Использование Иерархического s-буфера Для Рендеринга Сцен Большой Сложности

А.В. Боресков
Факультете ВМиК МГУ
Москва, Россия

Abstract

В данной работе рассматривается метод, пригодный для рендеринга сцен большой сложности (количество граней от миллионов и выше). Метод основан на использование структуры восьмеричного дераева для ускорения работы метода s-буфера и быстром отсечении целых фрагментов сцен, основываясь на s-буфере.

Ключевые слова: *s-буфер, восьмеричное дерево*

1. ВВЕДЕНИЕ

Метод s-буфера [1] получил в последнее время довольно широкое распространение. Его несомненными плюсами являются простота, высокая эффективность, возможность обработки произвольных сцен без какой бы то ни было предварительной обработки, возможность работы с полупрозрачными гранями (правда, ценой некоторого усложнения s-буфера), полное устранение overdraw (многократного вывода в один и тот же пиксел), отделение удаления невидимых поверхностей от текстурирования и вычисления освещенности (они производятся только для тех элементов сцены, видимость которых уже установлена). Для повышения быстродействия метод удачно использует когерентность пикселов картинной плоскости по горизонтали.

2. ОПТИМИЗАЦИЯ СРАВНЕНИЙ

Рассмотрим сначала некоторые пути дальнейшей оптимизации этого метода, опирающиеся на простые геометрические соображения [3].

Одним из достаточно трудоемких моментов в методе s-буфера является необходимость постоянного сравнения сегментов между собой для выяснения того, какой из них какой загораживает. Считая, что все грани в сцене - выпуклые многоугольники, которые могут пересекаться только по границе, можно сформулировать следующее утверждение.

Утверждение 1. Пусть даны две выпуклые грани P и Q, которые могут иметь пересечение только по граничным точкам, и известно, что пересечение их проекций на картинную плоскость не пусто. Тогда существует единственный способ упорядочивания этих граней такой, что при их выводе в указанном порядке будет получено корректное изображение (т.е. грани правильно закроют друг друга).

Доказательство.

Пусть это не так. Тогда существуют две точки на картинной плоскости A и B такие, что вдоль проектора, проходящего через точку A, первая грань оказывается ближе к картинной плоскости, чем вторая, а вдоль проектора, проходящего через точку B, окажется ближе вторая грань. Введем расстояние со знаком между точками на гранях вдоль отрезка, соединяющего точки A и B, считая, что расстояние положительно, если соответствующая точка первой грани лежит ближе соответствующей точки второй грани, и отрицательно в противном случае. Так как в концах отрезка расстояние, являющееся непрерывной функцией, принимает значения разных знаков, то внутри отрезка AB существует точка C, где расстояние между соответствующими точками равно нулю, т.е. две грани имеют пересечение во внутренней точке (в силу их выпуклости), что противоречит условию. Утверждение доказано.

Заметим, что для трех и более граней это утверждение в общем случае неверно (рис. 1).

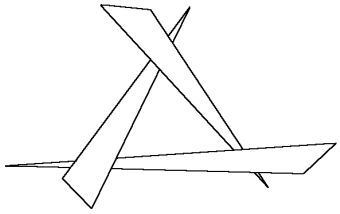


рис. 1

Рассмотрим пару произвольных граней P и Q. Из того, что их можно корректно упорядочить, следует, что для любой пары отрезков, получаемых при пересечении граней P и Q сканирующей плоскостью, результат сравнения этих отрезков одинаков и совпадает с результатом сравнения граней P и Q.

Воспользовавшись этим утверждением, можно кэшировать результаты сравнения отрезков текущей грани с отрезками в s-буфере, и если один раз сравнение произведено, то запомнить его в кэше и при обработке следующих строк вместо явного сравнения отрезков извлекать результаты из кэша.

Это позволит резко сократить общее количество проверок, так как вместо сравнения всех возможных пар отрезков достаточно лишь сравнить каждую грань со всеми гранями, чьи проекции на картинную плоскость пересекают ее проекцию. Тем самым за счет использования когерентности между отрезками, составляющими одну и ту же грань, быстродействие заметно повышается.

Попытаемся использовать когерентность между соседними отрезками в s-буфере.

Утверждение 2. Пусть отрезки A и B, лежащие в s-буфере, имеют общий конец. Тогда если отрезок C лежит впереди (позади) отрезка A, то он лежит впереди (позади) отрезка B при условии, что проекции отрезков A и B пересекают проекцию отрезка C (рис. 2).

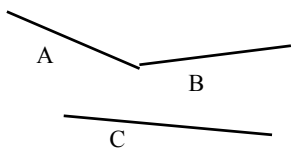


рис. 2.

Доказательство.

Пусть левый конец отрезка C лежит впереди отрезка A, а правый конец отрезка C - позади отрезка B. Пусть

проекцией отрезка C на картинную плоскость является отрезок αβ. Рассмотрим на отрезке αβ функцию, равную расстоянию со знаком между соответствующими точками отрезка C и ломаной $A\,B$.

В точках α и β эта функция принимает значения разных знаков. Так как эта функция непрерывна (отрезки A и B соединены друг с другом), то существует точка γ в которой отрезок C пересекает ломаную $A\,B$. Но так как грани могут пересекаться только по граничным точкам, то и отрезки этих граней могут пересекаться только в концевых точках. Получаем противоречие.

Аналогично формулируется и доказывается утверждение о когерентности смежных граней при их выводе в s-буфер.

Утверждение 3. Пусть грани P и Q являются смежными (имеют общее ребро) и их проекции на картинную плоскость пересекают проекцию грани R. Тогда, если грань P лежит впереди (позади) грани R, то и грань Q также лежит впереди (позади) грани R.

Последние два утверждения позволяют заметно сократить общее число попарных сравнений граней. Так перед сравнением с очередным отрезком мы сначала ищем результат сравнения в кэше, и если его там нет, то проверяем, сравнивали ли мы этот отрезок с предыдущим отрезком в s-буфере и соединены ли эти отрезки. Если да, то результат сравнения с предыдущим отрезком принимается за результат сравнения с текущим отрезком и заносится в кэш.

Список граней желательно упорядочить таким образом, чтобы смежные грани выводились подряд (для использования когерентности между ними).

Очевидно, что чем раньше заведомо видимый отрезок будет выведен в s-буфер, тем меньше потребуется затрат на отбрасывание закрываемых им невидимых отрезков. Это обстоятельство позволяет при построении анимаций использовать так называемую темпоральную (временную) когерентность.

Для ее использования, после завершения рендеринга текущего кадра составляется список всех видимых (чьи отрезки остались в s-буфере) граней. Тогда рендериг следующего кадра начинается с вывода этих граней, и чтобы избежать их повторного вывода, все эти грани помечаются как уже выведенные.

3. МЕТОД ИЕРАРХИЧЕСКОГО S-БУФЕРА

Классический метод s-буфера (наряду со всеми своими достоинствами) требует обработки всех лицевых граней, что делает его непригодным для работы с большими сценами.

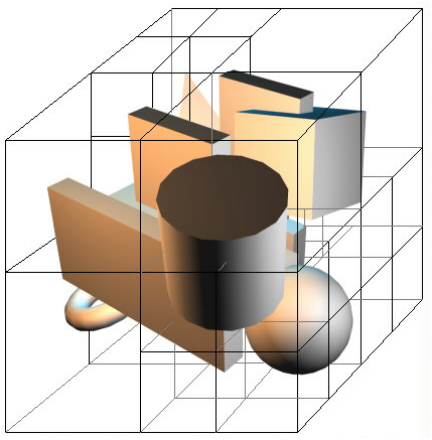


рис 3

Модифицируем этот метод, используя восьмеричное дерево [2].

Для этого опишем вокруг всей сцены прямоугольный параллелепипед со сторонами, параллельными координатным плоскостям. Если он содержит внутри себя достаточно большое число граней (больше, чем некоторое заданное число), разобьем его на восемь равных параллелепипедов. Далее для каждого из параллелепипедов, содержащего больше заданного числа граней, снова применим описанную процедуру разбиения и т.д. В результате мы получим восьмеричное дерево (рис. 3).

Тем самым каждой сцене, представленной как произвольный набор плоских граней, можно сопоставить восьмеричное дерево (octree).

Данная информация может быть с успехом использована для оптимизации процесса удаления невидимых поверхностей.

Для начала введем несколько необходимых определений.

Определение 1. Грань P будем называть невидимой по отношению к s-буферу, если любой пиксел ее растровой развертки находится не ближе к наблюдателю, чем соответствующий пиксел, находящийся в s-буфере.

Для проверки грани на невидимость она разбивается на набор горизонтальных отрезков и для каждого такого отрезка производится сравнение его с соответствующей строкой s-буфера. Если при этом окажется, что отрезок лежит дальше от наблюдателя, чем отрезки, занимающие соответствующие позиции в s-буфере (если бы этот отрезок был выведен в s-буфер, то при операции вставки его в буфер, он был бы полностью отброшен), то данный отрезок является невидимым. Если все отрезки растрового представления грани являются невидимыми по отношению к s-буферу, то и сама грань по отношению к этому s-буферу является невидимой.

Определение 2. Прямоугольный параллелепипед называется невидимым по отношению к s-буферу, если все его грани невидимы по отношению к этому s-буферу.

Достаточно ограничиться лишь проверкой лицевых граней параллелепипеда. Ясно, что если параллелепипед невидим, то и все содержащиеся в нем грани также невидимы. Для удобства проверки с каждым отрезком s-буфера свяжем ограничивающий его прямоугольник, т.е. два числа - максимальное и минимальное значения глубины, а с каждой строкой s-буфера свяжем максимальное значение глубины для отрезков этой строки.

Рассмотрим теперь, как можно использовать восьмеричное дерево и введенные выше определения для ускорения работы метода s-буфера.

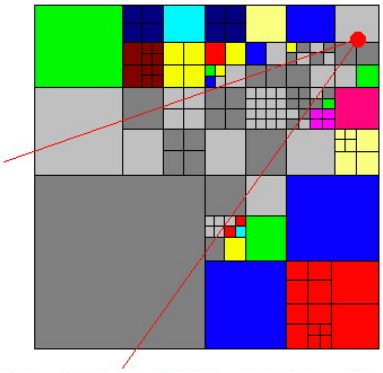
Ключевой концепцией здесь является проверка узла дерева (прямоугольного параллелепипеда) на видимость. В случае, если он невидим, все соответствующее поддерево можно сразу отбросить, не рассматривая. В противном случае проверяем восемь его непосредственных потомков на видимость и т.д.

Опишем теперь процедуру вывода всей сцены. Так как вокруг сцены описано восьмеричное дерево, то процедура получается рекурсивной.

Работа начинается с корневого параллелепипеда.

Очередной параллелепипед сначала проверяется на попадание в область видимости (четырехгранный угол с вершиной в точке положения наблюдателя (рис. 4)). Если этот параллелепипед в область видимости не попадает, то его

сразу же можно отбросить. В противном случае параллелепипед проверяется на видимость относительно текущего s-буфера. Если параллелепипед невидим, то его можно сразу же отбросить, так как ни одна грань,



содержащаяся в нем, заведомо видна не будет.

рис. 4

Если параллелепипед не является невидимым, но представляет из себя лист дерева, то в s-буфер выводятся все содержащиеся в нем лицевые грани. В противном случае рекурсивно обрабатываются все его восемь подпараллелепипедов в порядке их удаления от наблюдателя (front-to-back), т.е. каждый из них сначала проверяется на попадание в область видимости наблюдателя, а затем - на видимость относительно s-буфера, отбрасываются заведомо невидимые, после чего для оставшихся осуществляется та же процедура (либо непосредственный вывод всех содержащихся граней в s-буфер, либо последовательная обработка всех восьми подпараллелепипедов). Так как при построении дерева разбиение граней не производится, то возможна ситуация, когда одна и та же грань попадет сразу в несколько листьев восьмеричного дерева и, следовательно, будет выведена несколько раз. Чтобы этого не происходило, необходим механизм отслеживания уже выведенных граней. Тогда при выводе грани проверяется, не была ли грань уже выведена и в случае, если нет, она выводится и помечается как выведенная.

Каждый узел дерева можно представить как потомок следующего класса

```
class     OctTreeNode
{
   Vector3D min;
   Vector3D max;

public:
   OctTreeNode () : Storable () {}
   virtual ~OctTreeNode () {}

   int liesInViewingFrustrum () const;
   int isVisible (const Sbuffer& sBuffer)
const;

   virtual void render (const Camera&) const;
};
```

Все узлы дерева делятся на листья и внутренние узлы. Они представлены следующими классами

```
class     InternalNode : public OctTreeNode
{
   OctTreeNode * child [8];// child nodes

public:
   InternalNode () : OctTreeNode () {}
   ~InternalNode ();

   virtual void render(const Camera&) const;
};

class     LeafNode : public OctTreeNode
{
   int     * surfaceList;
   int       numSurfaces;
public:
   LeafNode () : OctTreeNode () {}
   ~LeafNode ()
   {
          delete surfaceList;
   }
   virtual void render(const Camera&) const;
};
```

Рассмотрим, как выглядят методы для каждого из этих классов

```
void      InternalNode :: render ( const
Camera& camera, SBuffer& sBuffer ) const
{
   if ( !liesInViewingFrustrum ( camera ) )
          return;

   if ( !isVisible ( sBuffer ) )
          return;

   int
   firstXIndex=(loc.x<0.5*(min.x+max.x)?0:1);
   int
   firstYIndex=(loc.x<0.5*(min.x+max.x)?0:2);
   int
   firstZIndex=(loc.x<0.5*(min.x+max.x)?0:4);
   int     index   = firstXIndex | firstYIndex
| firstZIndex;

          // now render child nodes in
          // front-to-back order
   child[index]      ->render(camera,
                              Sbuffer);
   child[index^1]    ->render(camera,
                              Sbuffer);
   child[index^2]    ->render(camera,
                              Sbuffer);
   child[index^4]    ->render(camera,
                              Sbuffer);
   child[index^1^2]  ->render(camera,
                              Sbuffer);
   child[index^1^4]  ->render(camera,
                              Sbuffer);
   child[index^2^4]  ->render(camera,
                              Sbuffer);
   child[index^1^2^4]->render(camera,
                              Sbuffer);
}

void      LeafNode :: render (const Camera&
camera, SBuffer& sBuffer) const
{
   for ( int i = 0; i < numSurfaces; i++ )
          if(!renderedSurfaces.find(
                 &surfaceList [i] ) )
          {
                 sBuffer.addSurface   (
camera, surfaces [surfaceList [i]] );
                 renderedSurfaces.add (
&surfaceList [i] );
          }
}
```

Описанный выше алгоритм удачно использует все возможные виды когерентности - в пространстве сцены

(благодаря использованию восьмеричного дерева), в картинной плоскости (как по горизонтали, так и по вертикали) и темпоральную когерентность.

Несложно убедиться в локальности данного метода - незначительные изменения сцены затронут всего лишь несколько узлов дерева, не изменяя его целиком (в отличие от BSP-деревьев). Тем самым метод иерархического s-буфера позволяет эффективно работать со сценами с изменяющейся геометрией.

Временные затраты данного метода составляют $o(n)$ в силу быстрого отбрасывания заведомо невидимых граней посредством s-буфера.

Непосредственно метод s-буфера работает с любыми полигональными сценами, таким образом и построенный метод позволяет обрабатывать сцены с любой структурой, необходимо лишь, чтобы они были представлены в виде набора граней.

Построение сцены крайне просто - берется произвольный набор граней и по нему строится соответствующее восьмеричное дерево. Сам процесс построения дерева не требует вмешательства человека.

Еще одним преимуществом метода иерархического s-буфера являются весьма невысокие требования на количество оперативной памяти - восьмеричное дерево можно легко разбить на ряд фрагментов (поддеревьев), которые будут динамически загружаться в память по мере необходимости (здесь также возможно использование proxy) и удалять из памяти, когда они больше не нужны.

При работе со сложными объектами (требующими использования уровня детализации или состоящими из искривленных поверхностей, и поэтому требующих полигонализации) можно с успехом использовать ограничивающие тела. При этом сначала ограничивающее тело объекта проверяется на видимость в s-буфере и только в том случае, когда хотя бы частичная видимость установлена, производится детализация (полигонализация) объекта. При этом для сложных объектов можно использовать иерархии ограничивающих тел (BVH).

Метод иерархического s-буфера может с успехом применятся и при использовании аппаратных ускорителей трехмерной графики. При этом s-буфер используется для отсечения невидимых граней (узлов восьмеричного дерева). Каждая потенциально-видимая грань выводится при этом как в s-буфер, так и в аппаратный z-буфер.

При этом для ускорения рендеринга можно использовать s-буфер с более низким разрешением, чем разрешение экрана.

Прозрачные грани требуют в этом случае специальной обработки –они не выводятся в s-буфер, а их вывод в z-буфер откладывается до того момента, когда будут выведены все непрозрачные грани. При этом прозрачные грани запоминаются, и перед выводом сортируются и выводятся уже в отсортированном порядке.

Ясно, что не все грани имеют одинаковое значение с точки зрения загораживания. Поэтому имеет смысл сразу пометить ряд граней как незагораживающие, эти грани в s-буфер не выводтся. К незагораживающим граням относятся мелкие грани, детали рельефа или объектов, вывод которых в s-буфер является нерациональным с точки зрения отношения затраченных ресурсов на загораживающую площадь. Любая достаточно мелкая грань автоматически помечается как незагораживающая в данном кадре в дополнении к тем граням, которые были помечены как незагораживающие при создании сцены.

Еще одним плюсом метода иерархического s-буфера является возможность использования построенного восьмеричного дерева для других целей, таких как проверки на столкновение (collision detection), определение взаимной видимости, быстрое нахождение объектов, удовлетворяющим каким-либо пространственным условиям. При этом схема использования восьмеричного дерева остается такой же как и при рендеринге – сперва проверяется корень дерева, далее его непосредственные потомки и так далее.

4. СПИСОК ЛИТЕРАТУРЫ

[1] Е.В.Шикин, А.В.Боресков. Компьютерная графика. Полигональные модели.//Москва. ДИАЛОГ-МИФИ, 2000

[2] А.В. Боресков. Метод иерархического s-буфера.// Программирование. 1998 N4

[3] А.В. Боресков. О некоторых геометрических свойствах s-буфера.// Программирование. 1999 N3

Об авторе

Боресков Алексей Викторович младший научный сотрудник факультета ВмиК МГУ.

E-mail: **alexboreskoff@mtu-net.ru**