Shadow Volumes

Для расчета теней в реальном времени реализовано два независимых алгоритма теневых объемов : собственно Shadow Volumes для точных «жестких» теней от точечных источников света, и алгоритм, основанный на технике Penumbra wedges для «мягких» теней от объемных источников. Их можно применять независимо для одного или разных источников света в пределах одной сцены.

Реализован метод z-fail с использованием буфера трафарета и геометрического шейдера для построения теневых объемов.

Реализация предполагает использование теневых объемов только для некоторого числа объектов, так как их расчет для всей сцены слишком затратен по ресурсам. Информация о каждом объекте, для которого предполагается рисовать теневые объемы, хранится в специальном файле, откуда загружается при запуске программы. Это дает возможность использовать для их расчета модели с пониженным уровнем детализации, что будет незначительно сказываться на точности теней, и значительно — на их производительности.

Основной цикл работы состоит из 3 этапов:

1. Нарисовать всю сцену в буфер глубины
Для каждого источника света -

2. Для всех объектов, отбрасывающих тень дважды вызвать шейдер, строящий теневые объемы для силуэтных ребер — сначала нелицевые грани, с увеличением значения в буфере трафарета для тех фрагментов, что не прошли тест глубины. Второй раз — лицевые, с уменьшением значения в буфере трафарета для тех фрагментов, что не прошли тест глубины. Сами теневые объемы строятся геометрическим шейдером на основании информации о смежных гранях для каждого треугольника (тип TRIANGLES_ADJACENCY).

3. Снова рисуется вся сцена, но лишь те фрагменты, для которых значение буфера трафарета отлично от 0, т.е. находящиеся в тени. В красный канал при этом пишется 1, в зеленый — некоторая функция от расстояния до объекта, тень более темная в начале, ближе к источнику света, и более бледная вдали от него.

Результатом работы является маска тени для текущего кадра, которая потом используется в шейдере расчета освещения. Такой подход является более затратным, но позволяет комбинировать в одном кадре тени разных типов.

Soft Shadow Volumes

Основная идея — построение некоторых «клиньев полутени» в дополнение к теневым объемам, взята из статьи (Assarsson, Akenine-Möller) [1]. Предложенный в статье алгоритм был серьезно изменен, чтобы иметь возможность выполнять его в реальном времени полностью на GPU.

Основные этапы
1. - 3. Расчет теневых объемов (см. выше)
4. Очистка буфера трафарета. Построение «клиньев полутени» для силуэтных ребер в геометрическом шейдере. Здесь, в отличие от оригинального алгоритма, мы не располагаем

информацией о соседних клиньях и вынуждены рисовать их независимо, что может приводить к неточностям, если источник света слишком большой или тень падает на поверхность далеко от объекта, ее отбрасывающего. Однако, в большинстве случаев, эти неточности незаметны в результате. Как и в случае теневых объемов, заполнение буфера трафарета происходит в два прохода.
5. Аналогично 3. в методе теневых объемов, но в канал красного цвета пишется 0.5 вместо 1. Таким образом, полутень увеличивает размер затененной области, но при это уменьшает размер полностью темной ее части.
6. На этом этапе строится градиент в области полутени. Метод, описанный в оригинальной статье, требует информации о клиньях полутени для этого этапа. Метод, реализованный в данном проекте, не требует почти никакой информации о геометрии сцены и работает в экранном пространстве. Помимо текстуры, полученной на предыдущем этапе, требуется список координат некоторых центров теней. В большинстве случаев для простых объектов (близких к выпуклым многогранникам) они совпадают с центрами самих объектов, но их можно устанавливать и вручную. Для расчета затененности каждого фрагмента делается следующее:
- рассчитывается направление C на ближайший центр тени в экранном пространстве
- создается N случайных векторов, между каждым и C угол не более α и противоположно направленные к ним (N,α — настраиваемые параметры)
- каждый из векторов трактуется как направление, по которому от исходного фрагмента осуществляется бинарный поиск точек находящихся в тени для первой группы, и на свету — для второй. Вычисляется расстояние до найденных точек.
- Выбираем минимальные расстояния до света и тени R_{min} ,R_{max}. Тогда затененность в точке вычисляется во формуле $S = S_b * (0.5 + 0.5 * \sin(\pi * (\frac{R_{max} - R_{min}}{R_{max} + R_{min}})))$

где S_b - значение зеленой компоненты цвета в точке, вычисляется на предыдущем этапе и зависит от расстояния до источника света.

Данный метод дает хорошие результаты для теней относительно простой формы, при этом объем вычислений и выборок из текстуры (30 — 40 на фрагмент) не намного превышает аналогичные показатели для шейдера размытия по Гауссу. Для теней сложной формы, не представимых в виде нескольких выпуклых многоугольников (сетки, решетки, листва, наборы мелких деталей) данный метод избыточен, для создания правдоподобной тени подходят и более простые алгоритмы. Однако в пределах области своей применимости он позволяет достичь результатов лучше, чем при использовании размытия.

Для демонстрации обоих алгоритмов используется одна из локаций в основной сцене, два источника света — точечный и в виде системы частиц расставлены для демонстрации обоих типов теней и их комбинации друг с другом. Также в режиме четырех текстур можно посмотреть и карты теней для обоих алгоритмов.

Irregular Z-buffer Shadow Maps

Техника построения точных теней, основанная на статье (Johnson, Mark, Burns)[2]. Данная реализация основана на возможностях OpenGL 4.5, а именно Shader Storage Buffer Object и атомарные операции, и вынесена в отдельный проект. Работа алгоритма состоит из 3 частей и по большей части повторяет алгоритм, описанный в статье [2].
1) Построение вспомогательной структуры данных вида:

```
struct node
{
    uint link;
    uvec2 vp_pos;
```

```
    vec2 lp_pos;
};
layout(std430, binding = 2) buffer data
{
    volatile uint counters[CLUST];
    coherent uint head_data[W*H];
    writeonly node nodes[W*H];
};
layout(std430, binding = 3) buffer light_depth
{
    coherent uint depth[W*H];
};
```

По существу, эта структура представляет собой массив, каждый элемент которого есть список, состоящий из пар векторов. Так, `head_data` - это сам массив, содержащий ссылки на первые элементы списков (т. е. Их номера в массиве `nodes`). Каждый элемент этого массива отвечает за 1 пиксель на экране у источника света (считая, что пространство источника света проектируется на экран с теми же параметрами, что и у камеры). Каждый элемент списка — это фрагмент (пиксель) на экране наблюдателя, который проектируется в соответствующий пиксель на экране источника света, для него хранится его положение на экране источника света (точное, что и обеспечивает большую точность теней в итоге) и координаты в пикселях на экране наблюдателя для упрощения дальнейших вычислений. Массив `nodes` это пространство для хранения звеньев списка, а `counters` — целочисленные переменные, ответственные за выделение номеров для новых звеньев за счет атомарных операций, число CLUST выбирается эмпирически для достижения максимальной производительности, в данной реализации CLUST = 32.

2) Irregular Rasterization. Поскольку нам требуется вычислить глубину в системе координат источника света для полученных на предыдущем этапе точек, не допуская округления до центра ближайшего пикселя, то стандартный механизм растеризации нам не подходит. Вместо этого реализован механизм растериции в геометрическом шейдере, используется вторая вспомогательная структура данных:

```
layout(std430, binding = 3) buffer light_depth
{
    coherent uint depth[W*H];
};
```

Она, по сути является аналогом одноканальной текстуры, в которую будет записываться глубина фрагмента, но, поскольку над ней необходимо проводить атомарные операции, то вместо глубины d хранится как целое число 10^9*d. Заполняется следующим образом:

- Рисуем сцену с позиции источника света, для каждого треугольника определяем те пиксели на экране источника света, которые пересекает данный треугольник (не обязательно центры пикселей, а хотя бы малую его часть). Проходим циклом по соответствующим элементам основной структуры данных и по всем элементам списка, для каждого элемента `node` рассчитываем барицентрические координаты `node.lp_pos` относительно данного треугольника, если положительные — рассчитываем глубину этого элемента (сумма глубин в вершинах с коэффициентами — барицентрическими координатами) и записываем во вспомогательную структуру данных по адресу `node.vp_pos` имеющему смысл положения этого элемента на экране у наблюдателя. Таким образом, мы узнаем глубину в системе координат источника света в точности для тех точек, которые формируют картину на экране у наблюдателя.

3) Рисуем квадрат на весь экран и во фрагментом шейдере формируем из вспомогательной структуры данных обычную текстуру — маску освещенности текущего кадра.

Таким образом, получен алгоритм построения точных «жестких» теней в реальном времени. Однако, данная реализация имеет один существенный недостаток, а именно: крайне ресурсоемкий процесс растеризации выполняется здесь хоть и на GPU, но программно, что не позволяет получить приемлемую производительность. Сами авторы в статье [2] и (Johnson et al.)[3] указывают, что архитектура видеокарт на тот момент не позволяла применять данный алгоритм для приложений в реальном времени и предлагали свои варианты архитектуры для этого. На данный момент есть и другие подходы к этому вопросу, например «Fast triangle rasterization using an irregular z-buffer on CUDA» (Zhang, Majdandzic)[4].

Из-за низкой производительности демонстрация алгоритма вынесена в отдельный проект, сцена упрощена, а камера двигается не свободно, а облетает сцену по кругу

Physical Based Rendering

В проекте используется модель освещения Кука-Торренса. Материалы имеют две характеристики: металличность и грубость. Коэффициент Френеля вычисляется приближенно. Поддерживается использование текстуры материала. (Можно увидеть на дверях и шторах).
Также на этапе расчета освещения создается эффект Bloom, это позволяет оперировать настоящим коэффициентам освещенности в точке, до применения гамма-коррекции и приведения цвета к диапазону [0,1], что позволяет отличать сильно освещенные области от просто ярких по цвету.

Экранные отражения (Screen Space Reflections)
Реализованы экранные отражения, зависящие от характеристик отражающей поверхности. Идея создания эффекта глянцевых отражений взята отсюда:

- https://lukas-hermanns.info/download/bachelorthesis_ssct_lhermanns.pdf — описание sslr.
- http://advances.realtimerendering.com/s2014/#_REFLECTIONS_AND_VOLUMETRICS

Алгоритм состоит из 3-х шагов:

1. Рейтрейсинг в пространстве экрана.
   Для каждой точки вычисляется вектор отражения, после чего в экранном пространстве алгоритм проходит с небольшим шагом (чем более шероховатая поверхность, тем больше шаг) вдоль прямой от точки, для которой вычисляется отражение, до конца вектора отражения. Ищется пересечение отраженного луча и геометрии из буффера глубины. Если оно найдено, то в текстуру отражения записывается информация о цвете точки и длине луча, а в текстуру маски отражений записывается единица. Если пересечение не найдено, то в текстуру маски записывается 0.
2. Создание мипмапов.
   Создается цепочка мипмапов, где каждое следующее изображение получается размытием предыдущего. Размываются одновременно и текстура отражения, и маска отражения. Чтобы отражения не занимали на экране больше места, чем отражающие поверхности, перед размытием пикселя мипмапа проверяется маска отражений: если в ней 0, то пиксель переносится «как есть».
3. Сборка
   Зная грубость материала пикселя и расстояния, которое прошел отраженный луч, вычисляется номер мипмапа, из которого берется цвет отражения для пикселя. Экранное отражение смешивается с отражениями из кубмапа на основе числа в маске отражений.

Image based lighting
По кубической карте окружения строятся два кубмапа:
Карта облученности — в ней хранится вся информация о непрямом облучении среды.

Предварительно отфильтрованная карта окружения — кубмап с несколькими мипмапами,

каждый из которых представялет карту отражений для разного уровня грубости поверхности. С помощью этих кубмапов приближенно решается уравнение рендеринга. Диффузное освещение объекта берется из карты облученности. Отраженное освещение приближенно представляется в виде произведения интегралов двулучевой функции отражательной способности (BRDF) (предварительно вычисленного и сохраненного в отдельную текстуру) и интеграла освещения, который берется из предварительно отфильтрованной карты окружения для нужного уровня грубости материала.

-

Ссылки:

[1] Approximate Soft Shadows on Arbitrary Surfaces using Penumbra Wedges Tomas Akenine-Möller and Ulf Assarsson Chalmers University of Technology, Sweden 2002
URL: https://pdfs.semanticscholar.org/2cd5/b473313df4631a8ad7948e20cc0d31aabea4.pdf?_ga=2.19560636.50134113.1579000189-349826618.1577699624

[2] The Irregular Z-Buffer and its Application to Shadow Mapping Gregory S. Johnson William R. Mark Christopher A. Burns The University of Texas at Austin 2004
URL: http://www.cs.utexas.edu/ftp/techreports/tr04-09.pdf

[3] The Irregular Z-Buffer: Hardware Acceleration for Irregular Data Structures GREGORY S. JOHNSON, JUHYUN LEE, CHRISTOPHER A. BURNS, and WILLIAM R. MARK University of Texas at Austin Oct 2005
URL: http://pl887.pairlitesite.com/papers/tog05-izb/irregular-z-buffer-TOG-Oct2005-preprint.pdf

[4] Fast triangle rasterization using an irregular z-buffer on CUDA Wei Zhang Ivan Majdandzic UNIVERSITY OF GOTHENBURG June 2010
URL: http://www.cse.chalmers.se/~uffe/xjobb/Zhang%20and%20Ivan%20-%20irregular_rast.pdf