Глобальное освещение с помощью октантных текстур

К.А. Востряков
Томский государственный университет, Томск, Россия
vconst@inbox.ru

Аннотация

Эта статья посвящена ускорению финального сбора в методе фотонных карт. Финальный сбор состоит из трассировки луча и запроса облучения из фотонной карты. Сочетание kd-tree для геометрии и быстрого алгоритма пересечения луча с треугольником позволяет достичь скорости трассировки порядка 1M лучей/с. Следовательно, запрос облучения из фотонной карты должен быть также быстрым. Поиск в kd-tree фотонной карты в несколько раз медленнее, чем трассировка луча, поэтому предлагается использовать октантные текстуры. Это позволило ускорить запрос облучения на порядок, что дало ускорение финального сбора в 3 раза. Кроме того, предложен метод, который позволяет сохранять фотоны сразу в октантную текстуру, не используя kd-tree фотонной карты.

Ключевые слова: *global illumination, photon map, octree texture, final gathering, ray tracing.*

1. ВВЕДЕНИЕ

В работах [2] и [5] независимо предложено использовать октантные трехмерные текстуры для моделирования сложных трехмерных объектов, создание двумерной текстурной развертки, для которых отнимает значительную часть времени при создании модели, а также для текстурирования неявно заданных поверхностей. Christensen и Batali [4] использовали октантные текстуры для хранения облучения, полученного методом фотонных карт [8, 9]. В огромных сценах, которые они использовали, kd-tree фотонной карты, было в несколько раз больше доступной памяти, поэтому по kd-tree они строили октантную текстуру, которая может быть эффективно кэширована. 3D MIP map структура октантной текстуры позволяет квадролинейную фильтрацию. Christensen и Batali сначала сохраняли фотоны в kd-tree, вычисляли облучение в точках столкновения фотонов с поверхностями, используя запросы *N* ближайших фотонов в kd-tree, а затем сохраняли полученные значения облучения в октантной текстуре. В заключении они заметили, что эффективнее было бы сразу сохранять фотоны в текстуре и отказаться от kd-tree, но не привели способа как это сделать. Данная работа описывает алгоритм, который позволяет сохранять фотоны сразу в октантную текстуру, но акцент в использовании октантных текстур сделан не на кэширование, а на скорость финальной сборки, которая состоит из трассировки луча и запроса значения облучения.

Автор реализовал kd-tree (сбалансированное по стоимости bsp) [6, 7] для ускорения трассировки лучей. Для пересечения луча с треугольником был выбран алгоритм оптимизированной проекции, который предложил Wald [17], этот алгоритм в 2 раза быстрее алгоритма Möller-Trumbore [12], но требует дополнительно 48 байт на треугольник. Такое ускорение достигается за счет использования предвычисленных значений в дополнительной памяти и более эффективного использования кэша процессора. Вместе эти алгоритмы дают очень высокую производительность трассировки луча, поэтому для быстрой финальной сборки необходим также быстрый доступ к значению облучения.

2. ПРЕДЫДУЩИЕ МЕТОДЫ

Christensen [3] заметил, что скорость доступа к фотонной карте сильно замедляет финальную сборку, и предложил предвычислить значения облучения в точках столкновения фотонов с поверхностями, а после искать в ходе финальной сборки ближайший фотон, а не *N* ближайших. Такая кусочно-постоянная аппроксимация облучения в 5-7 раз ускорила финальную сборку. Но даже использование метода предложенного Christensen запрос из kd-tree фотонной карты в 3 раза медленнее, чем трассировка луча финальной сборки, если трассировка луча реализована достаточно быстро, причем, чем больше kd-tree фотонной карты, тем медленнее запрос. Если же реализовать пакетную трассировку лучей, с использованием SIMD инструкций процессора [17], что дополнительно ускорит трассировку в 2-3 раза, то требования к скорости запроса облучения из фотонной карты еще более возрастают.

Wald, Günther, Slusallek [18] предложили сбалансировать kd-tree фотонной карты, используя эвристику стоимости запроса, они сообщают о 1.3-3.5 ускорении скорости запроса.

Larsen, Christensen [10] использовали несколько фотонных карт, разделив сцену на несколько объектов, что уменьшило время балансировки и запроса. Этот же прием можно применить и к октантным текстурам.

Tabellion и Lamorlette [16], заметив проблему, предложили текстуры (radiosity maps), которые наложены на все поверхности, их разрешения пропорциональны размерам объекта либо в трехмерном пространстве, либо в пространстве экрана. Отображение поверхностей трехмерных объектов на двумерную текстуру может порождать определенные сложности [2, 5], которые не возникают, если использовать трехмерную октантную текстуру, которая не требует параметризации.

Другие методы [1, 11, 13] хранения фотонов на поверхностях, а не в kd-tree, не могут эффективно работать с очень сложными поверхностями, поскольку каждая маленькая тонкая поверхность будет иметь собственное значение облучения, даже если оно достаточно гладкое.

В этот работе предлагается отказаться от использования kd-tree для хранения облучения, поскольку использование октантной текстуры, может дать ускорение запроса более чем на порядок.

3. ОПИСАНИЕ АЛГОРИТМА

Представляемый алгоритм моделирования глобального освещения состоит из трех частей. Первая часть, вокселизация поверхностей сцены в октантной текстуре, т.е. нахождение вокселей, которые пересекаются с поверхностями сцены. Вторая часть – это трассировка фотонов из источников света и сохранение значений их энергии в трехмерной текстуре. Третья часть – финальная сборка, получение значений облучения в точках пересечения лучей финальной сборки.

3.1 Структура данных

Данные организованы в виде разряженного октантного дерева. Каждый узел дерева имеет указатель на дочерние узлы, а листовой узел еще и указатель на воксель. Каждый воксель может быть пустым или содержать значение облучения, усредненную нормаль к поверхности. Если воксель пересекает поверхности с сильно отличающимися нормалями, то создаются дополнительные воксели, которые соединяются в виде связного списка. Структура данных имеет вид:

```
struct Voxel
{
    float irrad[3];
    float normal[3];
    float weight;
    Voxel *next;
};
```

Заметьте, что child указывает на 8 подряд расположенных объектов Node.

```
struct Node
{
    Node *child;
    Voxel *voxel;
};
```

Переменная веса может быть использована при построении текстуры из kd-tree фотонной карты, как описано в [4], а так же при построении MIP map уровней.

3.2 Создание поверхностей в октантной текстуре

Для того чтобы сохранять в октантную текстуру не значения облучения, полученные с помощью kd-tree фотонной карты, а сразу фотоны, нужно сохранять их с большим радиусом, таким образом, что каждую точку поверхности будет покрывать своим влиянием сразу несколько фотонов. Большой радиус влияния требует учет поверхностей, на которые мы будет сохранять эти значения. Если этого не сделать, то значения будут сохраняться и на поверхностях с противоположенной нормалью, а также количество узлов дерева резко возрастет, поскольку значения будут записываться не только на поверхностях, но и рядом с ними. Поэтому автором предложено вокселизовать поверхности сцены в октантной текстуре, т.е. найти те воксели, которые пересекаются с поверхностями сцены, и записать в них усредненную нормаль к поверхности. Если в одном воксели существуют нормали различающиеся больше, чем на 45°, то создается еще один воксель, который присоединяется к уже существующему как следующий элемент списка. Таким образом, когда значение энергии фотона будет сохраняться в вокселях, будет известно проходит ли через него поверхность сцены, и нормаль к этой поверхности.

3.3 Трассировка фотонов и запись

Когда вся сцена вокселизована, то можно начинать трассировать фотоны. Фотоны излучаются из источников света согласно распределению излучения. Находятся точки пересечения фотонов с поверхностями сцены, далее фотоны отражаются или поглощаются в зависимости от функции отражения (BRDF). Энергия фотонов в точках столкновения с поверхностями записывается в октантную текстуру следующим образом: вычисляются все воксели, которые пересекаются с объемом фильтра и имеют подходящую нормаль. В эти воксели добавляется энергия фотона, используя фильтр со следующим ядром [13, 14]:

$$E = \frac{\Phi}{nh^2}\sum_{i=1}^{n} K(\frac{x - x_i}{h}),\ K(u,v) = \frac{3}{\pi}\max(0,(1 - \|x - x_i\|^2)^2)$$

Тогда к каждому вокселю будет добавлено значение

$$\phi = \frac{3}{\pi h^2}\left(1 - \frac{\|x - x_i\|^2}{h^2}\right)^2$$

Псевдокод алгоритма:

```
//point_side – ширина фильтра, power – мощность фотона
//pos – точка столкновения фотона с поверхностью
//incident – вектор направления падения фотона
node_side = side; // side – сторона корневого узла
node = root; // указатель на корневой узел
node_vmin = center - 0.5*side; // center – центр корневого узла
while (1)  {
 if (is node leaf) {
  for (j = 0; j < 8; j++) {
   child_side = 0.5*node_side; child_vmin = node_vmin;
   for (i = 0; i < 3; i++) if  (j & (1 << i)) child_vmin[i] += child_side;
   if (child node intersect volume)
    stack->push(&node->children[j], child_side, child_vmin);
  }
 }
 else {
  Voxel *voxel = node->voxel;
  while (1) {
   if (dot(voxel->normal, incident) > 0) {
    h = (point_side+v_side)*0.5;
    w = 3/(πh^2) max(0,(1 - ||pos - v_center||^2 / h^2)^2)
    voxel->irrad += power*w;
   }
   if (!voxel->next) break; voxel = voxel->next;
  }
 }
```

```
  if (stack->empty()) break;
  stack->pop(node, node_side, node_vmin);
}
```

3.4 Запрос значения

Для того чтобы получить значение облучения в точке необходимо найти узел и воксель принадлежащий этой точке. Описанный здесь метод значительно проще, чем в [4], он более быстрый и. реализует более простую ближайшую фильтрацию.

Входные данные: точка запроса, нормаль запроса, сторона корневого узла, указатель на корневой узел, центр корневого узла. Возвращаемое значение – облучение в точке запроса. Псевдокод:

```
//pos – точка запроса, normal – нормаль запроса
node_fourth_side = side*0.25; // side – сторона корневого узла
node = root; // указатель на корневой узел
node_center = center; // center – центр корневого узла
while (1) {
 child_center = node_center;
 for (j = 0, i = 0; i < 3; i++)
 if (pos[i] > node_center[i]) {
  j |= 1 << i;
  child_center[i] += node_fourth_side;
 }
 else  child_center[i] -= node_fourth_side;
 if (is node leaf) {
  Voxel *voxel = node->voxel;
  while (1) {
   if (dot(voxel->normal, normal) > 0.7) return voxel->irrad;
   voxel = voxel->next; if (!voxel) break;
  }
  return black_color;
 }
 node_center = child_center; node_fourth_side *= 0.5;
 node = &node->children[j];
}
```

4. РЕЗУЛЬТАТЫ

Все измерения проводились на AMD AthlonXP 2500+. Время вокселизации сцены из 200k треугольников составило 3.5 с. Скорость сохранения фотонов в текстуре очень сильно зависит от глубины текстуры и от ширины фильтра. При увеличении глубины на единицу, скорость сохранения уменьшается вдвое. Трудно точно сравнивать качество и скорость октантной текстуры и kd-tree, поскольку оба методы порождают разные артефакты. Тем не менее, можно сделать следующий вывод. При сохранении фотонов в октантной текстуре сложно добиться максимального качества, которое возможно при использовании kd-tree. При сохранении фотонов в октантную текстуру труднее бороться с граничными смещениями. Хотя в большинстве случаев октантная текстура обеспечивает допустимое качество для финального сбора, при этом с помощью kd-tree фотонной карты аналогичного качества удается достичь за время в несколько раз большее (см. рис. 1). Для больших сцен в октантной текстуре можно сохранять больше фотонов, чем, используя kd-tree, поскольку в случаи октантной текстуры память ограничена созданной воксельной структурой и не возникнет ситуации, когда kd-tree фотонной карты перестанет входить в память. Этот эффект напоминает алгоритм контроля плотности фотонной карты [15], когда энергия фотона, падающего на поверхность с высокой плотностью, перераспределяется по ближайшим фотонам и плотность фотонов не превышает требуемого значения.

Скорость запроса из октантной текстуры > 10 раз быстрее, чем запрос из kd-tree (1-ближайший фотон [3]), что ускоряет прямую визуализацию фотонной карты > 3 раз (см. рис. 1), аналогично ускоряется и финальный сбор.

Скорость авторской реализации финального сбора > 10 раз быстрее, чем финальный сбор в mental ray 3.2 из 3ds max 6. Такое быстродействие было достигнуто благодаря использованию алгоритмов ускоряющих трассировку луча (kd-tree и метода оптимизированной проекции), а также октантных текстур.

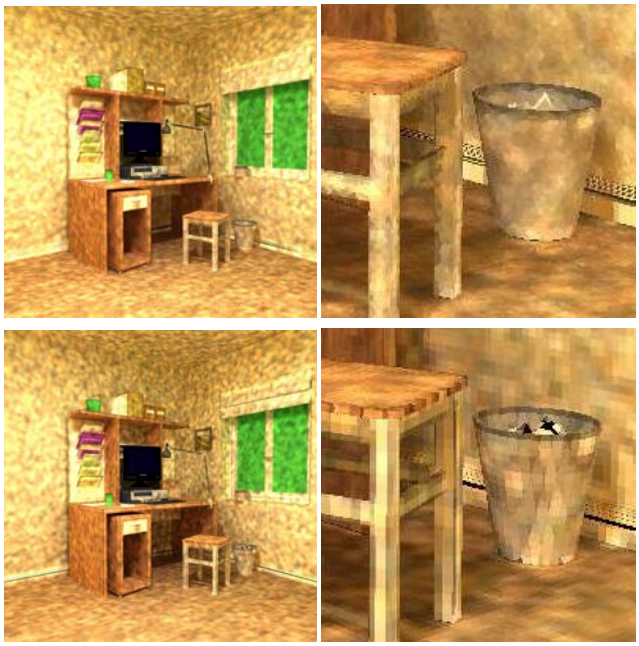

Рис. 1. 1024x1024 пикселей. Прямая визуализация фотонной карты (500k излучено, 3M фотонов сохранено). Сверху: kd-tree (трассировка, балансировка, предвычисление облучения ¼ фотонов – 182.7 с, визуализация 5.2 с), снизу: octree texture (трассировка – 59.5 с, визуализация – 1.54 с)

5. ЗАКЛЮЧЕНИЕ

Данная работа описывает алгоритм, который позволяет сохранять фотоны сразу в октантную текстуру, но акцент в использовании октантных текстур сделан не на кэшировании, а на скорости финального сбора.

Была достигнута высокая производительность трассировки луча (порядка 1M лучей/с), поэтому для быстрой финальной сборки необходим также быстрый доступ к значению облучения. Использование октантных текстур дало необходимую производительность. Время запроса из

октантной текстуры на порядок меньше, чем из kd-tree фотонной карты, что увеличило скорость финальной сборки в 3 раза.

6. БИБЛИОГРАФИЯ

[1] James Arvo. Backward Ray Tracing. In Course Notes of the 1986 Conference on Computer Graphics and Interactive Techniques, no. 12, (Dallas, Texas, Aug. 18-22). ACM.

[2] D. Benson, J. Davis. Octree textures. In ACM Transactions on Graphics, Proc. SIGGRAPH 02 (2002), pp. 785.790.

[3] Per H. Christensen. Faster Photon Map Global Illumination. Journal of Graphics Tools, volume 4, number 3, pages 1-10. ACM, April 2000

[4] Per H. Christensen, Dana Batali. An Irradiance Atlas for Global Illumination in Complex Production Scenes. Eurographics Symposium on Rendering (2004)

[5] D. Debry, J. Gibbs, D. Petty, N. Robins. Painting and rendering textures on unparameterized models. In ACM Transactions on Graphics, Proc. SIGGRAPH 02 (2002), pp. 763.768.

[6] Vlastimil Havran. Heuristic Ray Shooting Algorithms, Ph.D. Thesis. November 2000

[7] James T. Hurley, Alexander Kapustin, Alexander Reshetov, and Alexei Soupikov. Fast Ray Tracing for Modern General Purpose CPU. In Proceedings of Graphicon, 2002. Available from http://www.graphicon.ru/-2002/papers.html.

[8] Henrik Wann Jensen. Global Illumination using Photon Maps. Department of Graphical Communication. The Technical University of Denmark Rendering Techniques '96 (Proceedingsof the Seventh Eurographics Workshop on Rendering), pages 21{30, 1996}

[9] Henrik Wann Jensen, Per H. Christensen, Toshiaki Kato, Frank Suykens. A Practical Guide to Global Illumination using Photon Mapping. Siggraph 2002 Course 43.

[10] B. Larsen, Per H. Christensen. Optimizing photon mapping using multiple photon maps for irradiance estimates. In Proc. 11th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG) (2003), University of West Bohemia.

[11] Fabien Lavignotte, Mathias Paulin. A new approach of density estimation for global illumination. In Proceedings of WSCG 2002 (2002), pp. 263.270.

[12] Tomas Möller, Ben Trumbore. Fast, Minimum Storage Ray_Triangle Intersection. Prosolvia Clarus AB Chalmers. University of Technology. Program of Computer Graphics. Cornell University.

[13] Peter Shirley, Bretton Wade, Philip M. Hubbard, Bruce Walter and Donald P.Greenberg. Global llumination via density estimation. In Proceeding of the Sixth Eurographics Workshop on Rendering, June 1995

[14] B.W. Silverman. Density Estimation for Statistics and Data Analysis. Chapman and Hall, London and New York, 1986.

[15] Frank Suykens, Yves D. Willems. Density Control for Photon Maps. In Rendering Techniques '00, Proc.11th Eurographics Workshop on Rendering (2000), Springer-Verlag, pp. 11 - 22.

[16] Eric Tabellion, Arnauld Lamorlette. An Approximate Global Illumination System for Computer Generated Films. Siggraph 2004 Proceedings.

[17] Ingo Wald. Realtime Ray Tracing and Interactive Global Illumination. PhD thesis, Computer Graphics Group, Saarland University, 2004. Available at http://www.mpi-sb.mpg.de/_wald/PhD/.

[18] Ingo Wald, Johannes Günther, Philipp Slusallek. Balancing Considered Harmful – Faster Photon Mapping using the Voxel Volume Heuristic. EUROGRAPHICS 2004. Volume 23 (2004), Number 3.

Об авторе

Востряков Константин Анатольевич, студент 5 курса факультета прикладной математики и кибернетики Томского Государственного Университета.

Адрес: Томск, 634050, пр. Ленина, 36, ТГУ, 2-й учебный корпус, факультет ПМК, кафедра программирования.

E-mail: vconst@inbox.ru.

Global Illumination via octree texture

Abstract

The paper is devoted to final gathering acceleration in photon map method. The final gathering consist of ray tracing and irradiance query from photon map. Kd-tree for geometry and fast ray-triangle intersection algorithm allow to achieve ray tracing performance on the order of 1M ray/s. Therefore a irradiance query from photon map must be fast too. Search in photon map kd-tree slower in several times than ray tracing, therefore author suggest to use a octree texture. It provides to speed ups irradiance query on the order of 10× and accelerates final gathering in 3 times. Besides author suggest method, which allows to store a photons to octree texture without using photon map kd-tree.

Keywords: *global illumination, photon map, octree texture, final gathering, ray tracing.*

About the author

Konstantin Vostryakov is a 5-year student at Tomsk State University, Department of Applied Mathematics and Cybernetics. His contact email is vconst@inbox.ru.