# Обзор некоторых алгоритмов удаления перекрываемых поверхностей

Автор: Надежда Гнездилова nadya@pisem.net



## Введение

Современное программное обеспечение в таких областях как CAD, архитектурное моделирование и т.п. требует интерактивной визуализации динамических данных очень больших объемов, на порядки превосходящих возможности аппаратуры. Высокой скорости показа таких данных можно добиться с помощью поиска и удаление частей сцены, невидимых наблюдателю (в частности, перекрываемых поверхностей), таким образом, снижая нагрузку на графическую подсистему.

Задача удаления невидимых поверхностей является классической задачей компьютерной графики, и еще в 70-х годах было предложено значительное количество алгоритмов для ее решения. Удаление невидимых поверхностей основывается на том факте, что если пользователь не видит некоторый объект, то нет необходимости этот объект визуализировать, то есть визуализировать нужно только полностью или частично видимые объекты. Алгоритмы удаления невидимых поверхностей определяют невидимые для пользователя части сцены и не визуализируют их (удаляет из множества визуализируемых частей). На сегодняшний день можно считать базовую задачу решенной, а самым распространенным алгоритмом ее решения - алгоритм z-буфера, реализованный аппаратно. Краткое описание работы этого алгоритма дано в приложении.

Время работы z-буфера линейно относительно количества полигонов в сцене. Поэтому, использование одного только z-буфера не подходит для интерактивной визуализации современных сцен даже на самых мощных компьютерах. Таким образом, встает вопрос о разработке алгоритмов, которые бы быстро находили и отсекали пусть не всю, но значительную часть невидимой геометрии сцены, а уже все остальное можно было бы визуализировать с помощью z-буфера. У таких алгоритмов время работы, как правило, логарифмически зависит от количества полигонов в сцене.

### *Определения*

В зависимости от характера удаляемой геометрии алгоритмы делятся на (1) алгоритмы, удаляющие поверхности, чьи нормали направлены от наблюдателя (задние грани объектов) (back-face culling), (2) алгоритмы, удаляющие полигоны, не попадающие в пирамиду видимости (view-frustum culling), и (3) алгоритмы, удаляющие *перекрываемые поверхности* (occlusion culling). Даже те поверхности, которые находятся в пределах пирамиды видимости и повернуты своими нормалями к наблюдателю, все равно могут быть не видны, поскольку могут перекрываться другими непрозрачными объектами, которые находятся ближе к точке наблюдения (примером может служить солнечное затмение или игра в прятки). Такие поверхности и называются перекрываемыми поверхностями. Чтобы их удалить, алгоритмы тем или иным образом выбирают поверхности, относительно которых затем производится проверка, являются ли другие поверхности видимыми. Такие поверхности называются *перекрывающими*

*поверхностями*.

Важным понятием для алгоритмов удаления перекрываемых поверхностей является *консервативность*. Консервативный алгоритм всегда находит надмножество видимой геометрии - результатом его работы будет набор полигонов, включающий в себя все видимую геометрию и, по возможности, небольшое количество невидимой геометрии.

Целью алгоритмов удаления перекрываемых поверхностей является сведение стоимости визуализации сцены к сложности видимой части сцены. Идеально, алгоритм должен быть *чувствительным к выводу* - то есть время его работы должно быть пропорционально объему видимой геометрии.

## Классификация

Можно классифицировать алгоритмы удаления перекрываемых поверхностей по следующим критериям:

- Определение видимости для текущей позиции наблюдателя или же сразу для некоторой части сцены (from-point versus from-region). Достоинством второго класса алгоритмов является то, что результаты их работы действительны в течение нескольких кадров. Однако такие алгоритмы нуждаются в возможно достаточно длительном этапе предобработки и не так хорошо как алгоритмы первого класса, справляются с динамической геометрией в сцене.
- Работа в объектном или экранном пространстве (object-space versus image-space). Алгоритмы, работающие в объектном пространстве, для решения поставленной задачи используют непосредственно геометрию в сцене. Алгоритмы же, работающие в экранном пространстве, используют дискретное представление, полученное в результате растеризации объектов.
- Тип сцен, для которых предназначается алгоритм. Обычно проводят различие между архитектурными сценами и сценами общего класса (architectural versus generic). Данное разграничение обусловлено тем, что архитектурные сцены имеют изначальное разбиение на области, что естественно использовать в алгоритме.

Некоторые дополнительные характеристики:

- Обычно при работе алгоритмы создают и помещают сцену в некоторую структуру. Это может быть восьмеричное дерево (алгоритм построения см. в приложении), дерево объектов, BSP дерево (алгоритм построения см. в приложении), регулярная сетка, либо нечто другое. При использовании всех перечисленных выше типов структур, кроме дерева объектов, сцена воспринимается как "суп из полигонов", то есть некий не структурированный набор примитивов.
- Очень важным при работе алгоритма является способ выбора перекрывающих поверхностей. Алгоритм может либо заранее на этапе предобработки вычислить набор таких плоскостей или же находить их динамически во время работы со сценой.

## Обзор

### *Cells-and-Portals*

В одной из первых работ на тему удаления перекрываемых поверхностей [8] авторы предложили алгоритм для работы с архитектурными сценами. На этапе предобработки алгоритм создает описывающую структуру, используя естественное разбиение сцены на комнаты - BSP дерево, где в качестве секущих плоскостей выступают стены комнат.

Для каждой комнаты, также во время предобработки, находится потенциально видимая геометрия (PVS - potential visible set) - другие комнаты, которые можно увидеть через дверные проемы (иначе называемые порталами), находясь в этой комнате. Потенциально видимыми являются комнаты, через которые проходят лучи, выпущенные из обрабатываемой комнаты и соединяющие последовательности порталов (Рис. 1.). Во время интерактивной работы со сценой алгоритм определяет положение наблюдателя и вычисляет, какие из потенциально видимых комнат, найденных заранее, попадают в пирамиду видимости. Геометрия, ассоциированная с этими комнатами, посылается на обработку в графическую

подсистему для вывода ее на экран.

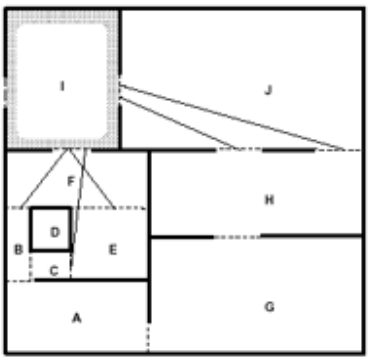


Рис. 1. Нахождение потенциально видимой геометрии для комнаты I.

Данный алгоритм, вообще говоря, предназначен для работы с двумерными сценами, в которых стены комнат параллельны осям координат и его сложность сильно зависит от конфигурации комнат. В худшем случае она равна сложности z-буфера (когда все комнаты видны).

Существует расширение этого алгоритма для трехмерного случая [9].

***Shadow Frusta***

Еще один алгоритм, предложенный в 1997 году [6], использует идею о том, что объект невидим наблюдателю, если он находится «в тени» некоторого другого объекта, то есть если принять положение наблюдателя за положение источника света и отбросить тень от перекрывающего объекта, то все объекты, попадающие в тень не освещены, то есть не видны наблюдателю. Рис. 2 иллюстрирует эту ситуацию.

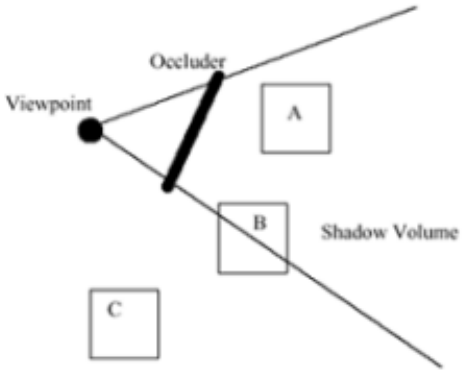


Рис. 2. Объект A полностью попадает в тень перекрывающей плоскости, B – частично, C – не попадает вовсе.

На этапе предобработки алгоритм разбивает сцену на области с помощью иерархической структуры данных (в том числе это может быть восьмеричное дерево). Далее, с каждой областью разбиения ассоциируется набор потенциально хороших перекрывающих плоскостей. (Насколько «хороша» та или иная поверхность определяется по разным критериям, в том числе алгоритм стремится найти поверхности, за которыми находиться много других объектов на большом от него расстоянии). При интерактивной работе со сценой алгоритм определяет, в какой области находится наблюдатель, какие из потенциально хороших перекрывающих поверхностей попадают в пирамиду видимости и для каждой такой поверхности производит следующие действия:

- Строит пирамиду ее тени. Вершиной пирамиды является точка положения наблюдателя, гранями

– плоскости, проходящие через вершину и периметр поверхности.

- Рекурсивно обходит иерархическую структуру, начиная с вершины, и для каждого ее узла определяет:
    - Попадает ли он в пирамиду видимости
    - Если да, то попадает ли он в пирамиду тени. Если попадает, то все поддерево не визуализируется. Если не попадает, то отправляет всю геометрию, ассоциированную с поддеревом в графическую подсистему для вывода на экран. Если попадает часть, то обработка происходит рекурсивно дальше для всех потомков этого узла.

Алгоритм имеет логарифмическую сложность.

***Hierarchical Z-buffer***

В 1993 году в качестве расширения традиционного алгоритмы Z-буфера был предложен алгоритм иерархического Z-буфера [5] (Hierarchical Z-Buffer – HZB), использующий восьмеричное дерево при работе в объектом пространстве и Z-пирамиду при работе в экранном пространстве. Алгоритм производит обход восьмеричного дерева, начиная с вершины и для каждого узла (1) определяет, попадает ли он в пирамиду видимости, если да, то (2) определяет, видны ли грани куба, соответствующего этому узлу, и если видны, то (3) производит аналогичную обработку для дочерних узлов, вплоть до того, пока не будут достигнуты листья дерева, тогда (4) ассоциированная с ними геометрия выводиться на экран.

Для того чтобы быстро определять видимость граней кубов, соответствующих узлами дерева, используется Z-пирамида. Z-пирамида – это иерархия карт глубины. Каждый следующий уровень иерархии имеет разрешение в два раза меньше предыдущего и строиться на его основе. В качестве самого нижнего уровня выступает Z-буфер. Из каждой четверки соседних значений с нижнего уровня выбирается самое большое значение, то есть соответствующее самой далекой точке, и помещается в элемент следующего уровня. На Рис. 2. это проиллюстрировано для Z-буфера размером 4x4.

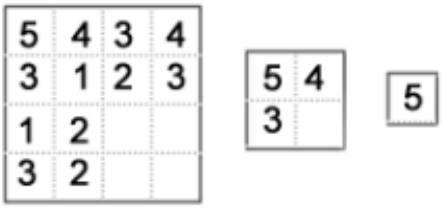


Рис. 3. Z-пирамида

Определение видимости грани куба осуществляется следующим образом:

- В пирамиде находиться такой уровень, на котором один пиксель соответствует области, полностью покрывающей проекцию грани куба.
- Если этот пиксель содержит величину меньшую самой близкой точки грани, то грань не видна (т.е. грань находиться дальше, чем самая дальняя точка области)
- Если самая ближняя точка грани меньше, чем величина, хранимая в карте (грань находиться ближе), то происходит переход к следующему более низкому уровню пирамиды, производиться вычисление самой близкой точки грани в квадранте и ее сравнение с элементом в карте на этом уровне иерархии.

Поиск ближайшей точки грани для каждого исследуемого квадранта требует значительного времени, поэтому в заявленной авторами реализации алгоритм останавливается на втором шаге, и сразу после первого сравнения, если выясняется, что грань видна относительно этого уровня, производит растеризацию грани для точного определения ее видимости.

Алгоритм имеет логарифмическую сложность и основывается на возможности быстро выяснить, будет ли точка видима, если ее визуализировать. Для достижения интерактивной работы со сценой ответ на этот

вопрос (z-readback) необходимо уметь получать с использованием аппаратных средств [12].

***Таблица сравнения***

| Метод | 2D/3D | Консервативность | Перекрывающие поверхности | Предобработка | Необходимость в специфической аппаратной поддержке |
|---|---|---|---|---|---|
| Cells-and-Portals | 2D, 3D | Да | Стены комнат | Вычисление PVS | Нет |
| Shadow frusta | 3D | Да | Большие, выпуклые полигоны | Выбор перекрывающ-их поверхностей | Нет |
| HZB | 3D | Да | Все | Нет | Требуется z-readback |

Таблица 1. Сравнение нескольких алгоритмов.

## Приложение

***Z-буфер***

В элементах массива, размерность которого соответствует разрешению экрана, содержатся значения координат z (глубины) визуализированных точек. Этот массив называется z-буфером. При растеризации новой точки, значение ее координаты z сравнивается с текущим значением в элементе массива с соответствующими координатами x, y. Если значение оказывается больше, т.е. точка находиться дальше, чем текущая точка в буфере, то ничего не происходит, если же оказывается, что значение координаты меньше, хранимого в массиве, т.е. точка находиться ближе, то значение элемента массива устанавливается равным новому значению глубины.

***Восьмеричное дерево***

Для работы со сценами большой сложности часто используется восьмеричное дерево (octree). Алгоритм его построения дан ниже:

1. Вся сцена помещается в куб с гранями, параллельными координатным плоскостям.
2. Этот куб делиться тремя плоскостями, параллельными координатным плоскостям, на восемь равных частей
3. Каждый из получившихся восьми кубов рекурсивно делиться еще на восемь кубов и т.д.
4. В каждой ветке деление прекращается, либо когда достигается заранее определенная глубина дерева, либо когда в очередном кубе количество примитивов становится меньше заранее определенного порога.
5. Примитивы сцены ассоциируются с кубами, соответствующими листьям получившегося дерева.
6. При обработке примитивов, которые разбиваются секущими плоскостями можно:
    1. Ассоциировать разбиваемый примитив с родительским кубом.
    2. Разбивать примитив и каждую его часть ассоциировать с соответствующим листом.
    3. Не разбивая примитив ассоциировать его со всеми листьями, в которые он попадает.

Недостатком втором подхода является увеличение количества примитивов в сцене. Обычно используют последний подход.

***BSP дерево***

BSP (binary space partitioning) дерево может быть определено в любом измерении. Это бинарное дерево, где каждый узел представляет собой некоторую гиперплоскость, левое поддерево соответствует отрицательному подпространству этой гиперплоскости, а правое – положительному. Например, в двумерном случае каждый узел представляет собой линию, а поддерево – области на плоскости. BSP

дерево является обобщением восьмеричного дерево, при котором секущие плоскости не должны проходит через центр разбиваемой области, и не должны быть параллельны координатным плоскостям (осям). Рис. 3. иллюстрирует пример построения BSP дерева. Буквами обозначены секущие плоскости, цифрами – области пространства.

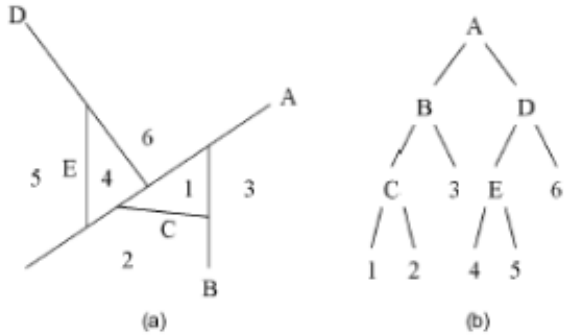


Рис. 3. BSP дерево.

## Литература


1. Daniel Cohen-Or, Yiorgos Chrysanthou, Claudio T. Silva, and Fredo Durand, “A Survey of Visibility for Walkthrough Applications,” to appear in IEEE Transactions on Visualization and Computer Graphics.
2. F. Durand. 3D Visibility: Analytical study and Applications. PhD thesis, Universite Joseph Fourier, Grenoble, France, July 1999.
3. Fredo Durand, George Drettakis, Jo? elle Thollot, and Claude Puech. Conservative visibility preprocessing using extended projections. Proceedings of SIGGRAPH 2000, pages 239–248, July 2000.
4. James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. Computer Graphics, Principles and Practice, Second Edition. Addison-Wesley, Reading, Massachusetts, 1990.
5. Ned Greene, M. Kass, and Gary Miller. Hierarchical z-buffer visibility. Proceedings of SIGGRAPH 93, pages 231–240, 1993.
6. T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow frustra. In Proc. 13th Annu. ACM Sympos. Comput. Geom., pages 1–10, 1997.
7. Oded Sudarsky and Craig Gotsman. Dynamic scene occlusion culling. IEEE Transactions on Visualization and Computer Graphics, 5(1):13–29, January - March 1999.
8. Seth J. Teller and Carlo H. Sequin. Visibility preprocessing for interactive walkthroughs. Computer Graphics (Proceedings of SIGGRAPH 91), 25(4):61–69, July 1991.
9. Seth J. Teller. Visibility Computations in Densely Occluded Environments. PhD thesis, University of California, Berkeley, 1992.
10. Hansong Zhang. Effective Occlusion Culling for the Interactive Display of Arbitrary Models. Ph.D.thesis, Department of Computer Science, UNC-Chapel Hill, 1998.
11. Hansong Zhang, Dinesh Manocha, Thomas Hudson, and Kenneth E. Hoff III. Visibility culling using hierarchical occlusion maps. In Turner Whitted, editor, SIGGRAPH 97 Conference Proceedings. Annual Conference Series, pages 77–88. ACM SIGGRAPH, Addison Wesley, August 1997.
12. Вадим Репин. Иерархический Z-буфер. Публикация на iXBT. Технология HyperZ, в ATI Radeon 256. http://library.graphicon.ru/paper/693. 2 июля 2000 года