Машинная графика
Computer Graphics

Лекция 12.

«Удаление невидимых линий и поверхностей II»

2006

План лекции

- Алгоритм Варнока
- Необходимость деления сцены
- Порталы
- Восьмеричные деревья
- BSP-деревья
- Прямая трассировка лучей

Алгоритм Варнока (Warnock)

- Элегантный гибрид пространства объектов и пространства изображений.
- В основе – стандартный подход: если ситуация слишком сложная – то делить задачу на подзадачи.
- Начиная с корневого окна:
  - Если пересечений полигонов в окне ноль или одно, то окно отрисовывается
  - Иначе окно делится на четверти как четверичное дерево (quadtree)
  - Вызов для каждого подокна рекурсии, до тех пор, пока в каждом не будет по одному или нулю пересечений, или пока не будет достигнута максимальная точность.
  - Максимальная точность – разрешение экрана (пиксель). Рисуется ближайший к центру пикселя полигон.

Алгоритм Варнока (Warnock)

Иными словами **алгоритм Варнока** работает следующим образом:

- делит изображение на 4 ячейки;
- для каждой ячейки рассматривает только случаи **пересечения** с полигонами;
- если видимость полигона не установлена, делим ячейку снова;
- продолжаем рекурсию вплоть до пиксельного уровня.

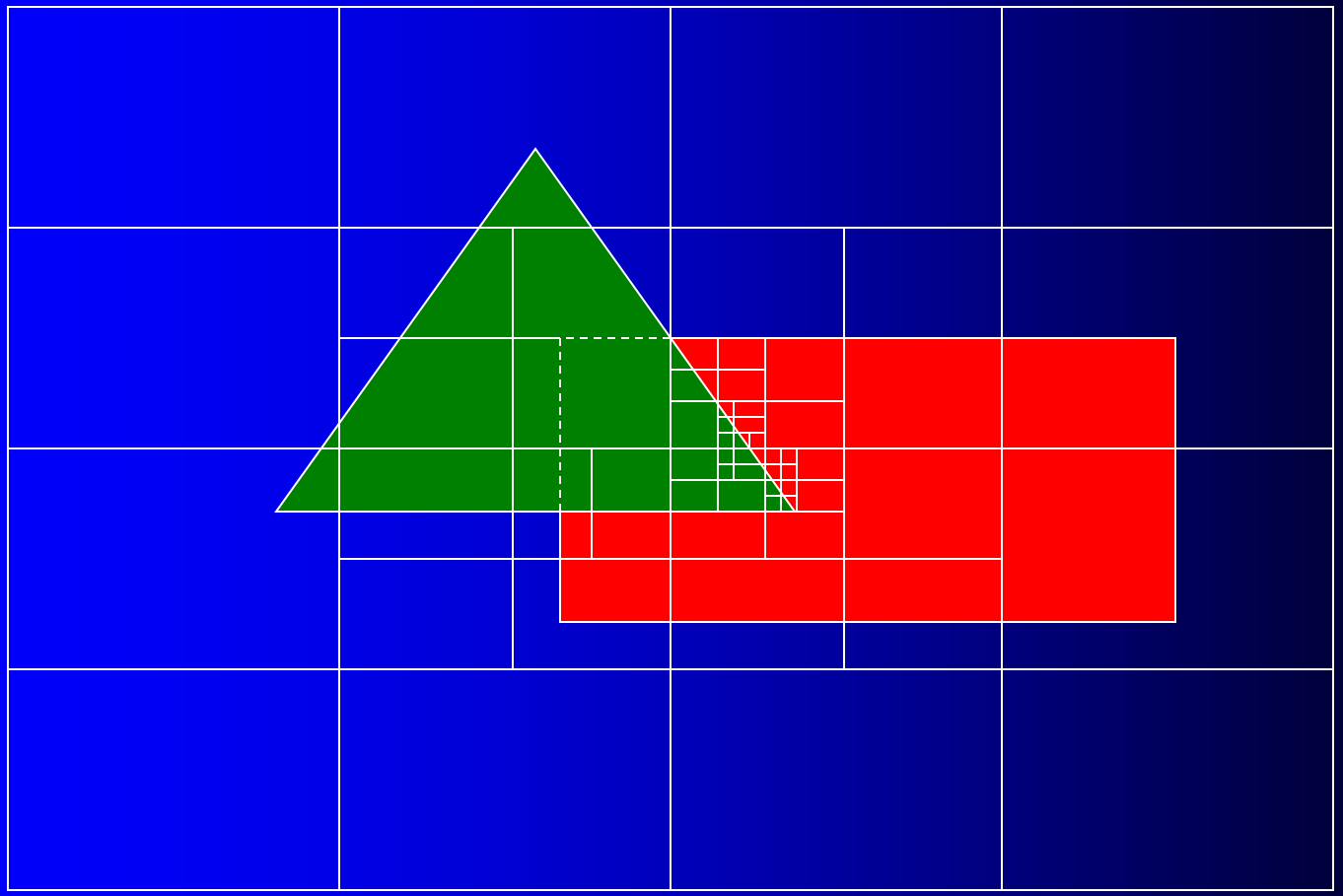
По отношению к ячейке, **алгоритм** классифицирует полигон как:

- внешний (охватывающий ячейку полностью)
- внутренний (полностью внутри ячейки)
- пересекающий (края полигона пересекают ячейку)
- расположенный за пределами ячейки.

Когда считать, что видимость полигона внутри ячейки окончательно определена?

- когда в ячейке нет ни одного полигона;
- внутри ячейки только один пересекающий или внутренний полигон;
- когда один внешний полигон находится ближе всех остальных.

Алгоритм Варнока (Warnock)



Пример

Алгоритм Варнока (Warnock)

- Вычислительная эффективность алгоритма:
  - Деление экрана на области (разрешение экрана, r = w*h) гибридное пространсто в object-space & image-space неплохо подходит для определённых примитивов, достаточная точность.
  - Требования по рабочему объёму памяти: O(n)
  - Объём доп. памяти для хранения (over & above model): O(n lg r)
  - Время на визуализацию с точностью до пикселя: O(n*r)
  - Перерисовка (как часто в среднем перерисовывется пиксель при растеризации): 1 раз.

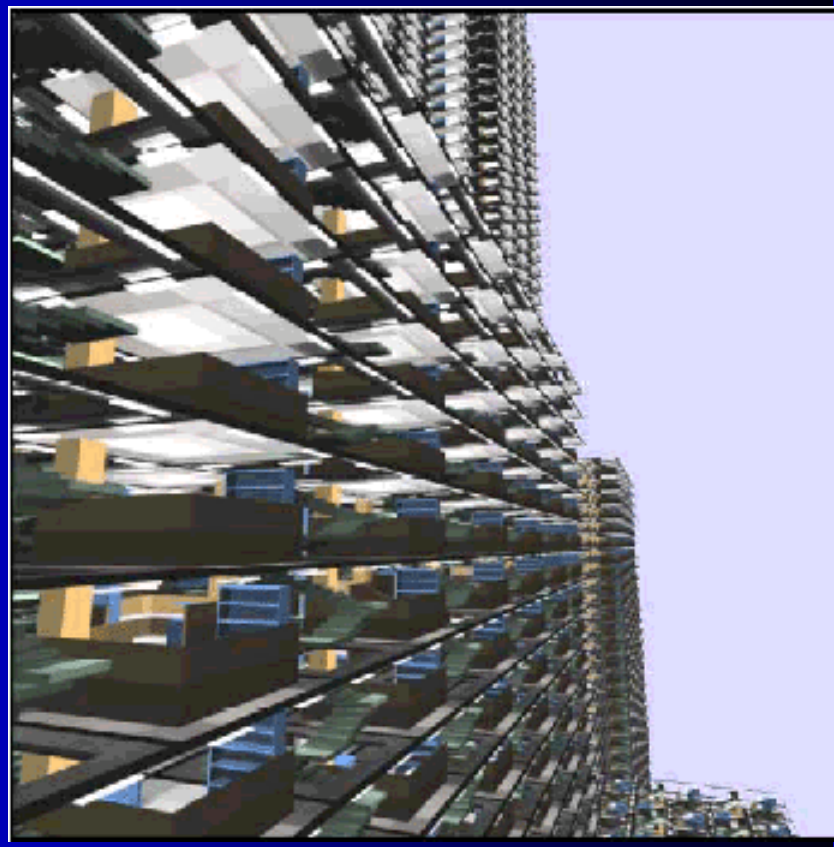
Перекрытия поверхностей на различных уровнях

Зачем делить пространство?

Кадр из анимации "Naked Empire". Модель этого 408 этажного здания состоит из 167 миллионов прямоугольников, 750000 из которых видимы в данном ракурсе. Это изображение размером 512x512 пикселей было просчитано алгоритмом иерархического тайлинга с включенной функцией фильтрации на базе стохастической сетки 4096x4096 сэмплов. Рендеризация этого кадра потребовала 5.15 минут на 75 Мгц процессоре.

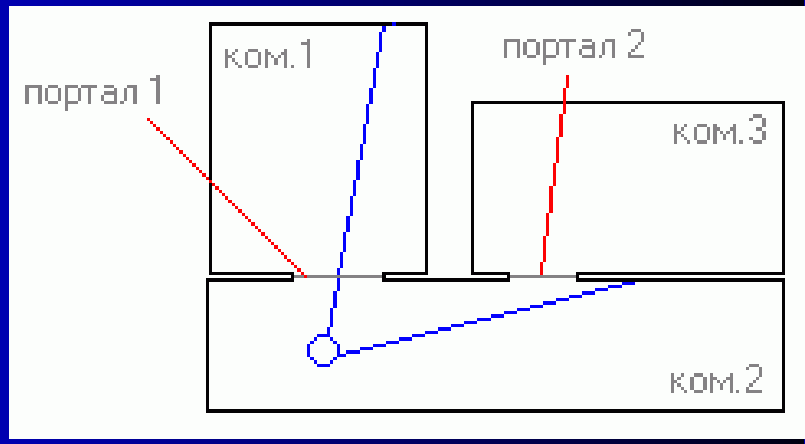
Зачем делить пространство?

Еще один кадр из Naked Empire. Модель здания не имеет внешних стен, что дало возможность заглянуть глубоко во внутрь модели небоскреба. Рендеризация данного кадра потребовала 34 секунды на процессоре в 75 Мгц.



Простейший вариант деления. Комнаты и порталы (cells & portals)

Этот метод создан для так называемых indoor environments - т.е. для замкнутого пространства. Суть его в том, что вся сцена разбивается на комнаты. Комнаты связаны между собой полигонами, которые называются порталами.



Для отрисовки сцены нам надо отрисовать сначала комнату, в которой находится камера, если камера "видит" какой-то портал, то аналогично рисуем комнату, которую видно через этот портал, отсекая ее по проекции полигона-портала

Простейший вариант деления. Комнаты и порталы (cells & portals)

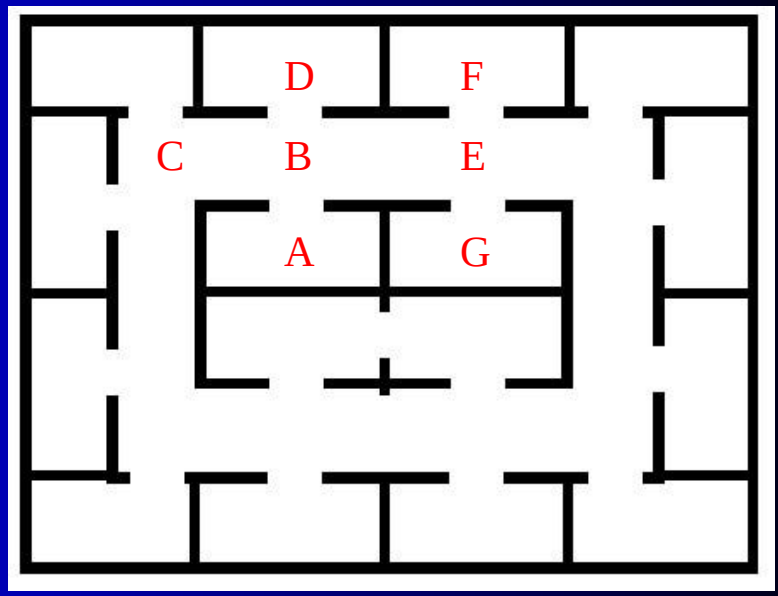
Модель сцены описывается графом:

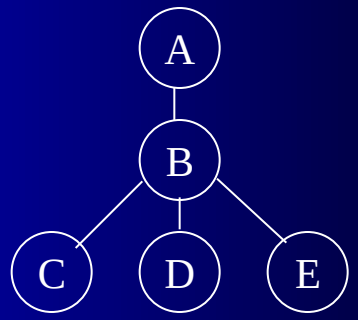
- Узлы: комнаты
- Дуги: порталы (двери)

Граф позволяет определить:

- Потенциально видимое множество
- Главное множество видимых полигонов
- Виды из одной комнаты в другую

И при этом это не сложное решение !





Octree. Восьмеричные деревья

Данный алгоритм производит разделение объектного пространства на восемь подпространств. Общую схему работы можно представить следующими шагами:

1) Помещаем всю сцену в выровненный по осям куб. Этот куб описывает все элементы сцены и является корневым (root) узлом дерева.

2) Проверяем количество примитивов в узле, и если полученное значение меньше определённого порогового, то производим связывание (ассоциацию) данных примитивов с узлом. Узел, с которым ассоциированы примитивы, является листом (leaf).

3) Если же количество примитивов, попадающих в узел, больше порогового значения, производим разбиение данного узла на восемь подузлов (подпространств) путём деления куба двумя плоскостями. Мы распределяем примитивы, входящие в родительский узел, по дочерним узлам. Далее процесс идёт рекурсивно, т. е. для всех дочерних узлов, содержащих примитивы, выполняем пункт 2.

Octree. Восьмеричные деревья

Данный процесс построения дерева может содержать несколько условий прекращения рекурсии:

1) Если количество примитивов в узле меньше или равно пороговому значению.

2) Если рекурсия (количество делений) достигла какой-то определённой глубины вложенности.

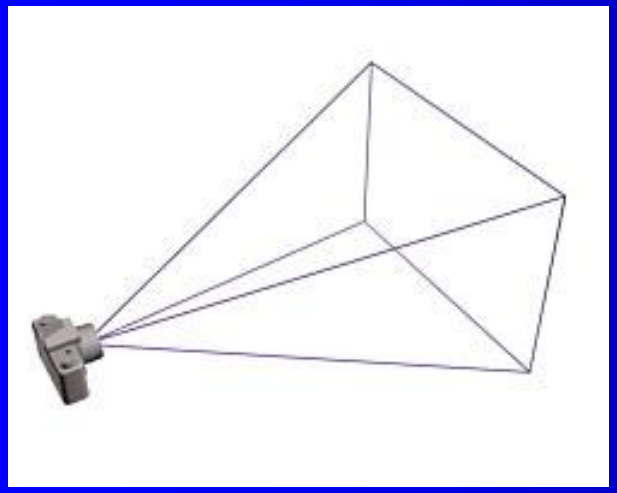
3) Если количество созданных узлов достигло порогового значения.

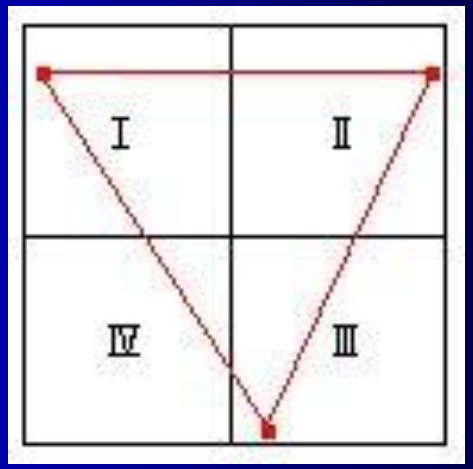
Теперь, в процессе рендеринга мы рекурсивно выполняем следующую процедуру: начиная с базового (root) куба, мы проверяем, попадает ли данный куб в поле зрения (viewing frustum). Если НЕТ - на этом всё и заканчивается, если же ДА - перемещаемся вглубь рекурсии на один шаг, т. е. поочерёдно проверяем видимость каждого из восьми подузлов корневого узла и т. д. Преимущество заключается в том, что если определено, что данный узел не виден, то можно смело не выводить и всю геометрию этого узла - она тоже будет не видна. Таким образом, ценой всего лишь нескольких проверок, мы отбросим значительную часть сцены. А в случае, если не виден корневой узел, на экран не будет выведено ничего. Сплошная экономия!

Octree. Восьмеричные деревья

Недостатки:

Первый из них - это возможное деления примитива ребрами кубов дерева, например, вот так:



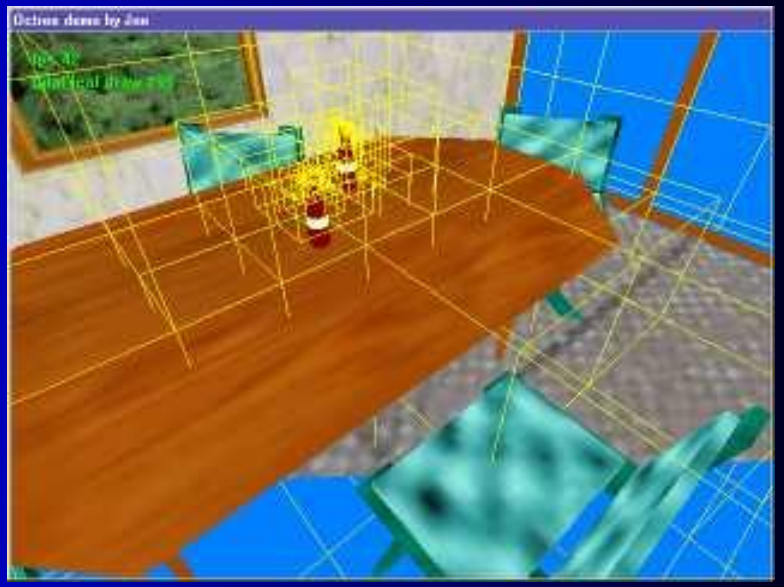


Соответственно с делением примитива есть ещё одна проблема. Допустим, узлы I, II и III не видны. Однако узел IV может всё же попадать в поле зрения камеры, и видно, что частичка полигона всё же может выпасть, вот так:

Octree. Восьмеричные деревья

Недостатки:

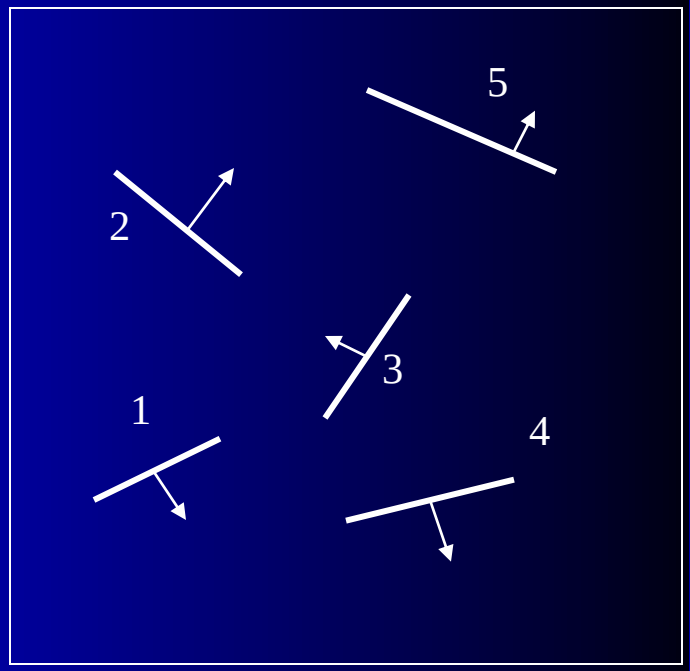
Второй недостаток octree-дерева - это вывод всех объектов, находящихся в поле viewing frustum, но на самом деле в конечном счёте невидимых. Например, стена кухни может закрывать бутылки с кетчупом, но они всё равно будут отсылаться на конвейер (так как находятся в области viewing frustum), и fps будет низким. По-видимому, чистым octree-based алгоритмом здесь не обойтись, необходимо дополнительно реализовать Z-буфер, например.



Выпадение полигона

BSP - деревья
BSP (Binary Space Partitioning) Tree

Данный алгоритм относится к классу "list-priority" алгоритмов – он строит упорядоченный список (дерево) полигонов (их фрагментов) для определённой зафиксированной точки наблюдения (построение производится один раз для сцены на стадии предпроцессигнга(pre-processing)).
Binary Space Partitioning представляет собой рекурсивное иерархическое разбиение в n-мерном пространстве. Построение дерева — процесс, использующий гиперплоскость для разбиения пространства.

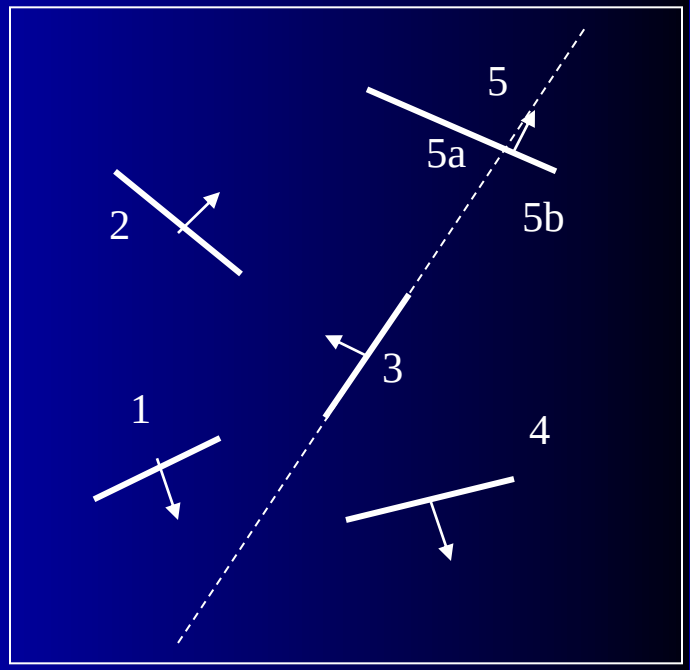


Вид на сцену сверху

Под гиперплоскостью в n-мерном пространстве подразумевается (n-1)-мерный объект, который позволяет разделить пространство на две части. Например, в 3-мерном пространстве гиперплоскость это просто плоскость, а в двумерном — линия.

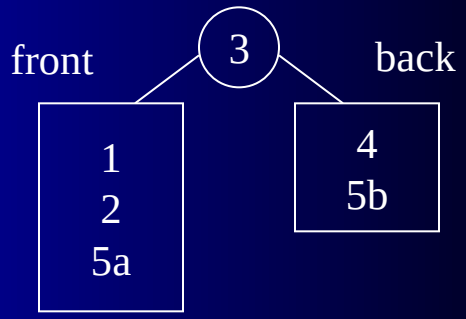
BSP - деревья
BSP (Binary Space Partitioning) Tree

Метод дерева «бинарного разбиения пространства» позволяет эффективно определить видимость объекта, занося поверхности в буфер кадров от фона к переднему плану, как в алгоритме художника. BSP-дерево особенно полезно, когда положение точки наблюдения меняется, но объекты на сцене фиксированы.



Суть алгоритма:

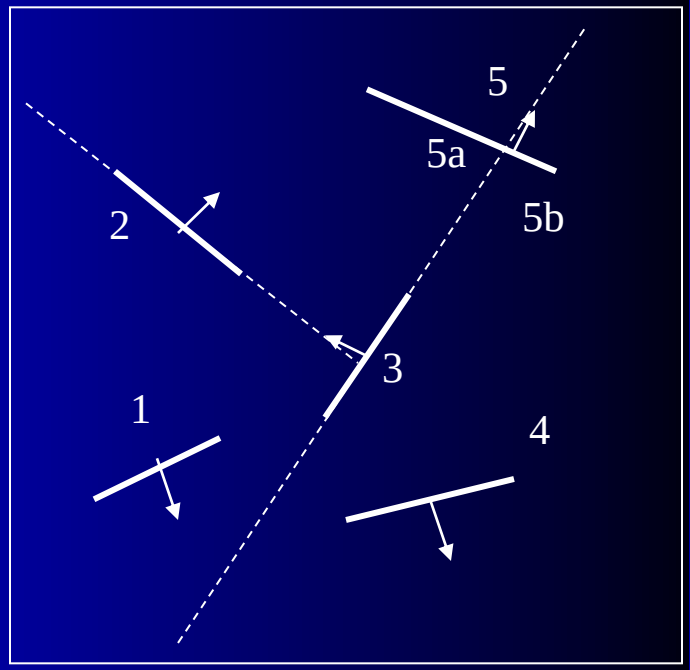
- Выбирается произвольный полигон.
- Сцена (пространство) делится данным полигоном на две части (полупространства) – фронтальную (по направлению его нормали) и заднюю….

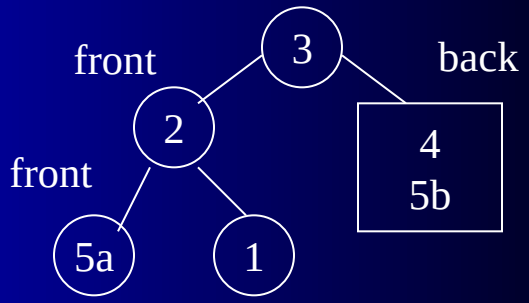


BSP - деревья
BSP (Binary Space Partitioning) Tree

Суть алгоритма:

- Выбирается произвольный полигон.
- Сцена (пространство) делится данным полигоном на две части (полупространства) – фронтальную (по направлению его нормали) и заднюю.
- Любой полигон лежащий в обоих пространствах разбивается на две части.
- Выбирается полигон с каждого полупространства – и каждая полусцена делится опять.
- Рекурсия повторяется до тех пор, пока в каждом узле дерева не останется только по 1 полигону.

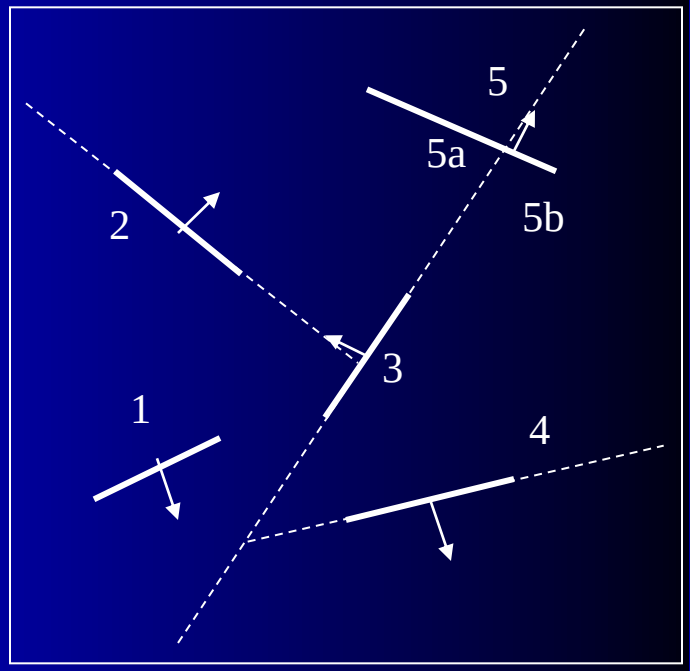


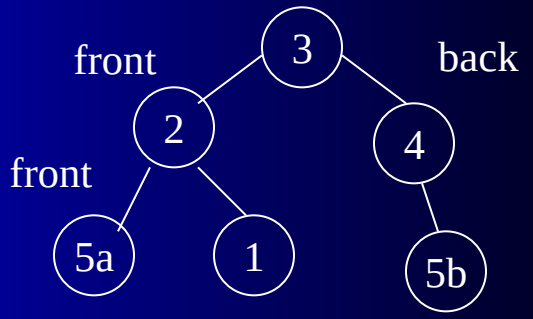


BSP - деревья
BSP (Binary Space Partitioning) Tree

Суть алгоритма:

- Выбирается произвольный полигон.
- Сцена (пространство) делится данным полигоном на две части (полупространства) – фронтальную (по направлению его нормали) и заднюю.
- Любой полигон лежащий в обоих пространствах разбивается на две части.
- Выбирается полигон с каждого полупространства – и каждая полусцена делится опять.
- Рекурсия повторяется до тех пор, пока в каждом узле дерева не останется только по 1 полигону.

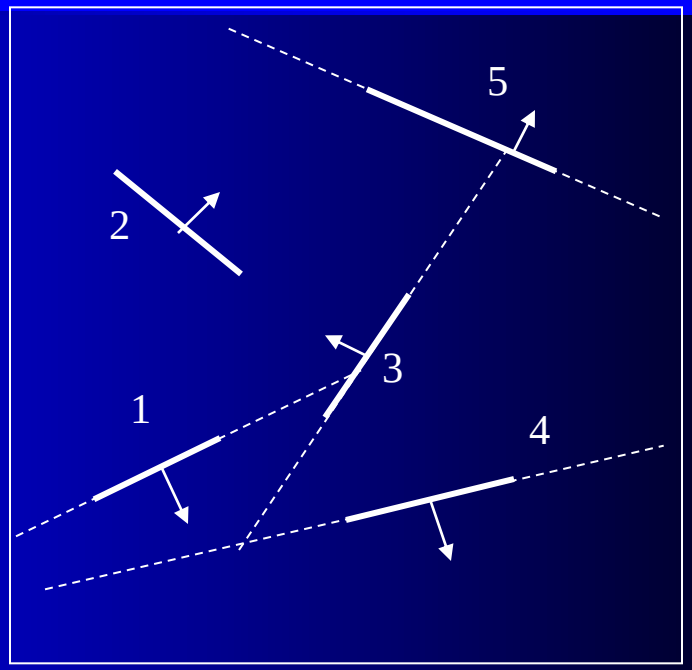


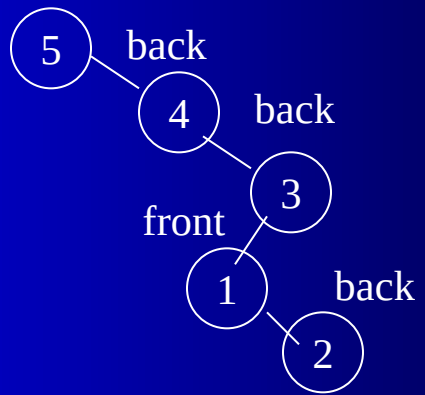


BSP - деревья
BSP (Binary Space Partitioning) Tree

Суть алгоритма:

- Выбирается произвольный полигон.
- Сцена (пространство) делится данным полигоном на две части (полупространства) – фронтальную (по направлению его нормали) и заднюю.
- Любой полигон лежащий в обоих пространствах разбивается на две части.
- Выбирается полигон с каждого полупространства – и каждая полусцена делится опять.
- Рекурсия повторяется до тех пор, пока в каждом узле дерева не останется только по 1 полигону.





Альтернативное дерево, построенное начиная с пл.5

BSP - деревья
BSP (Binary Space Partitioning) Tree

Применение деревьев:

Если мы имеем набор полигонов, то мы нуждаемся в списке их приоритетности для отображения. BSP дерево может быть обработано с целью получения корректного списка приоритетности для произвольной точки наблюдения.

Построенное BSP-дерево обрабатывается в следующем порядке – сначала правые узлы, затем левые. Таким образом поверхности подготавливаются к отображению в порядке от фона к переднему плану, так что объекты переднего плана рисуются поверх фоновых.

Начиная с корневого (вершины дерева) полигона:

- Если наблюдатель находится перед плоскостью, то сперва отрисовываем полигоны, находящиеся позади корневого полигона, затем сам корневой полигон, а затем полигоны, находящиеся перед корневым.
- Рекурсивно обходим всё дерево.
- Если наблюдатель находится позади плоскости, то данный полигон может быть забракован (не отображаться).

BSP - деревья
BSP (Binary Space Partitioning) Tree

- В классическом алгоритме BSPдеревьев во главу угла ставится минимальность количества узлов дерева – все равно выводятся все полигоны.
- Однако для модифицированных алгоритмов на основе BSP-деревьев вывод всех полигонов – непозволительная роскошь. В этом случае сбалансированное дерево (симметричное относительно как можно большего количества узлов) позволяет отсечь невидимые плоскости и не выводить их в принципе.

BSP - деревья
BSP (Binary Space Partitioning) Tree

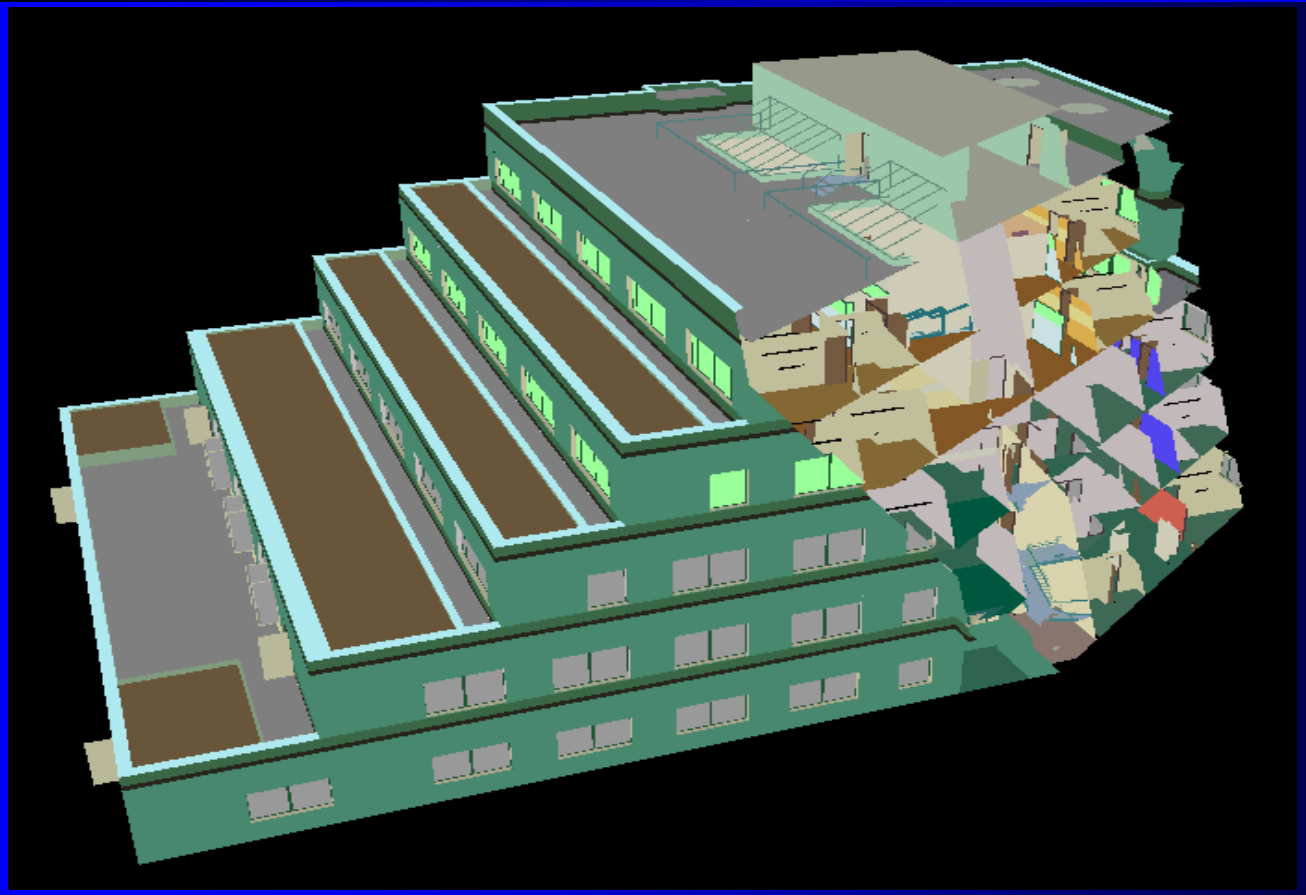
Недостатки:

- Большой объём вычислений на начальном этапе.
  - Вариантов построения деревьев для одной сцены может быть достаточно много. Определение оптимальных секущих плоскостей может быть затруднительным и потребовать перебора всех возможных вариантов.
  - Разделение полигонов сцены на части секущим может быть трудоёмким

Достоинства:

- Проверка видимости полигона по уже построенному дереву – не требует значительных вычислений.
- Одно и тоже дерево может быть использовано для проверки видимости полигонов с разных точек наблюдения.
- Таким образом, подход особенно эффективен, в случаях когда объекты сцены не часто меняются.

Пример необходимости разделения пространства – Архитектурные приложения

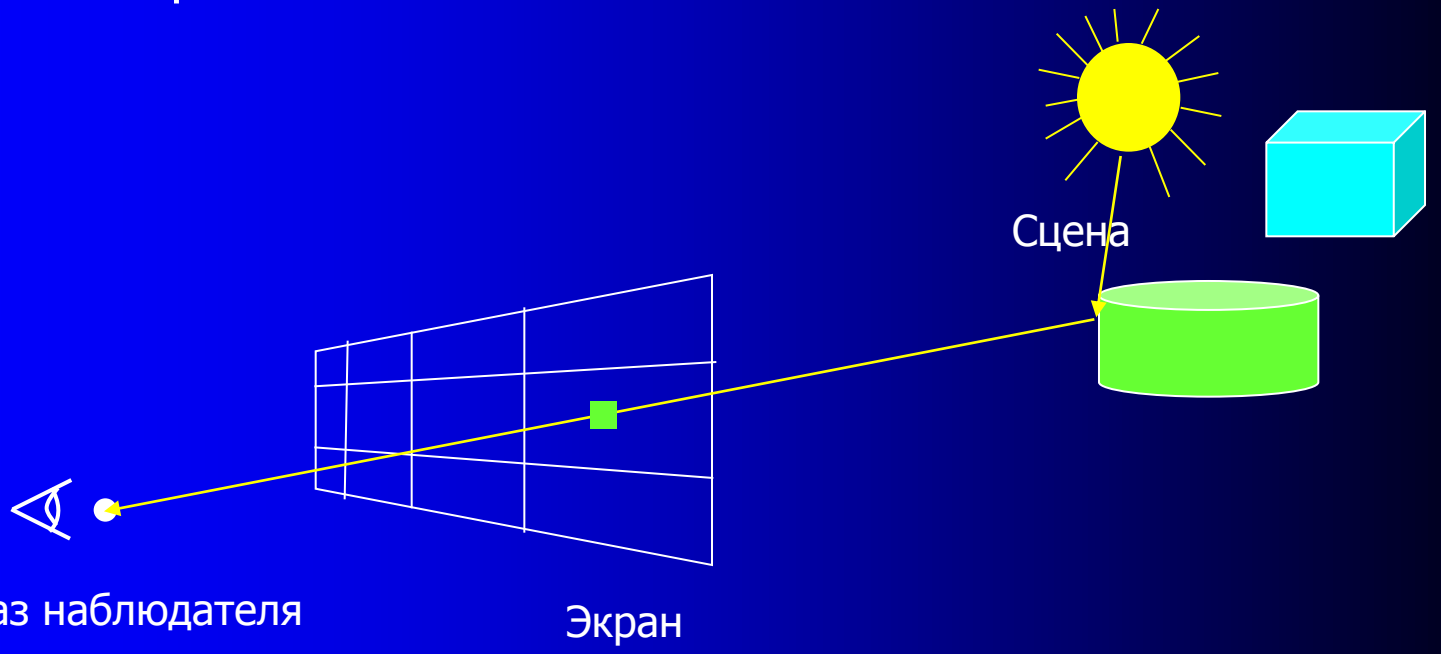


В данном случае сцена содержит «ненормально» большое количество стыковок полигонов.

Прямая трассировка лучей (Ray casting)

Не путать с «обратной трассировке лучей» (*Ray-tracing)*!

Подход использует воображаемые лучи, исходящие из источников освещения, проходящих через центр каждого пикселя экрана и попадающих в глаза наблюдателя.



Прямая трассировка лучей (Ray casting)

- Вычисление пересечений луча и объекта – основа алгоритма трассировки лучей.
- Пример – пересечение со сферой (простейший случай).

Уравнение луч - в параметрическом виде :

$$x = x_0 + t\Delta x \;\; ; \;\; y = y + t\Delta y \;\; ; \;\; z = z + t\Delta z$$

Уравнение сферы :

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

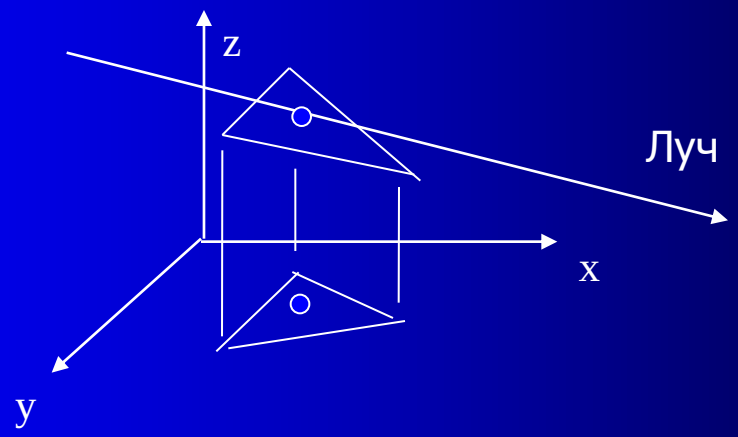
1. Перевести уравнение сферы в параметрическую форму.
2. Получить квадратичное уравнение пересечения луча и сферы.
3. Решить его:
   - -Если корней нет – луч не пересекает сферу.
   - -Если 1 корень – луч касается сферы.
   - -Если 2 корня – луч проходит сферу насквозь.

Прямая трассировка лучей (Ray casting)

С полигонами не всё так легко.

1. Сперва требуется определить пересекает ли луч плоскость полигона.
2. Затем – лежит ли точка пересечения внутри полигона.

Для проверки второго пункта лучше использовать ортографическую проекцию на подходящую координатную плоскость и проверять двухмерный случай теста «точка внутри многоугольника».



Прямая трассировка лучей (Ray casting)

- Легко применим для широкого круга примитивов – требуется только уравнение пересекаемой поверхности.
- Пиксель экрана принимает значение цвета ближайшей пересекаемой поверхности.
- Может рисовать кривые и поверхности точно – не требуется разбивка поверхностей на полигоны (треугольники) !
- Могут генерироваться новые лучи внутри сцены для корректного отображения отражений, преломлений, затенений – но алгоритм обратной трассировки лучей более эффективен.
- Может быть модифицирован для отображения общего освещения сцены.
- Может производить улучшенное качество изображений и использованием нескольких лучей на пиксель.
- Но … слишком не эффективен для приложений реального времени (real-time.)

Примеры изображений, полученных трассировкой лучей