Использование порталов для создания агрегатных сцен

Боресков А.В.
МГУ ВМиК
Москва, Россия

Аннотация

В статье рассматривается создание агрегатных сцен при помощи порталов. При этом каждая такая агрегатная сцена является объектом со своей внутренней структурой и своим методом рендеринга.
Ключевые слова: агрегатная сцена, портал

1.Введение

В методе трассировки лучей существует понятие агрегатного объекта - объекта, состоящего из множества других объектов, и знающего как находить пересечение заданного луча с объектами внутри себя, т.е. имеющего свой виртуальный метод для нахождения пересечения луча с содержащимися внутри него объектами. Обычно агрегатный объект инкапсулирует, т.е. содержит внутри себя, скрывая внутренние детали и структуру, один из стандартных методов оптимизации - иерархии ограничивающих тел (BVH), восьмеричные или BSP-деревья и т.п. При этом он может содержать внутри себя другие агрегатные объекты, так что возникает иерархия объектов. Преимущество подхода, при котором вся сцена представляется как один большой агрегатный объект, состоит в том, что появляется возможность группировать простые объекты в агрегатные структуры, выбирая для каждой свой способ оптимизации и свою внутреннюю структуру [1].

В данной работе делается попытка использовать аналогичный подход для рендеринга сложных сцен методами, отличными от метода трассировки лучей.

В настоящее время для рендеринга трехмерных сцен обычно используются методы z-буфера, метод BSP-деревьев и метод порталов.

Наиболее простым из перечисленных является метод z-буфера (буфера глубины). Он заключается в создании так называемого буфера глубины, где для каждого выведенного пиксела хранится расстояние от его прообраза в пространстве до картинной плоскости. При этом вывод очередного пиксела состоит в сравнении его глубины со значением глубины, уже записанном в буфер. Если глубина выводимого пиксела меньше глубины, хранящейся в z-буфере, то пиксел выводится, а его глубина запоминается в z-буфере.

Основным недостатком этого метода является его сильная избыточность при выводе граней - обязательно выводятся все лицевые грани и только в конце определяется, какие из них видны, а какие - нет. Для достаточно сложных сцен с текстурированием и освещением граней такой подход становится неприемлемым - слишком много усилий по текстурированию и расчету освещенности тратится впустую. Известный выигрыш способно дать использование множеств потенциально-видимых граней (PVS), когда выводятся не все грани, а лишь содержащиеся в списке граней, которые можно увидеть, находясь в данном месте (комнате, многограннике). При этом все пространство разбивается на многогранники и для каждого из них строится список видимых из него граней. Это значительно ускоряет рендеринг, но построение множеств потенциально-видимых граней довольно сложно и трудоемко и к тому же не годится для сцен с изменяемой геометрией.

Для не очень больших сцен можно с большой эффективностью использовать метод BSP-деревьев для упорядочивания граней и вывода их в порядке удаления от наблюдателя. Если этот метод совместить с каким-либо методом для отслеживания уже выведенных пикселов, например, с методом s-буфера, то получается довольно эффективный метод, способный быстро строить изображение сцены из произвольных точек обзора. Однако количество обрабатываемых граней оказывается довольно большим, а главное - при построении дерева происходит разбиение граней, тем большее, чем больше общее количество граней в сцене. Поэтому пригодность этого метода для обработки действительно сложных сцен вызывает большие сомнения. Кроме того, этот метод не годится для сцен с изменяемой геометрией.

В этом контексте более привлекательным выглядит метод порталов [2]. Он заключается в разбиении всей сцены на множество выпуклых многогранников (комнат), соединенных между собой порталами (своего рода дырками), через которые из одного многогранника можно увидеть содержимое другого (рис. 1). Рендеринг начинается с многогранника, в котором находится наблюдатель, затем для каждого портала этого многогранника производится рендеринг соседнего многогранника, при этом сам портал, соединяющий эти многранники, выступает в качестве области отсечения - все, что не попадает в портал (не видно сквозь него) автоматически отсекается. Далее осуществляется рендеринг многогранников, соединенных порталами с многогранниками, соседними для начального, и т.д. При рендеринге очередного многогранника в качестве области отсечения выступает пересечение всех порталов, ведущих от исходного многогранника к текущему. Рендеринг самого многогранника заключается просто в выводе всех его лицевых граней (напомним, что многогранник выпуклый).

Работу метода порталов можно представить следующим фрагментом псевдокода.

```
void renderScene ( Polygon clippingPoly, Cell
                   activeCell )
{
    for each facet of activeCell
        if ( facet intersects viewingFrustrum
)
             drawFacet ( clipFacet ( facet,
                  clippingPoly ) );

    for each portal of activeCell
         if ( portal intersects
               viewingFrustrum )
         {
              Polygon clippedPortal =
clipPortal ( portal, clippingPoly );

              renderScene ( clippedPortal,
adjacentCell ( activeCell, portal ) );
         }
}
```

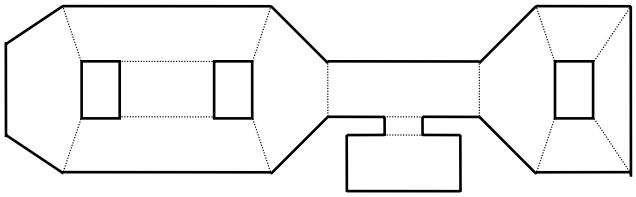

рис. 1

Однако само разбиение сцены на множество выпуклых многогранников довольно сложно, для больших сцен возникает огромное количество порталов, что заметно снижает общее быстродействие алгоритма.

2. Понятие агрегатной сцены.

Попробуем использовать метод порталов для создания агрегатных сцен - фрагментов сцен, умеющих себя визуализировать, т.е. содержащих внутри себя методы (программы) для своей визуализации. Ясно, что метод визуализации агрегатной сцены зависит от ее структуры и для разных фрагментов сцены может быть различным.

Разобьем всю сцену на несколько фрагментов (не обязательно выпуклых) при помощи порталов (рис 2.) и с каждым фрагментов свяжем наиболее подходящий для него метод рендеринга. Рендеринг всей сцены производится примерно так же, как и для чистого метода порталов - начиная с фрагмента, содержащего наблюдателя, далее, используя порталы, как области отсечения, проводится рендеринг соседних фрагментов и т.д.

Один из самых простых и в то же время достаточно эффективных подходов заключается в применении во всех получившихся фрагментах метода BSP-деревьев. Сильного разбиения граней при этом не возникает, а вместо одного большого дерева получается набор сравнительно небольших деревьев, причем для рендеринга всей сцены не обязательно обходить их все. Если же один или несколько фрагментов носят динамический характер, то для их рендеринга вполне можно воспользоваться либо методом z-буфера , либо методом сортировки граней.

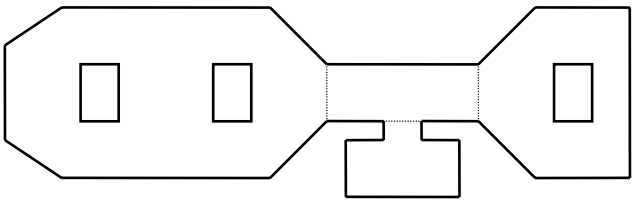

рис 2.

В случае использования метода z-буфера для обеспечения отсечения по порталу достаточно соответствующим образом проинициализировать z-буфер перед выводом в него граней текущего фрагмента.

Для рендеринга фрагментов открытого пространства можно с успехом использовать метод сортировки граней или метод плавающего горизонта, которые в данном случае работают крайне эффективно и обеспечивают быстрое отсечение невидимых граней.

Довольно интересным представляется использование метода s-буфера для рендеринга отдельных фрагментов. Этот метод в последнее время получил достаточно широкое распространение, однако он используется в большинстве случаев только для отслеживания уже заполненных фрагментов картинной плоскости. Здесь же можно применить его для удаления невидимых поверхностей, полностью используя его простоту и быстродействие.

При кэшировании результатов сравнения очередной грани с гранями, уже помещенными в буфер, необходимо сравнивать выводимую грань лишь с теми гранями из данного фрагмента, чьи проекции на картинную плоскость перекрываются, что не требует особенно больших затрат в силу локальности фрагмента.

Использование когерентности между отрезками s-буфера и когерентности между соседними лицевыми гранями позволяет еще больше снизить общее количество проверок.

В результате получается крайне простой и эффективный алгоритм, позволяющий полностью избежать вывода невидимых пикселов (т.е. расчета их освещенности и текстурирования), расщепления граней и ограничиться при этом сравнительно небольшим количеством проверок.

Сцену, представленную в виде отдельных фрагментов, каждый из которых для удаления невидимых поверхностей использует метод s-буфера, будет отличать простота построения, высокая гибкость (возможность на ходу изменять геометрию отдельных фрагментов) и высокаяч скорость рендеринга.

В результате подобного подхода всю сцену можно представить как единый агрегатный объект, состоящий

из отдельных фрагментов, соединенных при помощи порталов. Каждый из этих фрагментов имеет свою внутреннюю организацию, наиболее естественно ему соответствующую, и свой метод рендеринга. В частности, отдельные фрагменты также могут являться агрегатными сценами, разбитыми порталами на отдельные подфрагменты. В результате вся сцена описывается при помощи некой иерархической структуры (дерева), задающей разбиение сцены на первичные фрагменты. Для каждого агрегатного объекта этого дерева задается ненаправленный граф, несущий в себе информацию о том, какой фрагмент с каким соединен и при помощи какого портала.

Применение подобного подхода позволяет сравнительно легко строить сложные сцены, состоящие из разнородных фрагментов. Выбирая для каждого из фрагментов наиболее подходящую внутреннюю организацию и метод рендеринга, мы совмещаем тем самым легкость построения сцены с высокой скоростью и гибкостью рендеринга.

Основным, что должен уметь делать каждый фрагмент, является построение своего изображения в заданном многоугольнике картинной плоскости. Таким образом представленный фрагмент является потомком следующего чисто виртуального класса:

```
class SubScene
{
public:
   SubScene () {}
   virtual ~SubScene () {}

   virtual void render ( Polygon& ) = 0;
}
```

Сама же агрегатная сцена представляет собой коллекцию фрагментов и порталов, а также связывающий их между собой граф.

```
class AgregateScene : public SubScene
{
public:
   Collection <SubScene> phragmentCol;
   Collection <Polygon>  portalCol;
   int * connectionGraf;

   AgregateScene ();
   ~AgregateScene ()
   {
          delete phragmentCol;
          delete portalCol;
          delete connectionGraf;
   }

   int     isConnected ( int i, int j )
   {
          return               connectionGraf
[i*phragmentCol->itemCount + j];
   }

   virtual void render ( Polygon& );
}
```

3. Заключение

Подобный объектно-ориентированный подход, когда каждый фрагмент является объектом, инкапсулирующим в себе наиболее подходящий ему метод рендеринга, обеспечивает высокую степень гибкости применения, возможность активного использования сцен с изменяемой геометрией (поскольку появляется возможность изолировать фрагменты с изменяемой геометрией при помощи порталов и применить наиболее эффективные методы для их рендеринга, не затрагивая при этом обработку всех остальных фрагментов) и разнородной структурой.

При этом достигается легкость и удобство построения композитных сцен, когда сцена строится из готовых объектов, каждый раз выбирая наиболее подходящий объект, и , снижая тем самым, общую сложность сцены.

Одновременно при этом достигается и высокая скорость рендеринга вследствие локализации потенциально-видимых фрагментов и снижения затрат на рендеринг отдельных фрагментов.

Литература

[1]. Andrew Glassner. An introduction to ray tracing. - Academic Press, 1989.

[2]. Seth Teller. Visibilty computations in Densely Occluded Polyhedral Environments. Dissertation., 1992.

Автор

Боресков Алексей, младший научный сотрудник факультета Вычислительной математики и кибернетики Московского Государственного Университета им. М.В. Ломоносова.
Адрес: Москва, 103064, Большой Казенный пер., дом 7, кв. 14
Телефон: (095)917-3830
E-mail: alex@garant.ru

Summary

Boreskoff Alex.
Creating aggregate scenes using portals.
In the article the author considers splitting of entire scene into subscenes using portals. Each subscene is considered as an object with it's own internal structure and it's own method to render it. So a complex scene can be rendered as a sequence of subscenes, each subscene representing it's own method of rendering.