УДК 004.021

Структуры данных и методы визуализации плотных облаков точек

И.Г. Рекун[1]

[1]Московский физико-технический институт

Введение

На сегодняшний день, полностью исключить взаимодействие музейных экспонатов с окружающей средой, для предотвращения их старения, не представляется возможным. Для их сохранения требуется периодическая реставрация. Методами фотограмметрии можно получить точную копию поверхности объекта. Для более полного представления о состоянии экспоната, съемка может производиться в различных спектрах. Результатом 3D сканирования и фотограмметрии являются плотные облака точек, интерактивная визуализация которых является довольно вовлеченной задачей.

На текущий момент большинство предложенных решений используют архитектуру клиент-сервер, либо прибегают к агрессивным оптимизациям, в ущерб точности визуализации. В докладе предлагается система, основанная на иерархических структурах данных, обеспечивающая интерактивную работу с результатами музейной фотограмметрии.

В докладе приведены способы сжатия плотных облаков точек, структуры данных, обеспечивающие эффективную обработку с точки зрения кэш памяти процессора, и методы визуализации в реальном времени в среде выполнения браузера, а также альтернативы с использованием вычислительных шейдеров на видеокартах.

Обзор среды выполнения

Изначально программное решение разрабатывалось для выполнения в веб браузере на стороне клиента. Данный шаг обоснован снижением требований по вычислительной мощности к серверной части. В свою очередь, визуализация на машине клиента сопряжена с требованиями по совместимости с различными версиями браузеров и, возможно, устаревшими видеокартами и графическими драйверами.

Для обеспечения работы в условиях максимально широкого спектра конфигураций клиентских машин было решено отказаться от использования возможностей шейдеров, доступных на современном оборудовании и разработать систему для “программного рендеринга” (рендеринга без использования аппаратного ускорения).

Приемлемая скорость выполнения кода обеспечивается путем двухэтапной компиляции с использованием промежуточного вывода виртуальной машины LLVM (Low Level Virtual Machine) языка C и последующей его трансляцией в строго типизированный код на языке javascript. В общем случае код выполняется в целом лишь в два раза медленнее по сравнению с машинным [1].

Структуры данных

Естественным представлением результатов фотограмметрии и 3D сканирования являются облака точек (рис. 1).

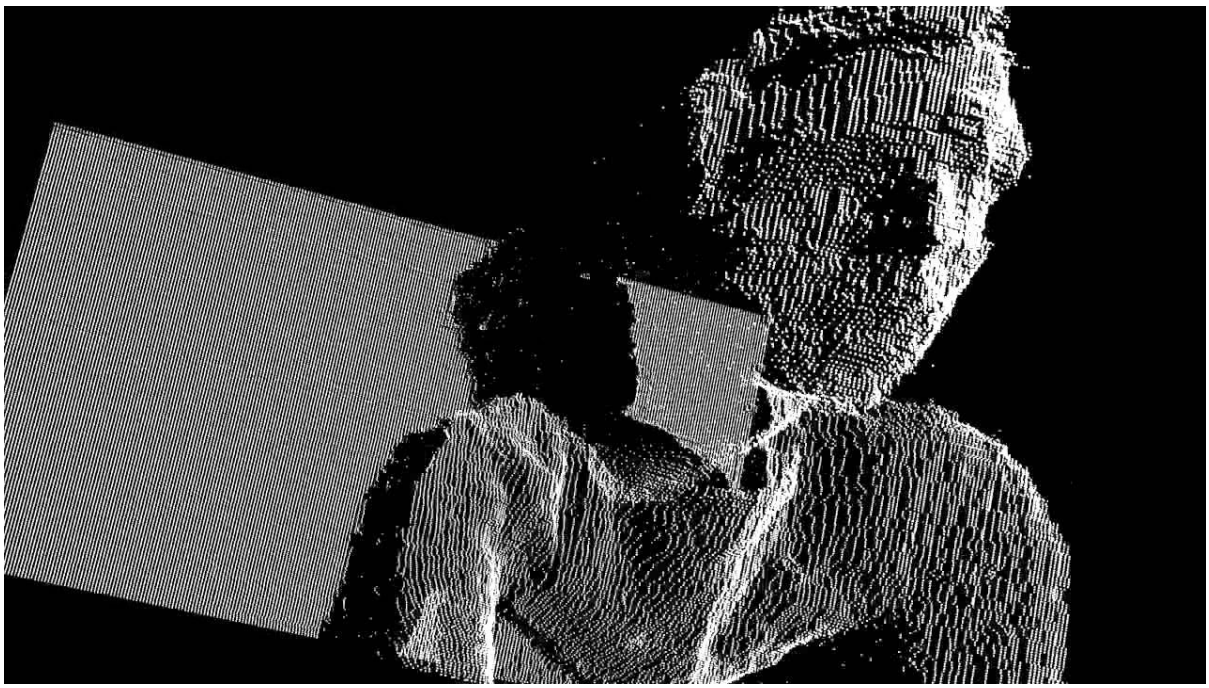

Рис. 1 Облако точек полученное с помощью сенсора XBOX Kinect

В большинстве случаев brute-force реализация, при наличии производительной видеокарты, справляется с поставленной задачей. При необходимости применяются методы разбиения на уровни детализации или стохастика при выборе подмножеств отображаемых точек [2].

При использовании данных фотограмметрии для последующей реставрации музейных объектов, неточности, возникающие как следствие приближенных алгоритмов визуализации, являются значимым фактором.

Справится с артефактами стохастического семплирования точек частично помогает предварительная сортировка облака гильбертовой кривой (рис. 2).

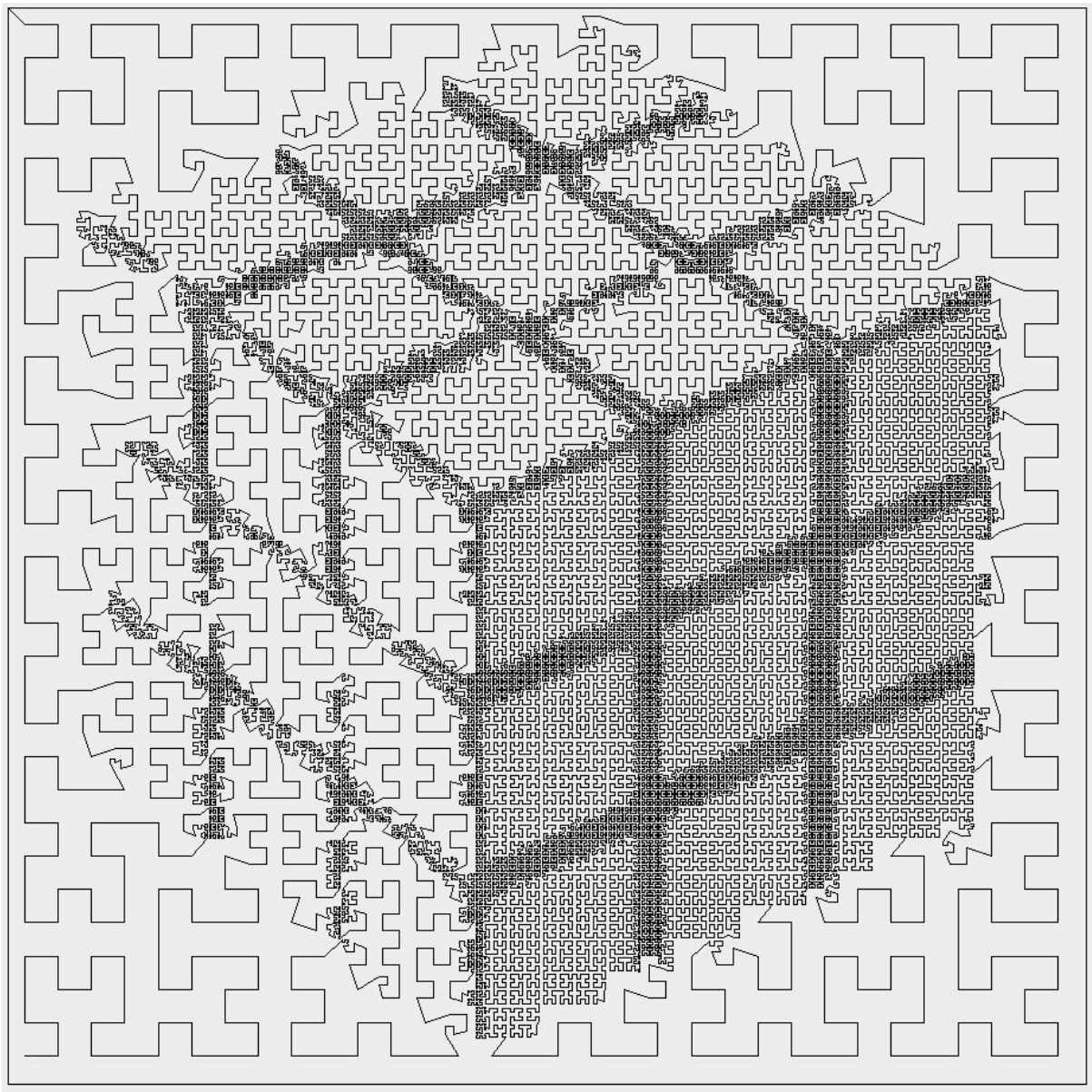

Рис. 2 Пространственная гильбертова кривая

Таким образом при равномерном исключении точек вдоль кривой минимизируется количество возникающих разрывов. Тем не менее, результат сильно зависит от балансирующих свойств структуры данных, используемой для создания последовательных уровней детализации.

Наиболее перспективным себя показал подход с преобразованием облака в воксельное октодерево [3], и последующей упаковкой в разреженное октодерево[4].

С учетом возросшего объема данных (съемка в нескольких спектральных диапазонах), требуется дополнительное сжатие модели для экономии сетевого траффика при передачи данных с сервера на машину клиента.

Кодирование лишь значимых узлов дерева (Рис. 3) позволяет добиться значительного сокращения размера файлов моделей [5]. Для этого требуется определить фиксированный порядок обхода узлов октодерева и соблюдать его при преобразовании облака точек. Геометрическая информация каждого узла занимает 1 байт, каждый бит показывает, наличие потомка у данного узла (есть ли точки в данном октанте).

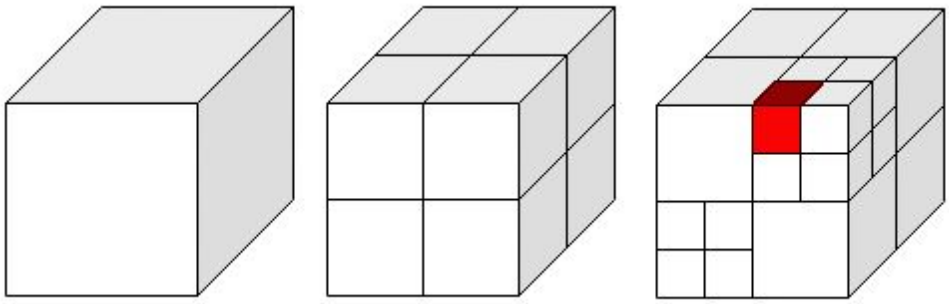

Рис. 3 Октодерево, содержащее данные лишь в закрашенном узле.

Результатом кодирования данного дерева, при обходе по часовой стрелке сверху вниз, будет следующая последовательность из двух байт:

00010000, 1000000

Побочным эффектом сжатия является усложнение доступа к потомкам, адрес которых на уровне определяется как сумма всех выставленных бит до бита интересующего нас узла. Данная проблема решается хранением заранее рассчитанных сумм с определенным выбранным шагом (256, 512, 1024 байт). Данный шаг ограничивает максимальное количество байт, которые следует проверить, чтобы определить адрес потомка.

Одним из важнейших моментов является распаковка узла и нескольких последующих уровней в простой массив, при достижении достаточно малого размера его экранной проекции. Необходимость обращаться к списку предрассчитанных сумм в процессе рендеринга катастрофически снижает скорость работы алгоритма, бутылочным горлышком которого является L1 кэш процессора. Подобное преобразование позволяет сохранить большинство обращений внутри L1 кэша при выполнении наиболее трудоемкого этапа алгоритма.

Методы визуализации

Основная идея базируется на агрессивном отсечении скрытых узлов и асимптотических свойствах используемых структур данных. В общем случае, эффективность деревьев при поиске значительно зависит от их сбалансированности, однако в данном случае мы можем строго ограничить глубину чтения дерева не нарушая точности визуализации.

Отсечение скрытых узлов выполняется с использованием вспомогательного bitmask буфера (Рис. 4).

Рис. 4 Однобитная маска перекрытия заполненных узлов модели

Перед началом рендеринга каждого кадра маска обнуляется и корневой узел дерева помещается на стек. Далее, для каждого потомка проверяется соответствие критерию растеризации. Если размер проекции узла меньше чем 1 экранный пиксель либо узел не имеет потомков, то экранная проекция узла запекается в bitmask буфер. Если критерий растеризации не соблюдается, то проверяется, является ли узел видимым согласно текущему состоянию буфера перекрытия. Если нет - дальнейший спуск по всей ветке не выполняется и алгоритм переходит к следующему потомку.

Возможность обрабатывать узлы октодерева в порядке от ближнего к дальнему, в зависимости от позиции камеры, позволяет однозначно скрывать целые ветви и эффективно ограничить глубину чтения дерева вне зависимости от того, насколько хорошо оно сбалансировано.

Реализация алгоритма на GPU

Порядок обхода дерева является ключевым моментом алгоритма, что в некоторой степени ограничивает варианты параллелизации. Учитывая, что сложность алгоритма растет от разрешения экрана а не от сложности геометрии сцены и количества точек, наиболее эффективной будет возможность разбиения в экранном пространстве (screen space). Вопрос оптимального хранения и чтения древовидной структуры данных хорошо рассмотрен в [6].

Наиболее простым вариантом является гибридный алгоритм. Bitmask буфер заменяется на квaддерево (quadtree) вместо массива, тест на перекрытие сводится к проверке заполнен ли узел квaддерева. В очередь верхних узлов квaддерева добавляются видимые узлы октодерева модели и данный шаг последовательно выполняется до достижения критерия растеризации.

Альтернативным вариантом является модификация алгоритма, имеющая порядок роста O(W*H*C), где W - ширина экрана, H - высота экрана, C - константа, определяющая максимальный размер очереди узлов модели. Выбор C позволяет смещать баланс в сторону точности алгоритма или скорости его выполнения. На этап проверки перекрытия включается frustum тест для определения содержащихся в ячейке квадерева узлов октодерева. Выбор малого C может привести к заполнению очереди и потере значимых узлов.

Оптимальным вариантом является выбор первоначального множества узлов на процессоре и дальнейшая обработка на GPU. Таким образом на видеокарте требуется лишь реализовать шейдер, делящий квадерево и распределяющий узлы модели в очереди новых квадрантов.

Литература

1. Alan Zakai, Emscripten: an LLVM-to-JavaScript compiler, 2011
2. W. Schroeder, K. Martin, and B. Lorensen, Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 4th Edition, 2006
3. Donald Meagher, Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer, 1980.
4. Laine Samuli, Tero Karras, Efficient Sparse Voxel Octrees – Analysis, Extensions, and Implementation, 2010.
5. Myung Geol Choi, Eunjung Ju, Jungwoo Chang, Young J Kim, Jehee Lee,Linkless Octree Using Multi-Level Perfect Hashing, Pacific Graphics, 2009
6. Matt Pharr, Randima Fernando, GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation, Addison-Wesley Professional, 2005.