вычислять дистанции только в тех вокселях, где лежит хотя бы один треугольник. Остальные помечать как пустые. После чего применить процедуру redistancing[5].[6], для того чтобы заполнить значения SDF в пустых вокселях уже без информации о меше.

D3. Аналитический метод (2 балла)
Реализовать аналитический метод пересечения луча с листом SDF octree, описанный в [2]
D4. Метод Ньютона (1 балл)
Реализовать пересечение луча с листом SDF octree методом Ньютона, как описано в [2].

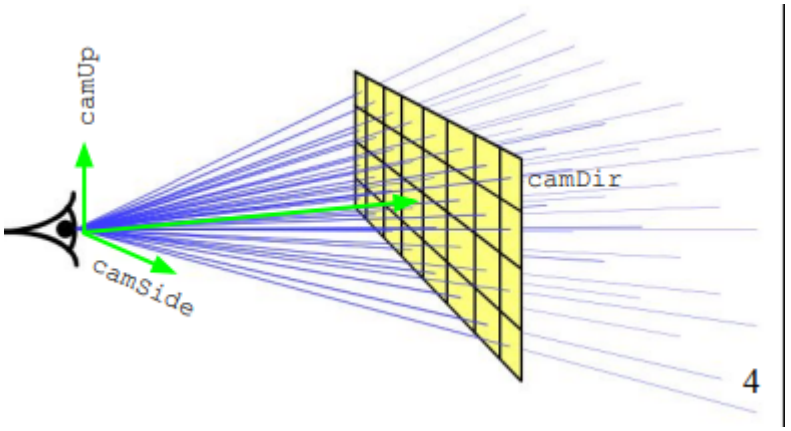
Шаблон задания

https://github.com/SammaelA/SdfTaskTemplate
Шаблон содержит:

- Определения структур SdfGrid, SdfOctree, и SimpleMesh (mesh.h) и функции для загрузки их из файлов
- Пример отрисовки силуэта модели SdfGrid
- Пример сохранения кадра на диск с помощью LiteMath/stb_image
- Пример создания окна приложения с помощью библиотеки SDL и обработки нажатий клавиш

Краткие теоретические сведения

Получение изображения трассировкой лучей



На каждый пиксель изображения выпускается несколько один или несколько лучей, итоговый цвет получается усреднением цветов лучей. Для определения цвета луча необходимо найти точку его пересечения со сценой. Если луч не пересекается ни с

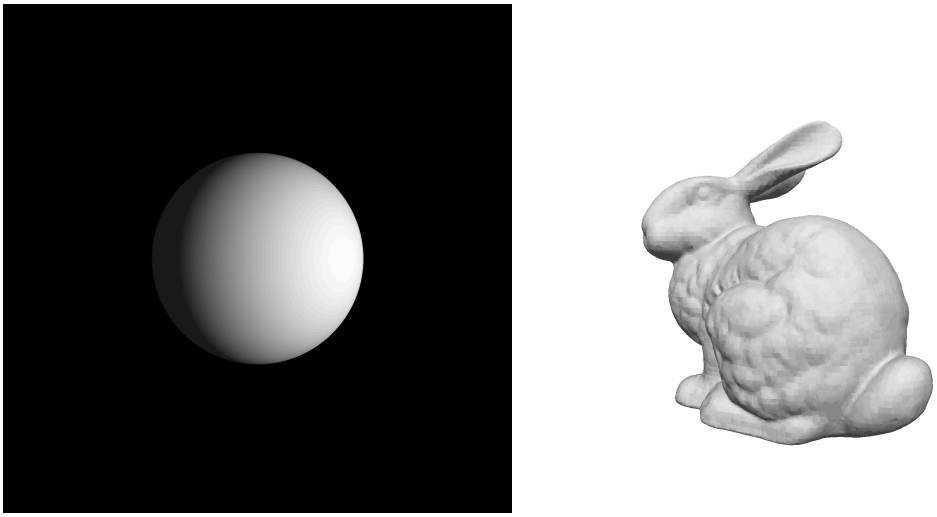
одним объектом на сцене, то он приобретает цвет фона (обычно константный белый или черный цвет, также может быть цвет из панорамного изображения (cubemap)). В случае попадания, цвет луча вычисляется с использованием некоторой модели освещения. В задание предлагается использовать модель освещения Ламберта - одну из самых простых.
Для этого нам нужно знать нормаль к поверхности там, куда попал луч. Теперь цвет луча можно рассчитать по следующей формуле, где `light_dir` - направление НА источник света.

```
color = base_color*max(0, dot(n, light_dir));
```

Также, чтобы половина модели не сливалась с фоном и не становилась черной, можно немного модифицировать формулу:

```
color = base_color*max(0.1f, dot(n, light_dir));
```

Для имитации освещения от солнца (бесконечно удаленной точки) light_dir фиксируется для всей сцены.
Base_color - это свойство того объекта, в который попал луч. Он может быть константным, зависеть от позиции объекта или получаться путём сэмплирования ассоциированный с объектом текстуры.

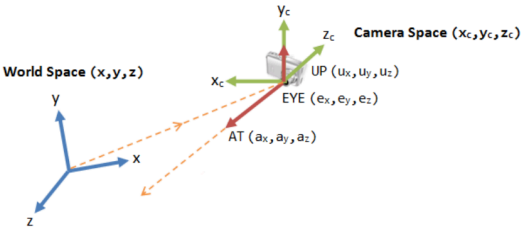


Примеры изображений с моделью освещения Ламберта и base_color = (1,1,1)

Все лучи начинаются в точке расположения камеры, а их направление вычисляется с использованием обратных матриц вида (view) и проекции (projection) камеры. Для теней, ambient occlusion и других более сложных эффектов потребуется запускать дополнительные лучи из других точек, так что интерфейс пересечения луча со сценой должен принимать сам луч, а не координаты пиксела.

Видовое преобразование (View transform)

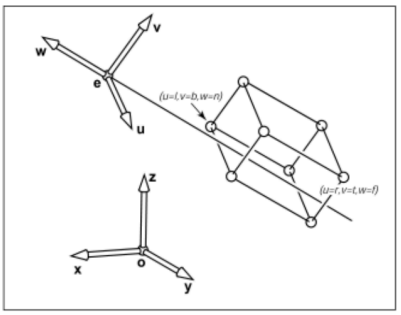
Видовое преобразование – подгоняет мир под камеру, - преобразует мировую систему координат в видовые координаты.
Перенос, чтобы камера располагалась в начале координат **+ смена базиса**, чтобы векторы вправо, вверх и вперед (взгляд) для камеры совпали с (1, 0, 0), (0, 1, 0) и (0, 0, 1).



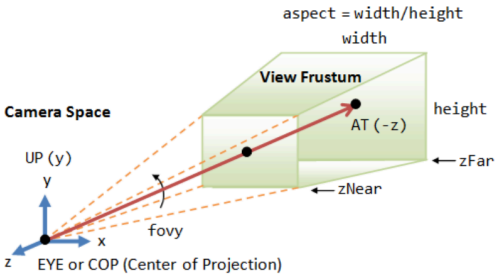
$$\mathbf{M}_{\text{cam}} = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{e} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} x_u & y_u & z_u & 0 \\ x_v & y_v & z_v & 0 \\ x_w & y_w & z_w & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -x_e \\ 0 & 1 & 0 & -y_e \\ 0 & 0 & 1 & -z_e \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{w} = -\frac{\mathbf{g}}{\|\mathbf{g}\|},$$
$$\mathbf{u} = \frac{\mathbf{t} \times \mathbf{w}}{\|\mathbf{t} \times \mathbf{w}\|},$$
$$\mathbf{v} = \mathbf{w} \times \mathbf{u}.$$

- положение наблюдателя/камеры **e**,
- направление взгляда **g**,
- вектор «вверх» **t**.



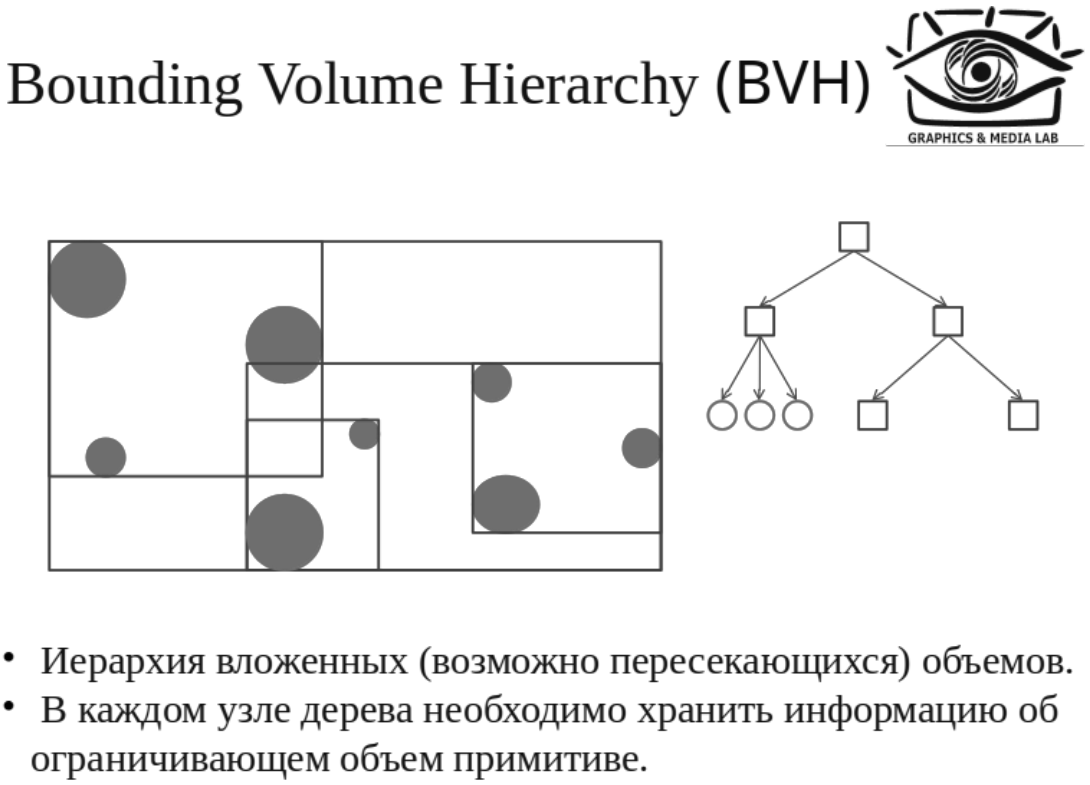
Перспективная проекция



$$left = -zNear * \tan\left(\frac{fovy}{2}\right) * aspect$$
$$right = +zNear * \tan\left(\frac{fovy}{2}\right) * aspect$$
$$bottom = -zNear * \tan\left(\frac{fovy}{2}\right)$$
$$top = zNear * \tan\left(\frac{fovy}{2}\right)$$

$$M_{proj} = \begin{matrix} \frac{2 * zNear}{right - left} & 0 & -\frac{right + left}{right - left} & 0 \\ 0 & \frac{2 * zNear}{top - bottom} & -\frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & \frac{zFar + zNear}{zFar\ - zNear} & -\frac{2 * zFar * zNear}{zFar - zNear} \\ 0 & 0 & 1 & 0 \end{matrix}$$

Bounding Volume Hierarchy



Bounding Volume Hierarchy (BVH) - Иерархия ограничивающих объемов. Как правило (включая данное задание), в качестве объемов выступают параллелепипеды, ориентированные по осям координат (axis-aligned bounding box, AABB).
Построение BVH имеет в своей основе минимизацию метрики - Surface Area Heuristic (SAH). Подробное описание относительно простого алгоритма построения BVH дерева можно найти в [7].

Алгоритм обхода BVH со стеком

Идея заключается в следующем: рекурсивно обходим дерево, каждый раз выбирая ближайший узел при спуске. На каждом шаге считаем пересечение луча с обоими боксами, выбираем ближайший бокс и переходим в него. А дальний помещаем в стек (если конечно луч пересекает оба узла). Если луч пересекает только один узел просто переходив в него. Если луч не пересекает ни одного узла, достаем узлы из стека. К сожалению, нельзя остановить траверс BVH если мы нашли первое пересечение, лежащие внутри какого-либо листа (в kd-tree мы могли в таком случае остановить поиск). Так как в BVH возможны самопересекающиеся узлы, мы обязаны продолжить траверс до тех пор, пока стек не пуст. Однако, если мы видим, что ограничивающий бокс узла лежит дальше, чем уже найденное пересечение, мы можем не траверсить этот узел.

Полезные ссылки

1) Sphere tracing
https://graphics.stanford.edu/courses/cs348b-20-spring-content/uploads/hart.pdf
2) Sparse voxel octree https://www.jcgt.org/published/0011/03/06/paper-lowres.pdf
3) Octree traversal https://otik.uk.zcu.cz/bitstream/11025/15821/1/X31.pdf
4) SDF ambient occlusion
https://www.aduprat.com/portfolio/?page=articles/hemisphericalSDFAO
5) Fast sweep (implementation) https://github.com/rgl-epfl/fastsweep
6) Fast sweep (theory)
https://www.math.uci.edu/~zhao/homepage/research_files/FSM.pdf
7) BVH http://www.ray-tracing.ru/articles184.html