Регулярные сетки для высоко реалистичной визуализации 3D сцен в реальном времени[1]

А.В. Мальцев, М.В. Михайлюк

Аннотация. В статье предлагаются методы и алгоритмы построения и использования в реальном времени ускоряющей структуры типа регулярная сетка, применяемой при высоко реалистичной визуализации трехмерных сцен с большим числом полигонов путем трассировки лучей. Предлагаемые подходы используют архитектуру параллельных вычислений CUDA и подходят для визуализации как статических, так и динамических сцен в реальном времени.

Ключевые слова: высоко реалистичная визуализация, трассировка лучей, ускоряющие структуры, регулярная сетка.

Введение

Среди значимых направлений современной компьютерной графики можно выделить повышение качества создаваемых с помощью компьютера изображений и приближение их к окружающей нас реальности. Такие изображения принято называть высоко реалистичными. Одним из способов получения высоко реалистичного изображения является метод трассировки лучей. Он состоит в прослеживании путей распространения световых лучей между виртуальной камерой и источниками света в трехмерной виртуальной сцене. Основной проблемой данного метода является его большая вычислительная сложность для полигональных сцен с количеством полигонов, превышающим 10^5-10^6. Это обусловливается необходимостью определять точную траекторию каждого из десятков миллионов лучей и точку его пересечения с поверхностью треугольного полигона, ближайшего по направлению этого луча к его началу. Элементарный подход базируется на тестировании каждого луча с каждым треугольником на возможность их пересечения. Однако, когда количество полигонов превышает несколько десятков или сотен тысяч, требуется высокое разрешение синтезируемой картинки и сцена является динамической (объекты меняют свое положение с течением времени), эта задача становится слишком сложной для реализации в режиме реального времени (то есть с частотой смены кадров не менее 25 раз в секунду) даже на современных компьютерах.

С целью сокращения времени генерации изображений были разработаны так называемые структуры ускорения. Они представляют разбиение всего пространства виртуальной сцены на трехмерные области, каждой из которых соответствует список пересекающих ее полигонов или вложенных в нее областей (для иерархических структур). Данный подход позволяет избежать перебора всех треугольников сцены для проверки пресечения с трассируемым лучом. С помощью структуры ускорения луч тестируется на пересечение лишь с некоторым небольшим подмножеством полигонов.

[1] Работа выполняется при поддержке РФФИ (грант № 12-07-00256)

Одним из типов структур ускорения является регулярная сетка – разбиение пространства виртуальной сцены плоскостями, параллельными плоскостям мировой системы координат, на трехмерные ячейки (вокселы) одинакового размера, каждой из которых соответствует список полигонов, пересекающих эту ячейку. Имея сетку виртуальной сцены, можно для каждого луча определить последовательность ячеек, через которые он проходит, и тестировать луч на пересечение только с полигонами, принадлежащими этим ячейкам.

В данной статье предлагаются методы и алгоритмы построения регулярных сеток динамических трехмерных сцен с большим числом полигонов в реальном времени и использования их при высоко реалистичной визуализации путем трассировки лучей. Данные методы и алгоритмы адаптированы для эффективного применения на современных графических процессорах компании nVidia с поддержкой технологии CUDA.

1. Этапы формирования регулярной сетки для трехмерной сцены

Построение и заполнение регулярной сетки трехмерной виртуальной сцены, состоящей из полигональных объектов (Рис. 1), включает в себя нескольких основных этапов.

- Определение ограничивающего параллелепипеда сцены, стороны которого параллельны координатным плоскостям XY, YZ и ZX мировой системы координат WCS. В дальнейшем ограничивающий параллелепипед сцены (или треугольника), удовлетворяющий приведенным выше условиям, будем называть просто *AABB* (*Axis-Aligned Bounding Box*) и задавать двумя вершинами: *минимальной* ($\min(V_{i,x})$, $\min(V_{i,y})$, $\min(V_{i,z})$) и *максимальной* ($\max(V_{i,x})$, $\max(V_{i,y})$, $\max(V_{i,z})$). Здесь минимум и максимум берется по всем восьми вершинам *AABB* (i – номер вершины бокса AABB).
- Разбиение AABB сцены плоскостями, параллельными XY, YZ и ZX, на множество кубических ячеек, называемых вокселами.
- Составление таблицы (массива), хранящей для каждого воксела номера треугольников, с которыми он пересекается.

Так как мы рассматриваем динамические сцены, в которых многие объекты с течением

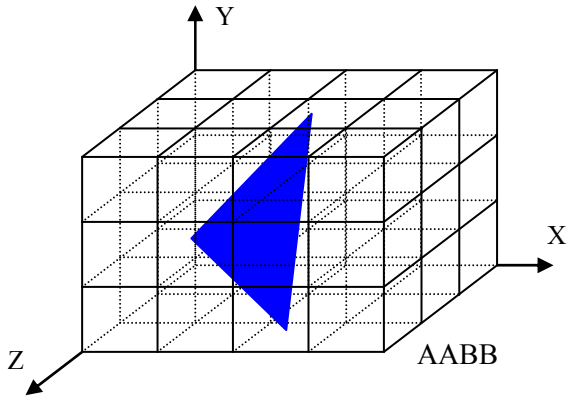


Рис. 1. Регулярная сетка сцены

времени могут менять своё положение, ориентацию и размер (например, при морфинге), то необходимо в каждом кадре осуществлять перестроение сетки. Для обеспечения реального времени составление таблицы, хранящей для каждого воксела номера треугольников, с которыми он пересекается, должно быть реализовано с малыми вычислительными затратами.

Существуют два основных подхода к составлению такой таблицы. Первый – пройти по всем вокселам и некоторым образом определить относящиеся к ним треугольники, например, перебором всех треугольников для каждого воксела. В этом случае, принимая за элементарную операцию тест пересечения одного полигона с одним вокселом регулярной сетки, мы получим $M \cdot N \cdot K \cdot t$ операций, где M, N, K – размерность AABB сцены в вокселах по осям X, Y и Z соответственно, а t – количество треугольников в сцене. Второй подход – пройти по всем полигонам, определив, какие вокселы каждый из них пересекает. Тогда, если $M \geq N \geq K$, то количество элементарных операций будет не больше, чем $M \cdot N \cdot t$, поскольку максимально возможное число вокселов, занимаемых одним треугольником, не превышает $M \cdot N$. При этом, чтобы составить таблицу, хранящую для каждого воксела номера треугольников, с которыми он пересекается, необходим свой быстрый и эффективный алгоритм.

Далее предлагается модифицированный вариант второго из описанных выше подходов, позволяющий обеспечить построение и заполнение регулярной сетки высоко полигональной динамической сцены в режиме реального времени.

2. Алгоритм построения и заполнения регулярной сетки сцены

На Рис. 2 представлена общая схема предлагаемого алгоритма построения регулярной сетки трехмерной виртуальной сцены. Поскольку этот процесс весьма трудоемкий, то данный алгоритм основан на использовании параллельных вычислений с помощью многоядерных процессоров фирмы nVidia с поддержкой технологии CUDA, позволяющих обрабатывать одновременно сотни потоков в режиме SIMD (*Single Instruction, Multiple Data*). Подробную информацию относительно архитектуры таких процессоров и основных принципов программирования с применением технологии CUDA можно найти в [1].

Рассмотрим идею предлагаемого алгоритма по этапам, сформулированным выше.

2.1. Определение AABB сцены

Для начала необходимо загрузить в память видеоадаптера с поддержкой технологии CUDA координаты вершин треугольников, представленные в локальных системах тех объектов, которым они принадлежат, и матрицы перехода из локальных систем в мировую WCS. Далее нужно найти координаты всех загруженных вершин в мировой системе координат WCS и определить AABB каждого треугольника сцены в виде пары минимальной B_0 и максимальной B_1 вершин бокса. Это делается с помощью параллельной обработки, где каждый поток отвечает за свой треугольник (Рис. 2, Ядро 1; под термином "Ядро" понимается программа для графического процессора, выполняемая на всех его аппаратных ядрах в режиме SIMD).

По рассчитанным AABB треугольников находим минимальную B_{min} и максимальную B_{max} точки для AABB всей сцены также с помощью параллельной обработки (Рис. 2, Ядро 2).

Графический процессор (GPU) фирмы nVidia содержит P_M мультипроцессоров, каждый из которых состоит из P_U универсальных процессоров. Числа P_M и P_U зависят от архитектуры чипа GPU. Разобьем совокупности ранее найденных вершин $B_{0,i}$ и $B_{1,i}$ ($i \in [0, n-1]$) AABB

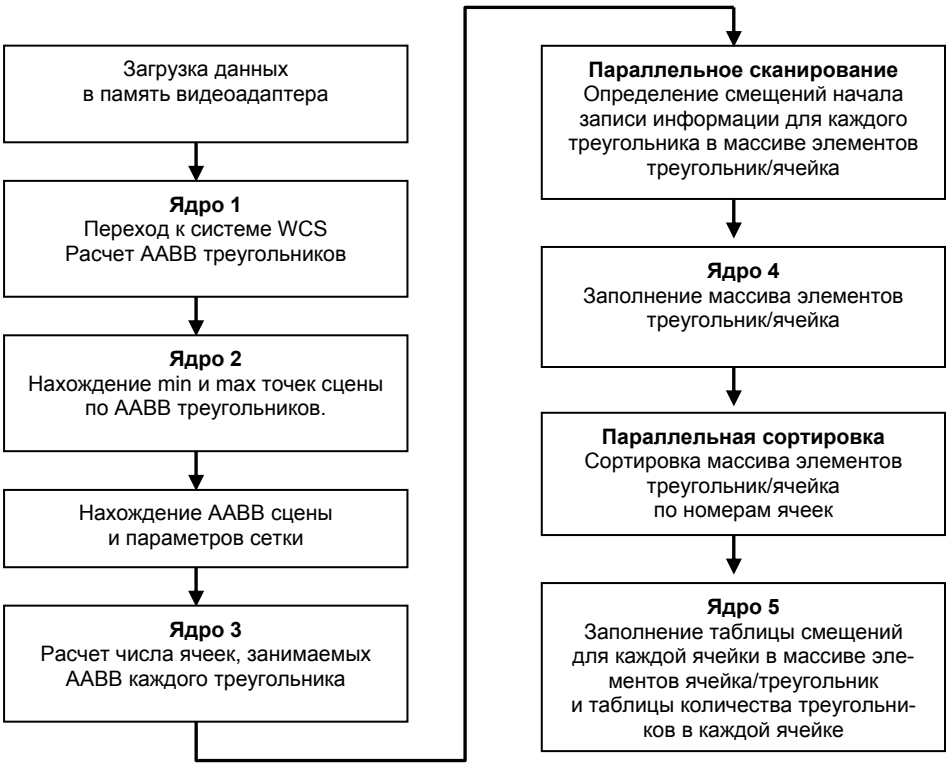


Рис. 2. Процесс формирования регулярной сетки

всех n треугольников сцены на группы по h элементов, где h – максимально возможное число потоков в блоке потоков GPU. Каждая группа будет обрабатываться на одном из P_M мультипроцессоров. Для чипа GK104 (GTX680) $P_M = 8$, $P_U = 192$, $h = 1024$.

В результате вычислений, для каждой группы мы получаем локальный минимум (максимум), после чего все локальные минимумы (максимумы) снова делятся на группы по h элементов и обрабатываются. Процесс повторяется до тех пор, пока мы не получим один общий минимум (максимум) для AABB всей сцены.

Если считать, что одна операция нахождения минимума (максимума) двух чисел выполняется за один такт, то (с учетом сложности чтения чисел из видеопамяти) нахождение минимума (максимума) из n элементов с использованием описанного алгоритма параллельного вычисления составляет не более

$$t = c\frac{n}{P_M \cdot P_U}$$

тактов, где c – константа (зависящая от h и не превышающая 20). Подставляя в эту формулу параметры чипа GK104, получим, что t= 0.013n.

Таким образом, мы описали параллельный **алгоритм определения AABB сцены:**

1. загрузка данных сцены в видеопамять;
2. перевод вершин полигонов объектов в мировую систему координат;
3. определение минимальной и максимальной вершин для AABB каждого полигона;
4. расчет минимальной и максимальной вершин AABB всей сцены.

2.2. Разбиение AABB сцены на вокселы

Вычислим на CPU параметры сетки – длину A ребра кубического воксела и количество вокселов, помещающихся по осям X, Y и Z системы WCS в AABB сцены, т.е. размерность $\mathbf{D} = (D_x, D_y, D_z)$ сетки:

$$A = \sqrt[3]{\frac{V_{AABB}}{\lambda P}};$$

$$\mathbf{D} = \left(\lceil D'_x \rceil, \lceil D'_y \rceil, \lceil D'_z \rceil\right), \mathbf{D}' = \frac{1}{A}\left(B_{\max} - B_{\min}\right),$$

где V_{AABB} – объем AABB сцены, P – общее количество треугольников в сцене, λ – задаваемая пользователем константа, позволяющая делать сетку более плотной или более разреженной, квадратные скобки обозначают округление сверху до ближайшего целого, B_{max} и B_{min} – соответственно максимальная и минимальная точки AABB сцены. Формула для вычисления размера сетки A является эвристической.

2.3. Составление таблиц данных о заполнении вокселов сетки

Имея характеристики сетки сцены, необходимо определить, каким вокселам какие треугольники сцены принадлежат. Чтобы решить эту задачу, надо для начала найти для каждого треугольника те ячейки, которые он пересекает, и составить два взаимосвязанных массива, в одном из которых будут храниться номера треугольников (*T-массив*), а в другом – номера соответствующих им ячеек (*C-массив*). Под номером ячейки будем понимать число, рассчитываемое по формуле

$$i = i_z D_x D_y + i_y D_x + i_x, \quad (1)$$

где i_x, i_y, i_z – индексы ячейки по осям координат X, Y, Z соответственно, начиная с нуля от точки B_{min}.

На Рис. 3 показан пример C- и T-массива, где треугольнику 0 соответствуют вокселы с номерами 3, 4, 5, треугольнику 1 – воксел 0 и т.д. Совокупность массива номеров треугольников и соответствующего ему массива номеров ячеек сетки сцены в дальнейшем будем называть *TC-массивом,* или *массивом элементов треугольник/ячейка*.

Формирование TC-массива осуществляется с помощью параллельной (по треугольникам) обработки с использованием CUDA (Рис. 2, Ядро 4). Так как в технологии CUDA нет возможности динамического выделения глобальной (общей для всего видеоадаптера) памяти в процессе выполнения ядра, то выделять память под TC-массив необходимо заранее. Кроме того, для каждого потока, обрабатывающего свой треугольник, мы должны знать смещение в TC-массиве, с которого этот поток будет начинать запись данных. Поэтому для начала нужно произвести расчет числа ячеек сетки, занимаемых каждым треугольником, и сумму таких чисел по всем треугольникам. Поскольку на данном шаге нам необходимо знать лишь максимальное

количество ячеек под каждый треугольник, чтобы сделать выделение памяти, то с целью ускорения вычислений можно ограничиться грубой оценкой, т.е. количеством ячеек, которые пересекают AABB каждого треугольника.

Для этого в режиме параллельной обработки (Рис. 2, Ядро 3) заполняется массив, длина которого равна общему числу n полигонов сцены, а каждый элемент i соответствует треугольнику с тем же номером и содержит то количество вокселов сетки, которое занимает AABB этого треугольника (Рис. 4).

Далее нужно рассчитать суммарное число элементов в ТС-массиве и смещение (выраженное в количестве ячеек), с которого будет начата запись данных каждым потоком Ядра 4. Это можно сделать с помощью функции сканирования (параллельной *префиксной суммы* [2]) cudppScan массива из библиотеки CUDPP (*CUDA Data Parallel Primitives Library*). На вход cudppScan подается числовой массив длиной n, а на выходе получается массив длиной n+1, в каждом элементе $0 < i \leq n$ которого записана сумма всех элементов от 0 до i-1 входного массива, а нулевой элемент равен 0. Подробнее ознакомиться с библиотекой CUDPP и ее функциями можно в [3].

Зададим в качестве входа cudppScan созданный нами ранее массив (Рис. 4), каждый элемент i которого содержит количество вокселов сетки сцены, занимаемых AABB треугольника с номером i. Тогда на выходе получим *таблицу смещений* в ТС-массиве для каждого потока и размер (в ячейках) ТС-массива (Рис. 5).

Напомним, что занесение данных в ТС-массив производится в режиме параллельной (относительно треугольников) обработки (Рис. 2, Ядро 4), где каждый поток, отвечающий за свой полигон, осуществляет запись, начиная с некоторого смещения, задаваемого таблицей смещений (Рис. 5).

Рассмотрим *алгоритм определения номеров ячеек сетки, занимаемых некоторым треугольником* с нормалью **n** к его поверхности. Спроецируем сетку сцены и треугольник на одну из плоскостей XY, ZY или XZ мировой системы координат в зависимости от того, какая из координат n_z, n_x или n_y (соответственно) нормали является большей по модулю. Такой выбор плоскости позволяет исключить ситуацию, когда проекцией треугольника будет отрезок.

Пусть, например, n_z – максимальная по модулю координата нормали **n**. Тогда в качестве плоскости проецирования будет выбрана плос-

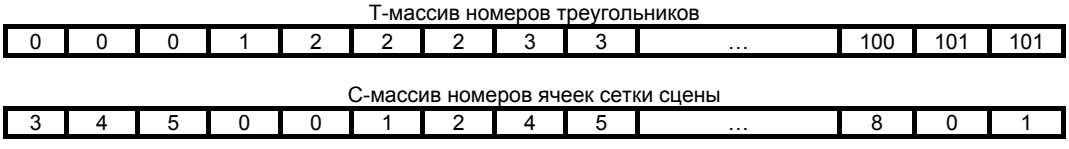


Рис. 3. ТС-массив

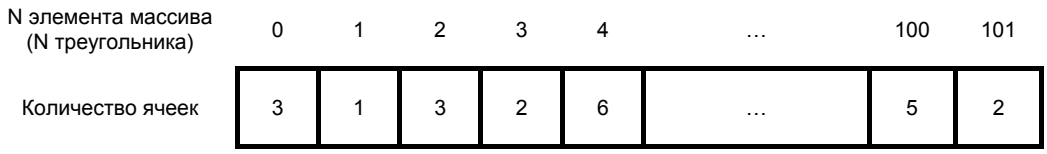


Рис. 4. Информация о количестве вокселов сетки, занимаемых каждым треугольником

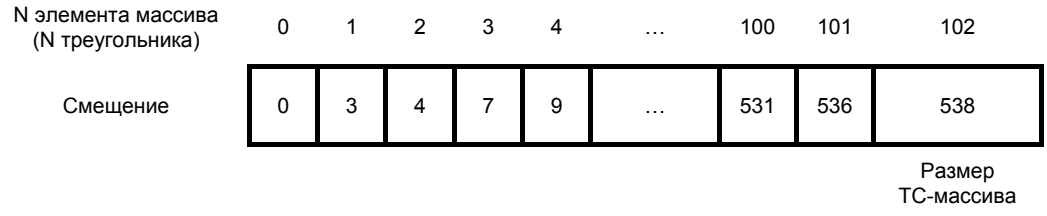
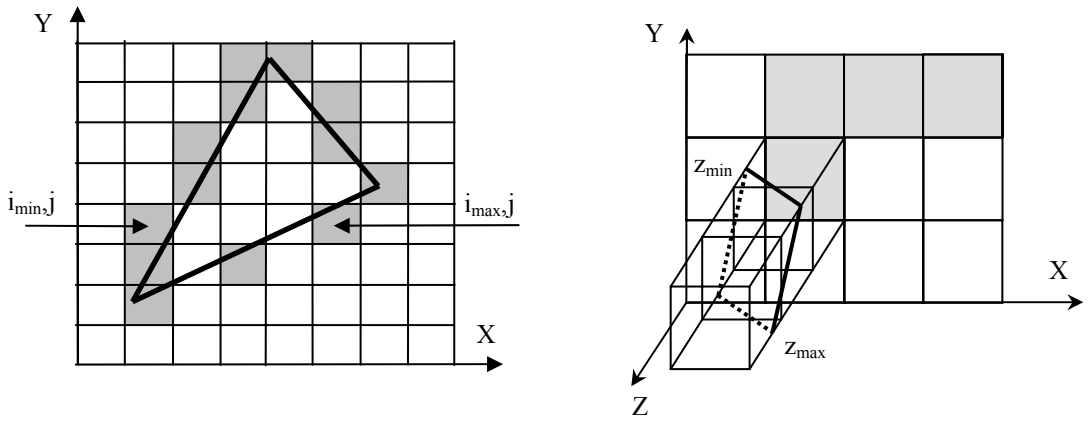


Рис. 5. Таблица смещений начала данных о треугольнике в ТС-массиве



а) б)

Рис. 6. Определение ячеек сетки, занимаемых треугольным полигоном

кость XY. Проведем "растеризацию" полученного треугольника-проекции, в ходе которой определим для каждого слоя j ячеек по оси Y индексы i_{min} и i_{max} минимальной и максимальной ячеек по X, пересекающихся с треугольником (Рис. 6 а, искомые ячейки выделены).

Далее, для каждого индекса i из $[i_{min}, i_{max}]$ каждого слоя j мы должны определить "отрезок" из ячеек, расположенных параллельно оси Z и пересекающихся с плоскостью треугольника (Рис. 6). Для параллельных оси Z ребер каждого такого "отрезка" мы получим четыре (это обусловлено сделанным ранее выбором плоскости проецирования) точки пересечения с плоскостью треугольника – по одной точке для каждого ребра. Среди полученных точек выберем две: с минимальной z_{min} и максимальной z_{max} z-координатой (Рис. 6 б). По z_{min} и z_{max} вычислим минимальный k_{min} и максимальный k_{max} индексы ячеек "отрезка", которые пересекают треугольник. И, наконец, в рассматриваемом "отрезке" для всех ячеек с индексами по Z от k_{min} до k_{max} по формуле (1) определим номера, которые в паре с номером треугольника запишем в TC-массив.

Выполнив Ядро 4 (Рис. 2) мы получаем заполненный TC-массив, отсортированный по номерам треугольников (Рис. 3). Такое представление неудобно для дальнейшего использования. Действительно, зная номер ячейки сетки сцены, необходимо делать полный перебор C-массива, чтобы найти индексы, обращаясь по которым в T-массив, уже можно будет составить список принадлежащих этой ячейке треугольников. Чтобы повысить эффективность обработки информации в процессе трассировки с использованием регулярных сеток, нужно перейти от представления TC-массива, где пары треугольник/ячейка упорядочены по номерам треугольников, к представлению ячейка/треугольник, т.е. упорядочить пары относительно номеров ячеек. Для этого можно применить функцию cudppSort параллельной сортировки массивов из библиотеки CUDPP. Функция cudppSort использует алгоритм поразрядной сортировки (*radix sort*), описанный в [4]. На вход cudppSort подаются массив ключей и массив данных, которые содержат одинаковое количество элементов. При этом считается, что оба массива взаимосвязаны, а именно: i-ый элемент первого массива образует с i-ым элементом второго массива пару (ключ, информация). Функция производит сортировку ключевого массива и аналогично изменяет индексы элементов массива данных. На выходе имеем пару массивов, отсортированных по ключам.

Если в функцию cudppSort в качестве ключевого массива подать C-массив, а в качестве массива данных – T-массив, то в результате мы получим TC-массив, отсортированный по номерам ячеек, который будем называть *CT-массивом*, или *массивом элементов ячейка/треугольник* (Рис. 7).

Кроме CT-массива нам необходимо составить таблицу смещений, по которой мы для любой ячейки с номером i сможем определить индекс первого вхождения этой ячейки в CT-массиве, и таблицу заполнения ячеек, где будет

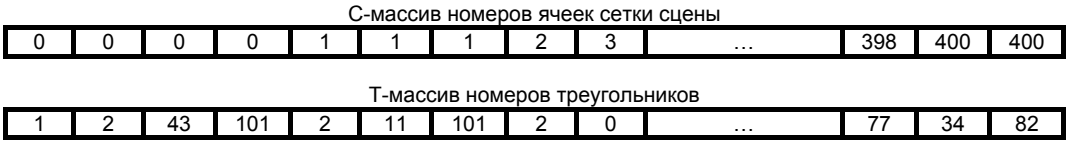


Рис. 7. CT-массив

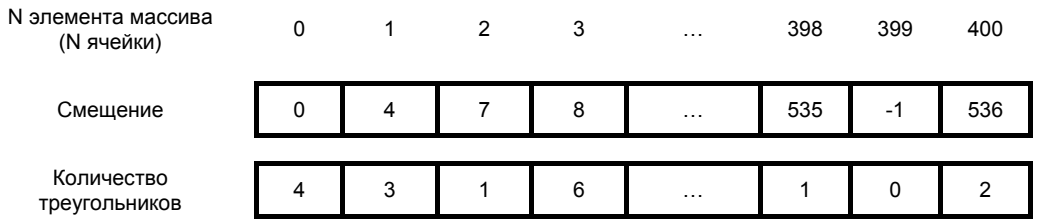


Рис. 8. Таблица смещений начала данных о ячейке в CT-массиве и таблица заполнения ячеек

записано количество треугольников, принадлежащих каждой ячейке (Рис. 8). Эта задача решается с помощью параллельного (относительно ячеек) бинарного поиска в C-массиве номеров ячеек (Рис. 2, Ядро 5). Для ячеек, которые не содержат ни одного треугольника и, следовательно, отсутствуют в CT-массиве, смещение можно задать равным -1, а количество треугольников равным 0.

Таким образом, мы описали следующий параллельный **алгоритм составления таблиц данных о заполнении вокселов сетки:**

1. расчет числа ячеек, занимаемых каждым треугольником;
2. составление таблицы смещений начала записей для каждого треугольника в TC-массиве и выделение памяти под TC-массив;
3. составление TC-массива, состоящего из пар (номер треугольника, номер пересекаемой им ячейки), отсортированного по номерам треугольников;
4. сортировка TC-массива по номерам ячеек с получением CT-массива пар (номер ячейки, номер пересекающего ее треугольника);
5. заполнение таблицы смещений для каждой ячейки в CT-массиве и таблицы количества треугольников в каждой ячейке.

В результате выполнения трех этапов предложенного алгоритма построения и заполнения сетки мы получили набор данных (CT-массив, таблицы смещений и заполнения ячеек), полностью описывающих регулярную сетку трехмерной виртуальной сцены и необходимых для дальнейшей трассировки лучей.

Одним из основных преимуществ изложенного алгоритма параллельного построения регулярной сетки на GPU, за исключением поддержки режима реального времени, является экономия памяти видеоадаптера, поскольку отсутствует предварительное выделение памяти по среднему, заранее рассчитываемому числу ячеек, которые занимает один треугольник, и свободные ячейки вообще не записываются в CT-массив.

3. Трассировка вторичных лучей с помощью ускоряющей структуры

Имея регулярную сетку сцены, нужно для каждого луча, пересекающего AABB сцены, вычислить последовательность ячеек, через которые он проходит, и тестировать его на пересечение только с полигонами, принадлежащими этим ячейкам. Этот процесс может иметь два возможных окончания: либо мы находим в какой-либо ячейке точку пересечения с ближайшим от начала луча треугольником этой ячейки (при условии, что данная точка лежит внутри самой ячейки), либо, пройдя все ячейки, выясняем, что таких пересечений нет, а, следовательно, данный луч несет свет с интенсивностью, равной цвету фона. Для обработки каждого луча будем использовать следующий **алгоритм трассировки одного луча с помощью регулярной сетки:**

1. чтение списка номеров треугольников, пересекающихся с текущей ячейкой, из CT-массива (число треугольников в списке опреде-

ляется из таблицы количества треугольников, составленной в п. 2); если ячейка пуста, т.е. число пересекающих ее треугольников равно 0, переходим к шагу 3;

2. проверка каждого треугольника из прочтенного списка на пересечение с лучом и поиск ближайшей к началу луча точки P пересечения, лежащей внутри воксела; если такая точка найдена, алгоритм переходит к шагу 4;

3. определение следующей по лучу ячейки и переход к шагу 1;

4. вычисление интенсивности освещения в точке P (обычно вычисление выполняется по модели Уиттеда [5]); на данном шаге обработка луча завершается и, если нужно, он разделяется на вторичные лучи следующего уровня.

При реализации приведенного алгоритма трассировки луча на GPU особый интерес в нем представляет пункт 3. Поэтому далее рассмотрим алгоритм нахождения следующего по лучу воксела регулярной сетки сцены, который оптимизирован для реализации на графических процессорах с поддержкой архитектуры CUDA и учитывает особенности выполнения расчетов на таких процессорах.

Для начала разберем двумерный случай. Пусть луч (Рис. 9) начинается внутри ячейки с индексами i, j в точке $P(P_x, P_y)$ и имеет направляющий вектор **D**(x,y): x>0, y≥0. В зависимости от соотношения углов α и φ луч может перейти из начальной ячейки в одну из трех ячеек: (i+1, j) при α<φ, (i, j+1) при α>φ или (i+1, j+1) при α=φ. В перечисленных условиях углы α и φ можно заменить на их тангенсы (поскольку тангенс – возрастающая функция), вычисляемые по формулам:

$$\operatorname{tg}\alpha = y/x, \qquad \operatorname{tg}\varphi = (1 - k_y) / (1 - k_x),$$

$$k_x = \frac{P_x - B_{\min,x}}{A} - i, \qquad k_y = \frac{P_y - B_{\min,y}}{A} - j,$$

где (k_x, k_y) – относительная точка начала луча, координаты которой выражены в долях ячейки, A – размер ячейки, B_{min} – минимальная точка сетки (координаты ячейки (0,0)). Заметим, что $k_x, k_y < 1$, поскольку точку P, лежащую на верхней или правой границе некоторой ячейки, мы будем относить к соседней верхней или правой ячейке соответственно.

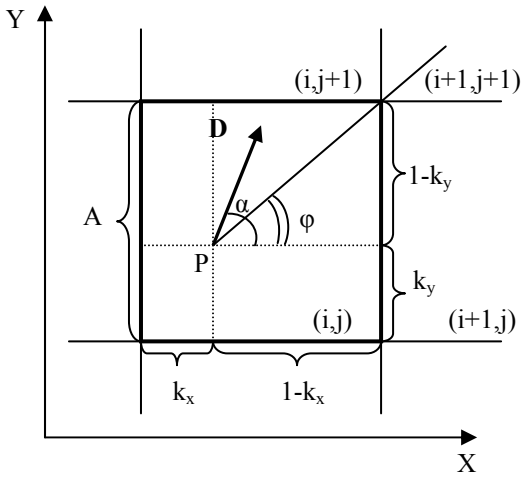


Рис. 9. Луч и ячейка в случае 2D

Таким образом, схему расчета индексов i′ и j′ следующей ячейки можно записать в виде:

if (tg φ > tg α) { i' = i + 1; j' = j; }
else if (tg φ < tg α) { i' = i; j' = j + 1; }
else { i' = i + 1; j' = j + 1; }

Однако одной из особенностей параллельных алгоритмов для GPU с поддержкой архитектуры CUDA является необходимость избегать применения в них ветвлений (условных переходов). Так, при выполнении трассировки лучей на ядрах GPU с обработкой каждого луча в отдельном потоке, если код алгоритма содержит ветвление и для одной части потоков в исполняемой группе (*warp*, [1]) нужно выбирать первую ветвь, а для другой – вторую, то для всех потоков из данной группы будут выполнены обе ветви. А именно, сначала вся группа будет осуществлять проход по первой ветви с блокированием несоответствующих этой ветви потоков. Другой раз – наоборот, алгоритм будет пройден для всей группы по второй ветви с блокированием потоков, относящихся к первой (Рис. 10). Таким образом, будет наблюдаться так называемое расхождение потоков в исполняемой группе (*warp divergence*, [1]), а группы расходящихся потоков будут выполняться последовательно (Рис. 10) до той точки, в которой ветви вновь сойдутся. Следовательно, общее время выполнения блока if/else будет равно сумме времен выполнения обеих его ветвей.

Если же ветвлений будет много и/или они будут вложенными, то это может привести

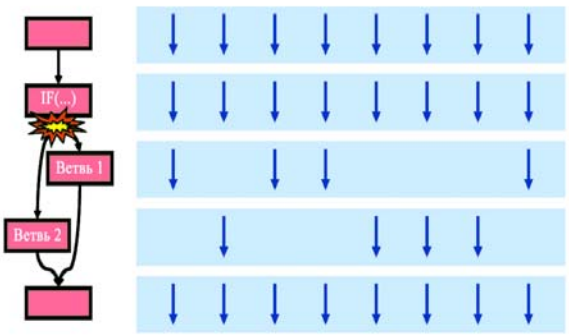


Рис. 10. Расхождение потоков при ветвлении

к значительному падению скорости выполнения алгоритма, а значит, будет теряться и весь смысл параллельности в вычислениях.

Для оптимизации расчета индексов i′ и j′ мы предлагаем заменить конструкцию if/else на вычисление формул, одинаковых для всех потоков в исполняемой группе, а значит, не вызывающих расхождение потоков.

Пусть S(x) – функция, принимающая значение 1, если число x – отрицательное, и 0 – в противном случае (использование такой функции обусловлено наличием ее аналога signbit(x) при программировании для GPU с поддержкой архитектуры CUDA). Тогда

$$S(tg\varphi - tg\alpha) = \begin{cases} 0, & \alpha \le \varphi \\ 1, & \alpha > \varphi \end{cases},$$

$$S(tg\alpha - tg\varphi) = \begin{cases} 0, & \alpha \ge \varphi \\ 1, & \alpha < \varphi \end{cases}.$$

Исходя из этого, ячейку (i′, j′), следующую по лучу за (i, j), можно определить как

$$i' = i + 1 - S(tg\,\varphi - tg\,\alpha) = i + 1 - S\left(\frac{x(1-k_y) - y(1-k_x)}{x(1-k_x)}\right),$$

$$j' = j + 1 - S(tg\,\alpha - tg\,\varphi) = j + 1 - S\left(\frac{y(1-k_x) - x(1-k_y)}{x(1-k_x)}\right).$$

Поскольку по условию $x > 0$, а $k_x < 1$, то знаменатели аргументов функции S не влияют на их знак, следовательно, их можно не учитывать:

$$i' = i + 1 - S(x(1-k_y) - y(1-k_x)),$$
$$j' = j + 1 - S(y(1-k_x) - x(1-k_y)).$$

Отметим, что после того, как мы перешли к таким формулам вычисления i′ и j′, ранее недопустимое значение x=0 стало возможным, а значит, диапазон допустимых векторов **D** расширился до **D**(x,y): x≥0, y≥0.

Аналогично можно рассмотреть случаи, при которых вектор **D** расположен во втором, третьем и четвертом квадрантах. Объединяя эти случаи, получим формулы для i′ и j′ при произвольном векторе **D**:

$$\begin{aligned} i' &= i + k_i(1 - S(|x|f_y - |y|f_x)), \\ j' &= j + k_j(1 - S(|y|f_x - |x|f_y)), \\ k_i &= 1 - 2S(x),\ k_j = 1 - 2S(y), \\ f_x &= 1 - S(x) - k_i k_x,\ f_y = 1 - S(y) - k_j k_y. \end{aligned} \quad (2)$$

В трехмерном случае рассмотрим проекции воксела и луча на плоскости XY, ZY и XZ системы координат WCS (Рис. 11). Каждая из этих проекций будет представлять собой уже описанный выше двумерный случай (с точностью до обозначения координатных осей). При переходе из одной ячейки в другую любой из индексов i, j, k будет изменяться, если он изменяется в двух из трех проекций XY, ZY и XZ. Для i – это проекции XY и XZ, для j - XY и ZY, а для k - ZY и XZ. Так, например, на Рис. 11 при переходе в следующую ячейку индекс j увеличится на 1. Учитывая вышесказанное, можно выписать формулы для получения индексов i′, j′ и k′ следующей ячейки, аналогичные формулам (2):

$$\begin{aligned} i' &= i + k_i((1 - S(|x|f_y - |y|f_x)) \cdot (1 - S(|x|f_z - |z|f_x))), \\ j' &= j + k_j((1 - S(|y|f_x - |x|f_y)) \cdot (1 - S(|y|f_z - |z|f_y))), \\ k' &= k + k_k((1 - S(|z|f_y - |y|f_z)) \cdot (1 - S(|z|f_x - |x|f_z))), \end{aligned}$$

где $k_i = 1 - 2S(x)$, $k_j = 1 - 2S(y)$, $k_k = 1 - 2S(z)$, $f_x = 1 - S(x) - k_i k_x$, $f_y = 1 - S(y) - k_j k_y$, $f_z = 1 - S(z) - k_k k_z$.

Отметим, что для выполнения вычислений по указанной схеме необходимо для каждой ячейки определять точку P начала луча R в данной ячейке. Точка P в первом вокселе, пересекаемом лучом R по ходу его распространения, совпадает с точкой пересечения R и AABB сцены (для первичного луча), ближайшей к началу луча O, или непосредственно с началом луча (для вторичного луча). Для всех остальных вокселов ее нужно рассчитывать как точку выхода из предыдущей ячейки с индексами (i, j, k).

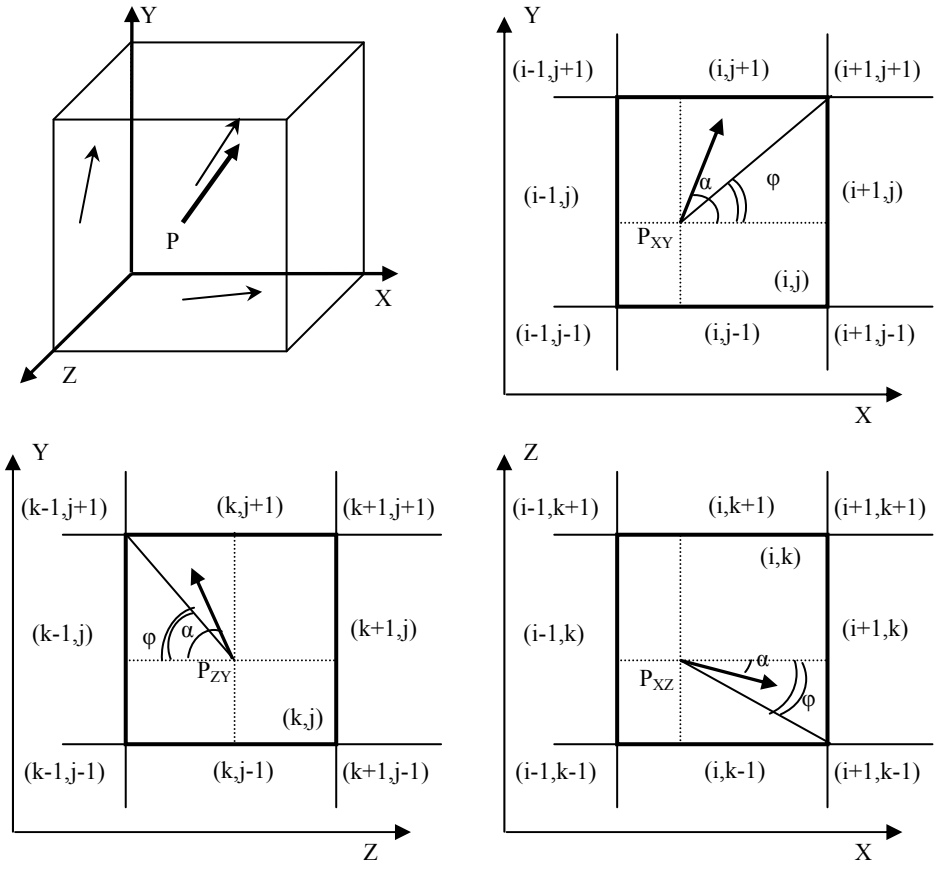


Рис. 11. Проекции луча и ячейки на координатные плоскости

Литература

1. NVIDIA CUDA C Programming Guide. Version 4.2. – http://developer.download.nvidia.com/compute/DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf (дата обращения 10.09.2012)
2. Ladner R.E., Fischer M.J. Parallel Prefix Computation // Journal of the ACM. – 1980. – Vol. 27, no. 4. – Pp. 831-838.
3. CUDA Data Parallel Primitives Library. General-Purpose Computation on Graphics Hardware. – http://gpgpu.org/developer/cudpp (дата обращения 10.09.2012).
4. Дональд Кнут. Искусство программирования. 2-е издание. – М.: «Вильямс», 2007. – т. 3.
5. Whitted T. An Improved Illumination Model for Shaded Display // Comm. ACM. – 1980. – June. – Vol. 23, no. 6. – Pp. 343-349.

Мальцев Андрей Валерьевич. Научный сотрудник Центра визуализации и спутниковых информационных технологий НИИСИ РАН. Окончил Московский государственный институт радиотехники, электроники и автоматики в 2008 году. Кандидат физико-математических наук. Автор 18 научных публикаций. Область научных интересов: компьютерная графика, системы виртуальной реальности, информационные технологии. Специалист в области компьютерной графики.
E-mail: avmaltcev@mail.ru

Михайлюк Михаил Васильевич. Заведующий отделом программных средств визуализации Центра визуализации и спутниковых информационных технологий НИИСИ РАН. Окончил Московский государственный университет им. М.В. Ломоносова в 1975 году. Доктор физико-математических наук, профессор. Автор более 120 научных публикаций. Область научных интересов: компьютерная графика, системы виртуальной реальности, информационные технологии.
E-mail: mix@niisi.ras.ru