

Лекция 8

Деревья разбиения пространства (space partition trees)

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com

WWW: www.mkurnosov.net

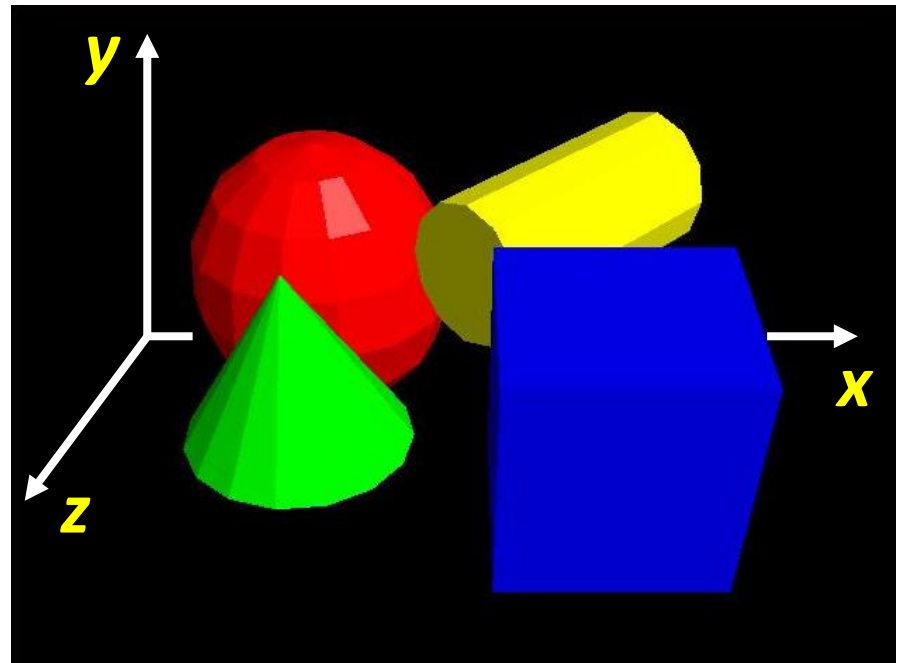
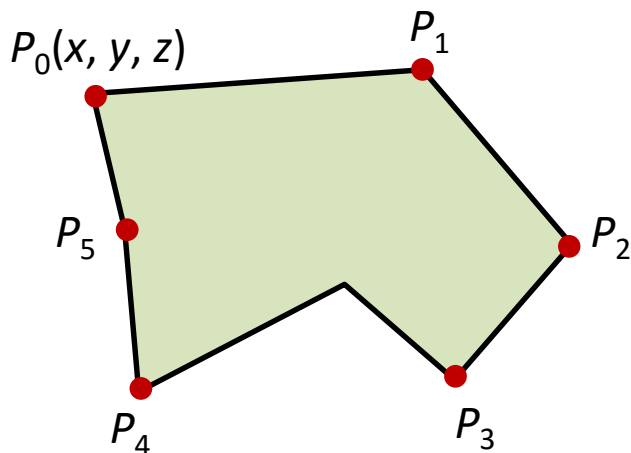
Курс «Структуры и алгоритмы обработки данных»

Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)

Осенний семестр, 2015

Визуализация трехмерных сцен

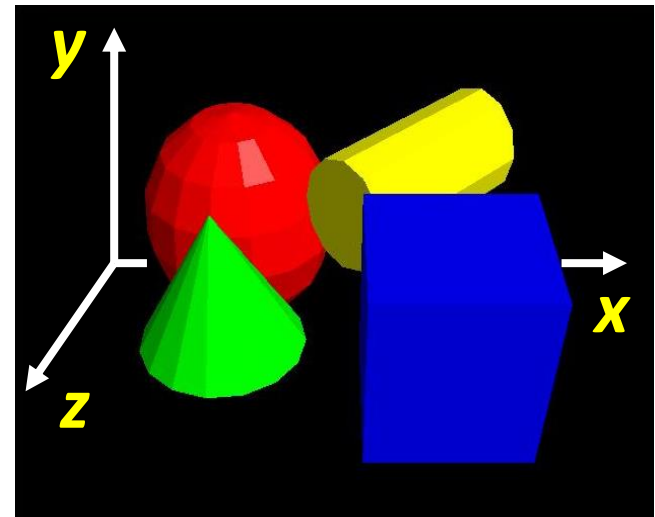
- Имеется трехмерное пространство, в котором размещено n полигонов (многоугольников) и известны их координаты
- Полигон (polygon) – это многоугольник, замкнутая ломанная линия (задает плоскость, на которой лежит)



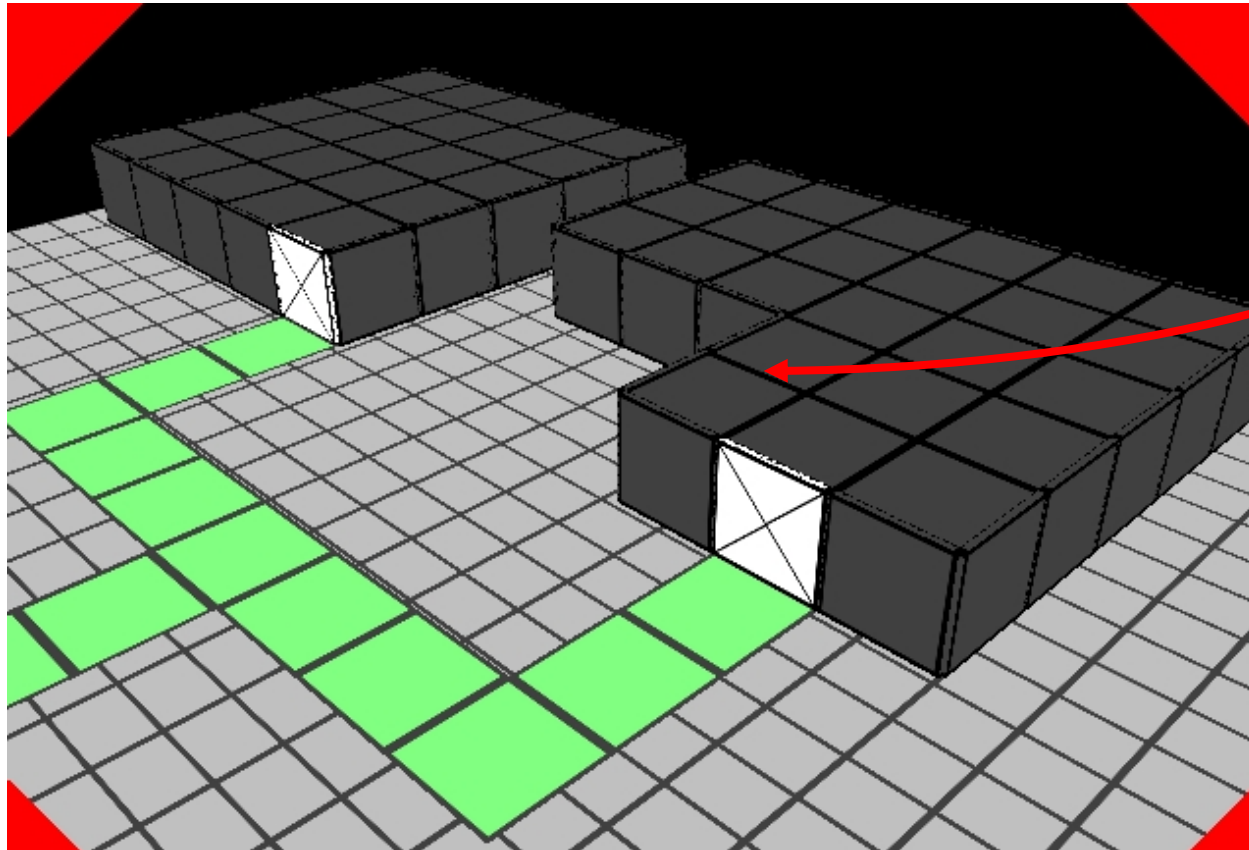
- Сложные поверхности представляются множеством полигонов

Визуализация трехмерных сцен

- Имеется трехмерное пространство, в котором размещено n полигонов (многоугольников) и известны их координаты
- Требуется решать следующие задачи:
 - Сортировать объекты (полигоны) в порядке удаления от наблюдателя (камеры)
 - Обнаруживать столкновения объектов

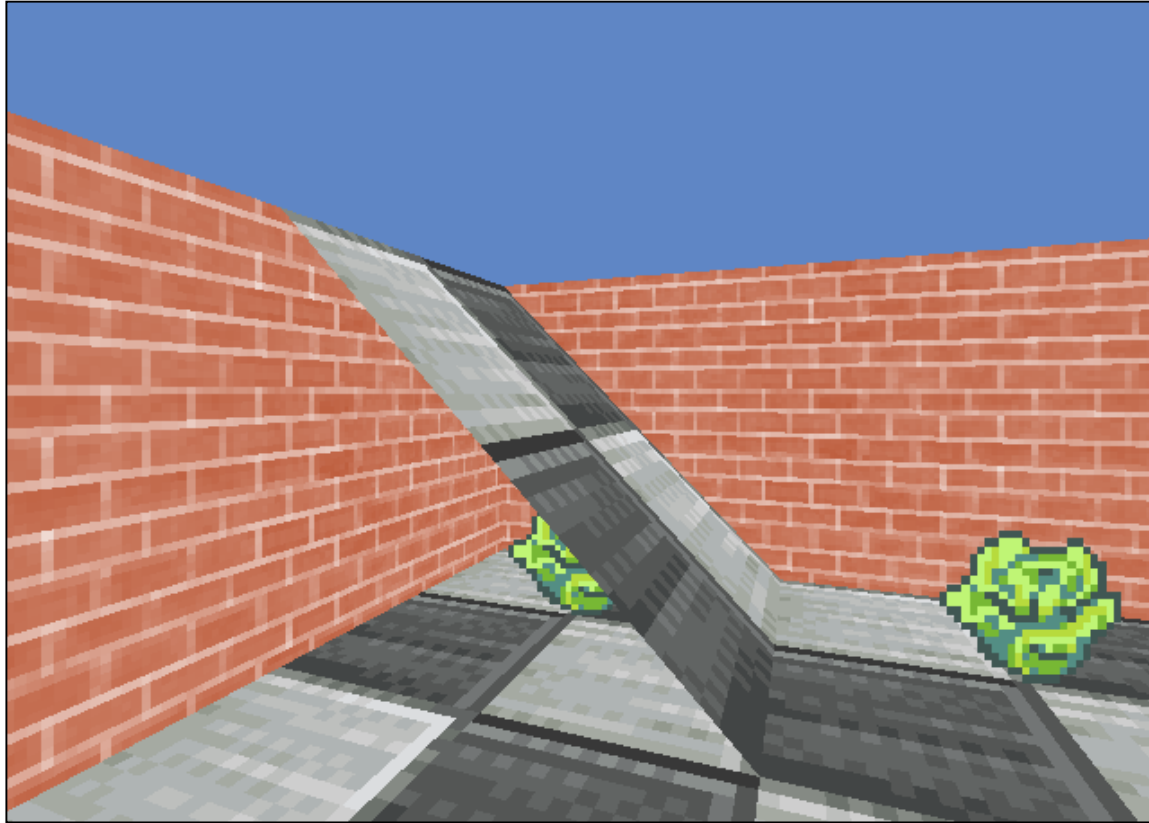


Визуализация трехмерных сцен



- Невидимые поверхности (полигоны) не отрисовываются
- Видимость определяется порядком расположения полигонов (от камеры)

Визуализация трехмерных сцен



Объекты нарисованы с учетом их расположения относительно наблюдателя (камеры)

Деревья двоичного разбиения пространства

- **Дерево двоичного разбиения пространства (binary space partitioning tree, BSP tree)** – это бинарное дерево, разделяющее пространство на подмножества
- **BSP tree** хранит информацию о расположении объектов сцены в упорядоченном порядке – от переднего плана (front) к заднему (back)
- Также применяется для определения столкновений объектов (collision detection) в компьютерных играх и робототехнике
- Использовались в играх Doom (1993), Quake (1996) и др.
- **Авторы:**
 - ❑ Schumacker R. A., Brand B., Gilliland M. G., Sharp W.H., **1969**
 - ❑ Fuchs H., Kedem Z. M, Naylor B. F., **1980**

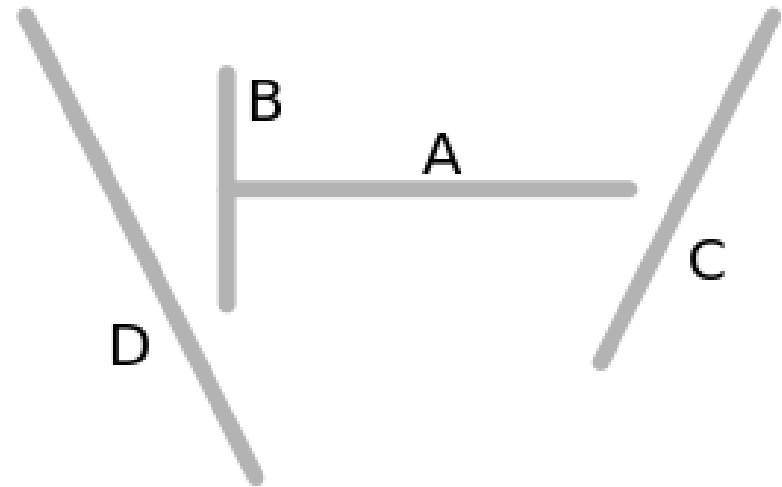
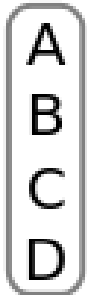
Построение BSP-дерева

1. Из заданного списка P полигонов выбираем разбивающий полигон S , он задает разбивающую плоскость (plane)
2. Создаем в дереве новый узел *node*
3. Для каждого полигона P в списке полигонов
 - a) Если полигон P находится с фронтальной стороны S , заносим его в список полигонов фронтальной стороны (front list)
 - b) Если полигон P находится с обратной стороны S , заносим его в список полигонов обратной стороны (back list)
 - c) Если полигон P пересекается плоскостью S , разделяем его на два и заносим их в списки front и back
 - d) Если полигон P лежит в плоскости S , то добавляем его в список полигонов узла *node*
4. Рекурсивно применяем алгоритм к списку front, затем к back

Пример построение BSP-дерева в 2D случае

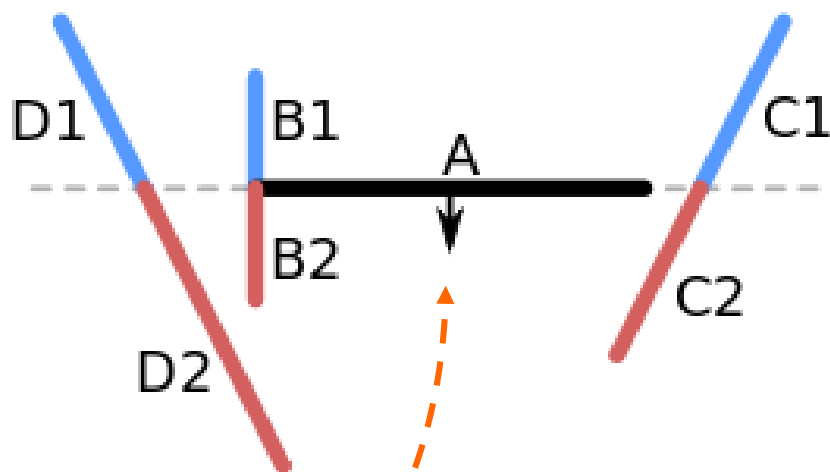
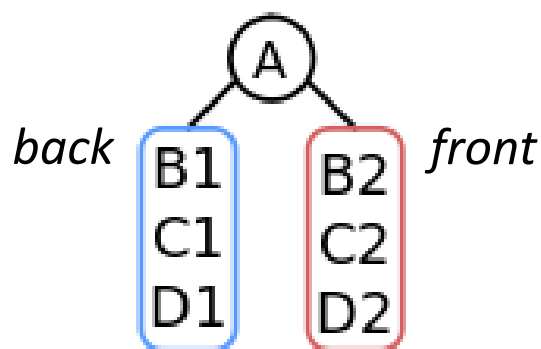
1. В двумерном пространстве задан список прямых (A, B, C, D) – список P
2. Требуется построить BSP-дерево

list P



Пример построение BSP-дерева в 2D случае

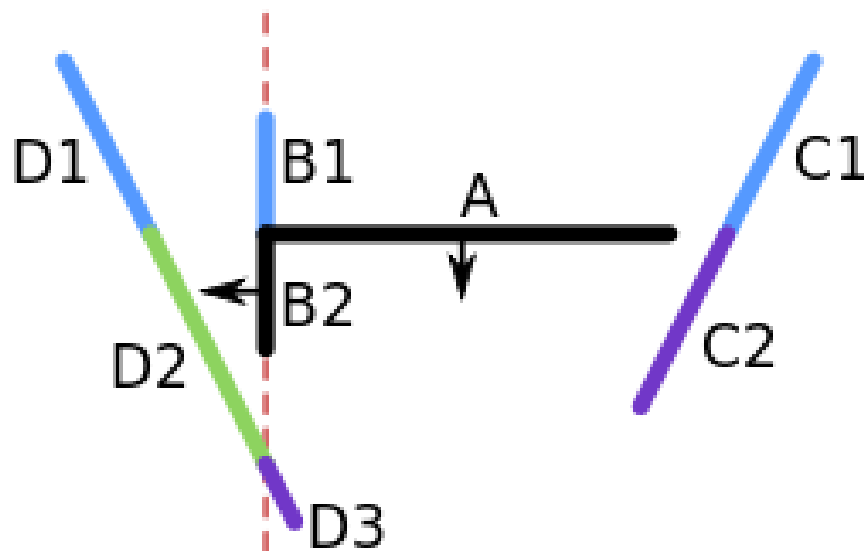
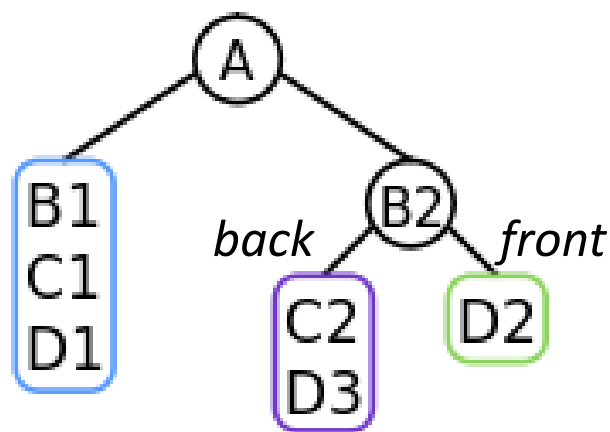
1. Из списка (A, B, C, D) выбираем разделяющую прямую A
2. Создаем новый узел A – корень BSP-дерева
3. Формируем списки *front* и *back*
4. Рекурсивно обрабатываем списки *front* и *back*



Вектор указывает
в сторону переднего плана (front)

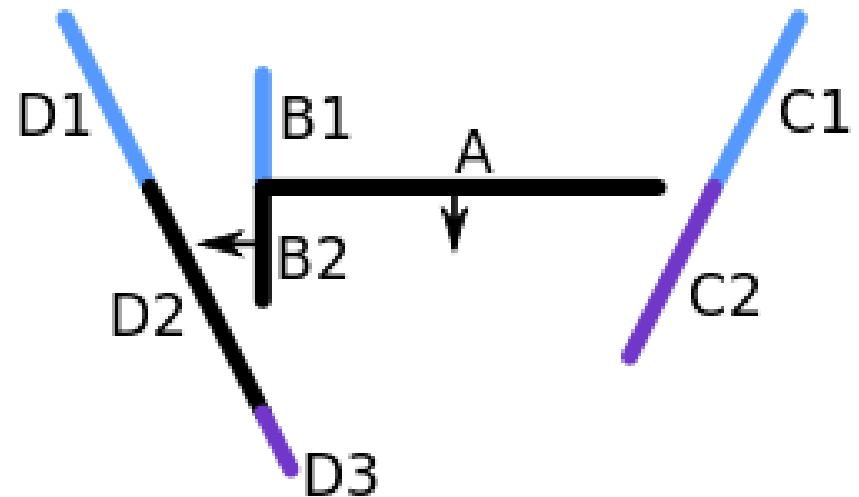
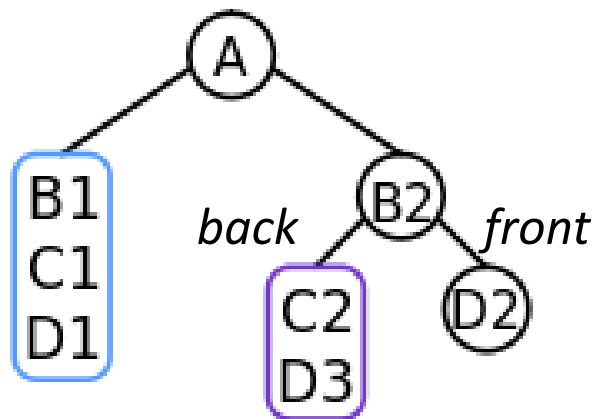
Пример построение BSP-дерева в 2D случае

- Из списка *front* узла A выбираем разделяющую прямую B_2
- Создаем узел B_2 и формируем его списки *front* и *back*



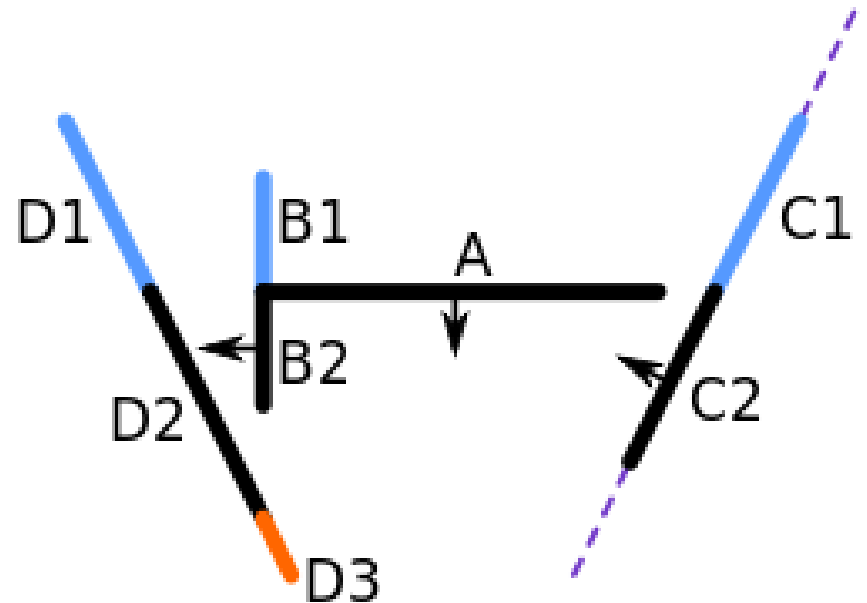
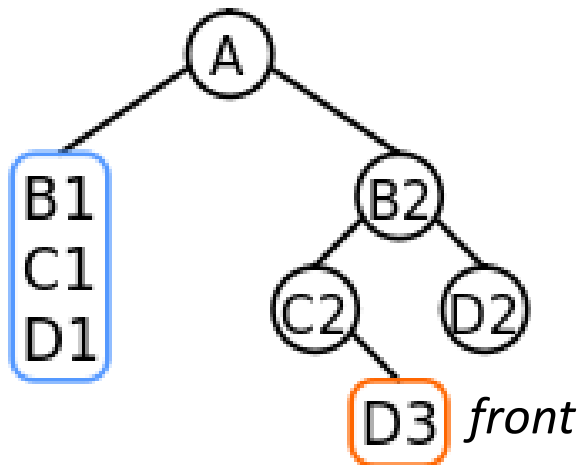
Пример построение BSP-дерева в 2D случае

- Из списка *front* узла B_2 выбираем разделяющую прямую D_2
- Формируем узел D_2



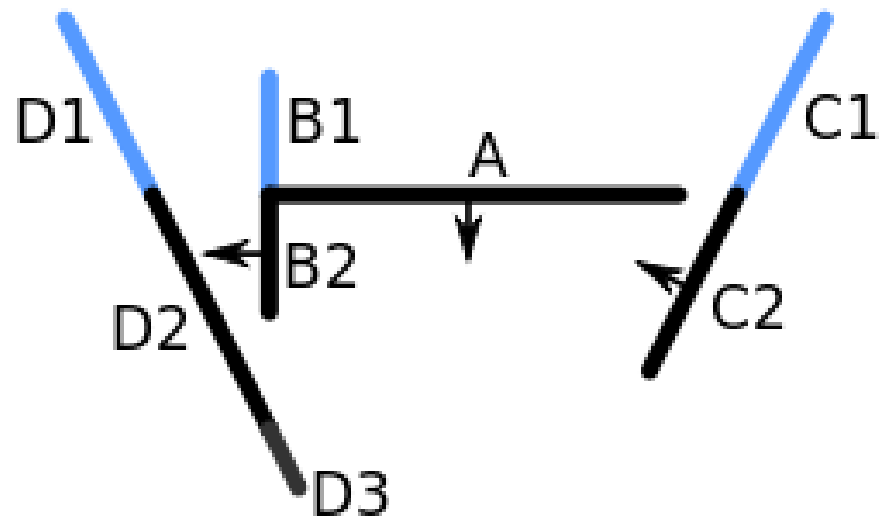
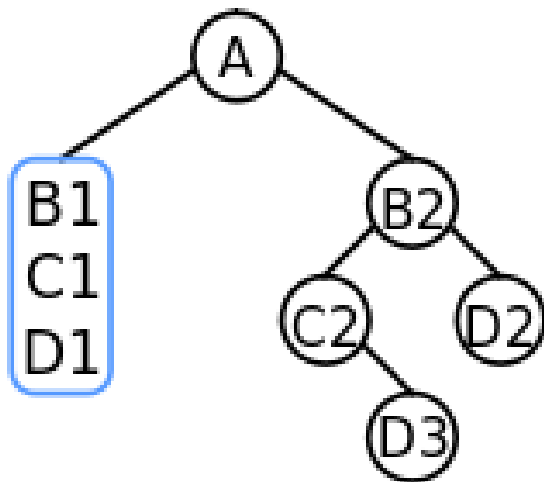
Пример построение BSP-дерева в 2D случае

- Из списка *back* узла B_2 выбираем разделяющую прямую C_2
- Формируем узел C_2 и его список *front* (список *back* пуст)



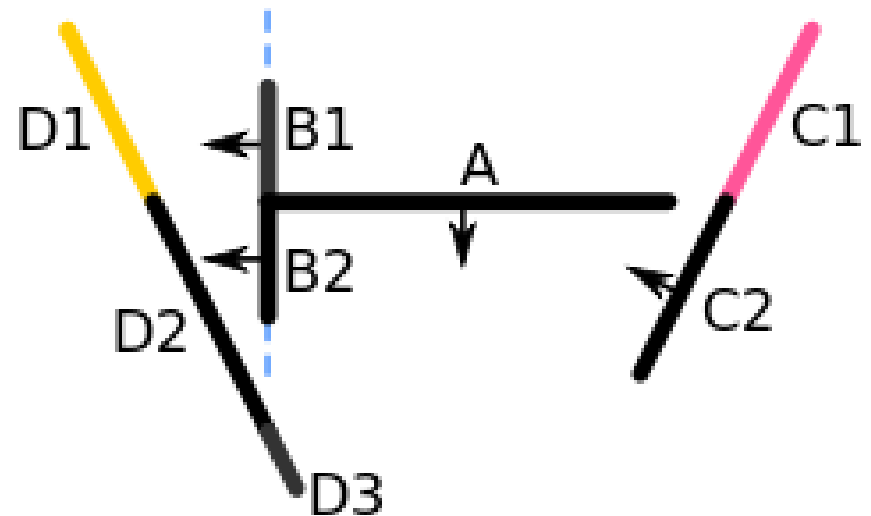
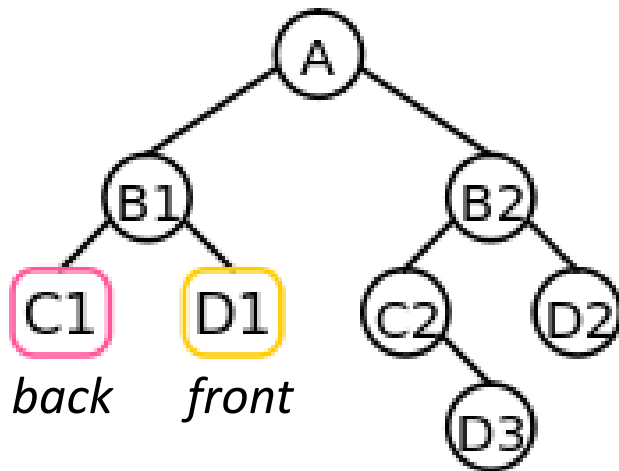
Пример построение BSP-дерева в 2D случае

- Из списка *front* узла C_2 выбираем разделяющую прямую D_3
- Формируем узел D_3



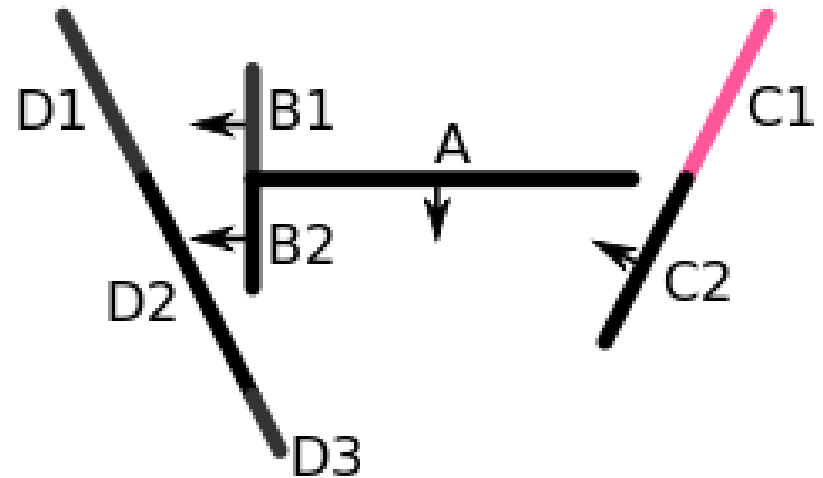
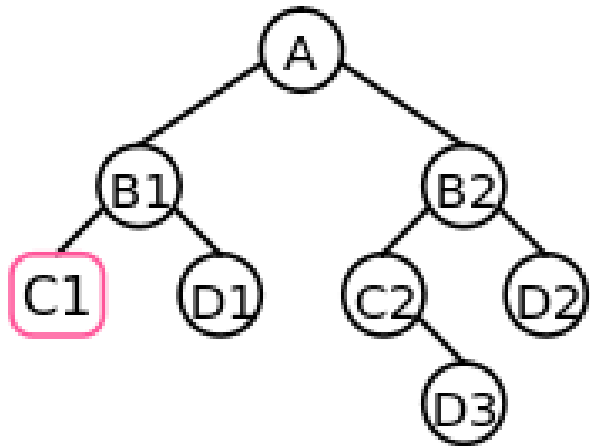
Пример построение BSP-дерева в 2D случае

- Из списка *back* узла A выбираем разделяющую прямую B_1
- Формируем узел B_1 и его списки *front* и *back*



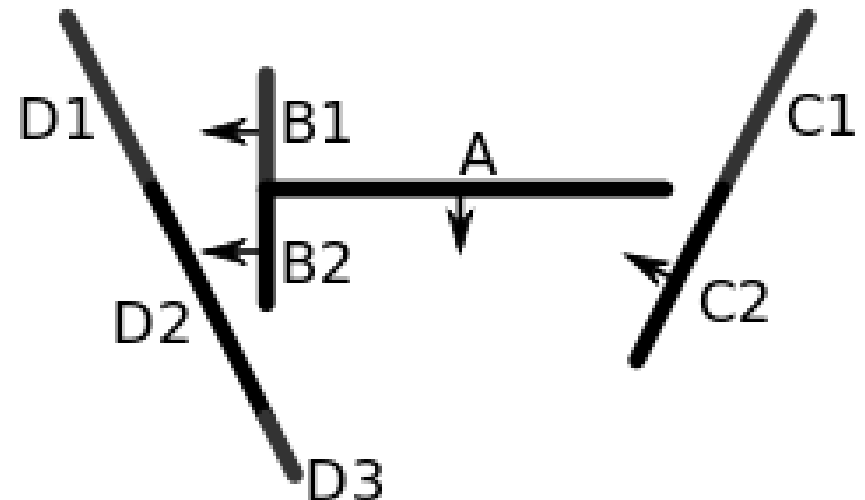
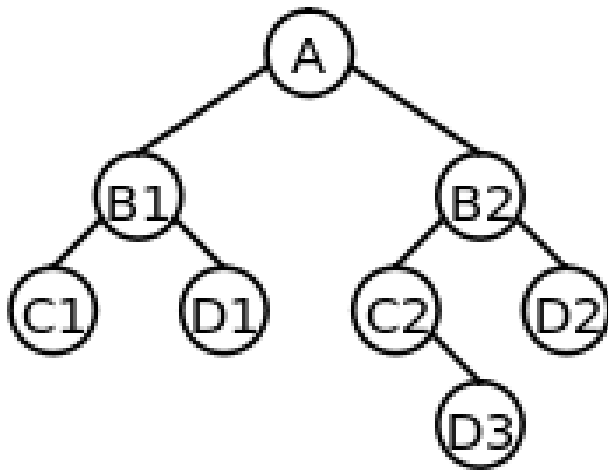
Пример построение BSP-дерева в 2D случае

- Из списка *front* узла B_1 выбираем разделяющую прямую D_1
- Формируем узел D_1



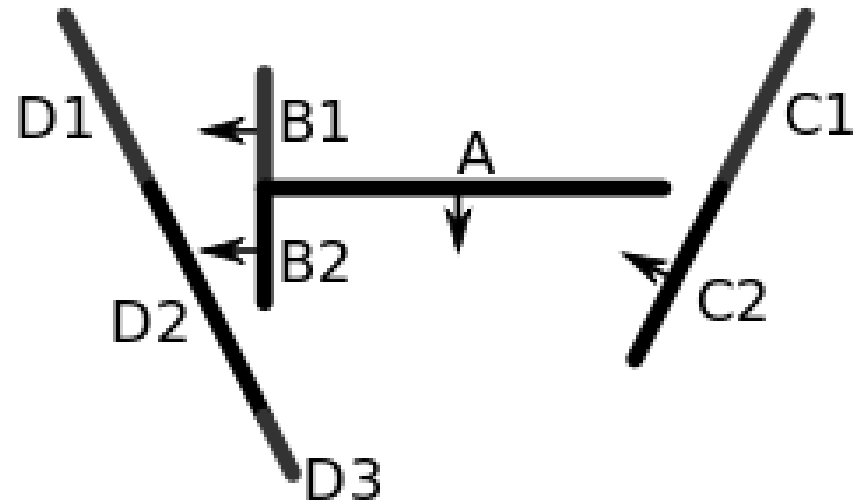
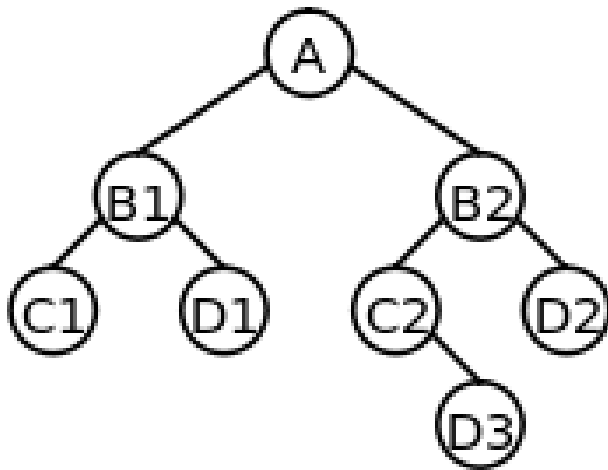
Пример построение BSP-дерева в 2D случае

- Из списка *back* узла B_1 выбираем разделяющую прямую C_1
- Формируем узел C_1



Пример построение BSP-дерева в 2D случае

- Количество полигонов в дереве больше их начального числа
- На плоскости было 4 прямые, в дереве 8



Сортировка полигонов

- Задано BSP-дерево
- Заданы координаты наблюдателя – POV (point of view)
- **Как изобразить на экране полигоны в порядке удаления от наблюдателя (упорядочить их)?**

Обход BSP-дерева (сортировка полигонов)

1. Если текущий узел лист, нарисовать его полигоны
2. Если наблюдатель (камера) расположен перед текущим узлом
 - рекурсивно обойти поддерево с узлами, находящимися позади текущего узла
 - нарисовать полигоны текущего узла
 - рекурсивно обойти поддерево с узлами, находящимися впереди текущего узла
3. Если наблюдатель (камера) расположен позади текущего узла
 - рекурсивно обойти поддерево с узлами, находящимися впереди текущего узла
 - нарисовать полигоны текущего узла
 - рекурсивно обойти поддерево с узлами, находящимися позади текущего узла
4. Если наблюдатель расположен в плоскости текущего узла
 - рекурсивно обойти поддерево с узлами, находящимися впереди текущего узла
 - рекурсивно обойти поддерево с узлами, находящимися позади текущего узла

Поиск столкновений

- Задано BSP-дерево
- Заданы координаты объекта
- Границы объекта задаются ограничивающей сферой (или окружностью) для упрощения вычислений
- **Требуется найти полигон пересекающий ограничивающую сферу объекта – полигон, с которым столкнулся объект**

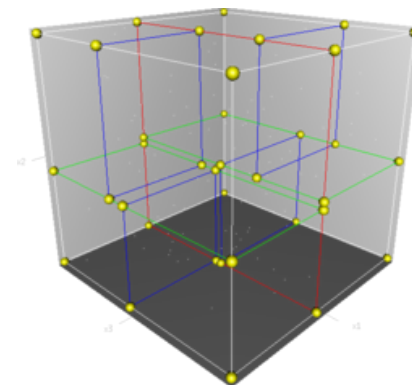
Поиск столкновений

```
function FindCollision(node, obj)
  if node = NULL then
    return NULL

  if Dist(node.polygon, obj.sphere.center) > obj.sphere.radius then
    if DotProduct(obj.center, node.normal) >= 0 then
      // Объект находится с фронтальной стороны плоскости,
      // обходим только фронтальное поддерево
      return FindCollision(node.front, obj)
    else
      // Объект находится с обратной стороны разбивающей плоскости,
      // обходим только обратное поддерево
      return FindCollision(node.back, obj)
    end if
  else
    // Столкновение с полигоном узла node
    return node
  end if
end function
```

***k*-мерное дерево (*k*-d tree)**

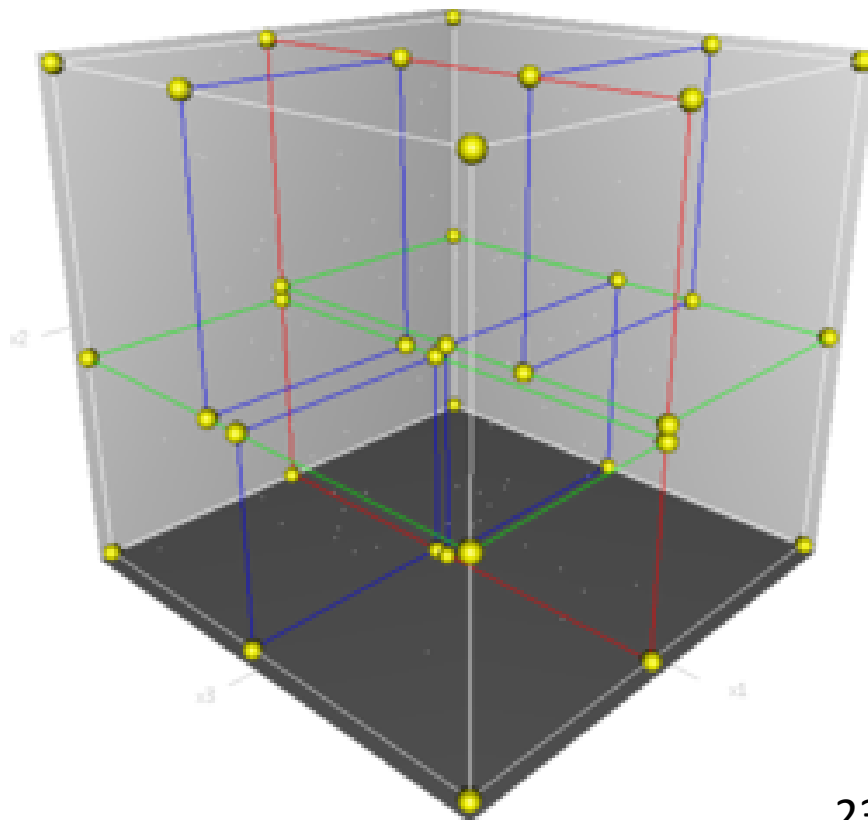
- ***k*-мерное дерево (*k*-d tree)** – это дерево разбиения пространства для упорядочивания точек в *k*-мерном пространстве
- *k*-d дерево – это разновидность дерева поиска
- **Автор:** Дж. Бентли, 1975



	Сложность в среднем случае	Сложность в худшем случае
Lookup	$O(\log n)$	$O(n)$
Insert	$O(\log n)$	$O(n)$
Delete	$O(\log n)$	$O(n)$
Объем требуемой памяти: $O(n)$		

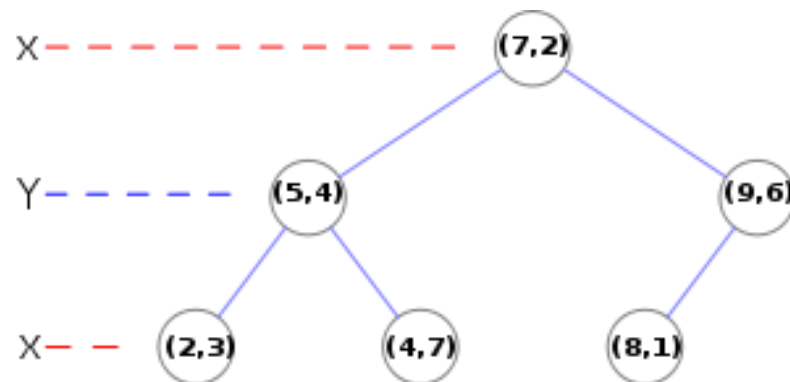
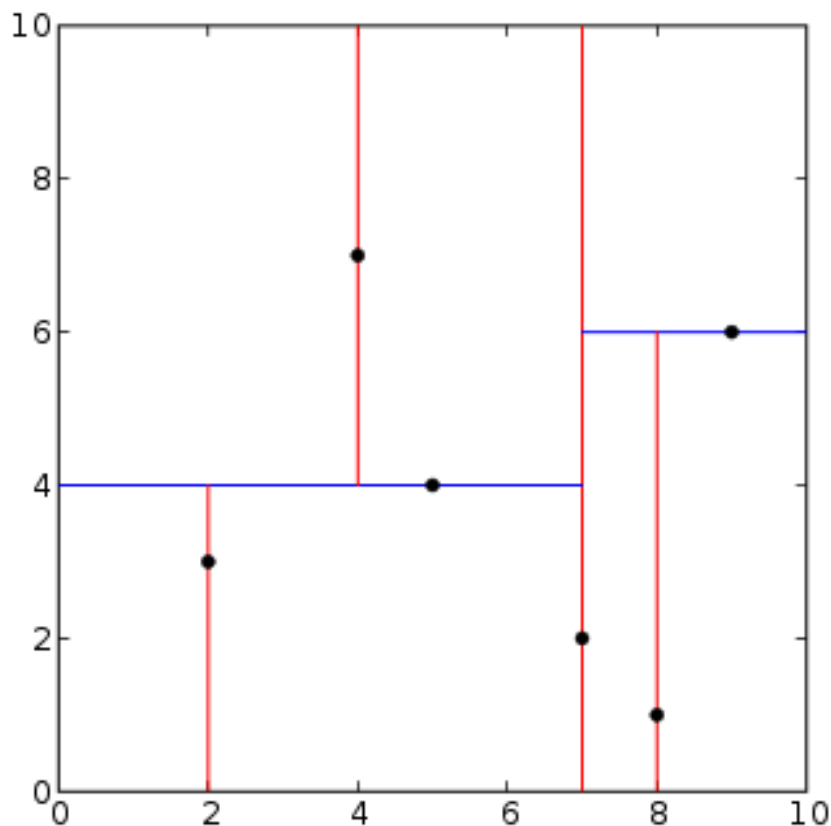
k -мерное дерево (k -d tree)

- k -мерное дерево (k -d tree) – это бинарное дерево разбиения пространства для упорядочивания точек в k -мерном пространстве
- Пространство разбивается гиперплоскостью на два подпространства
- k -d дерево строится для заданного множества точек



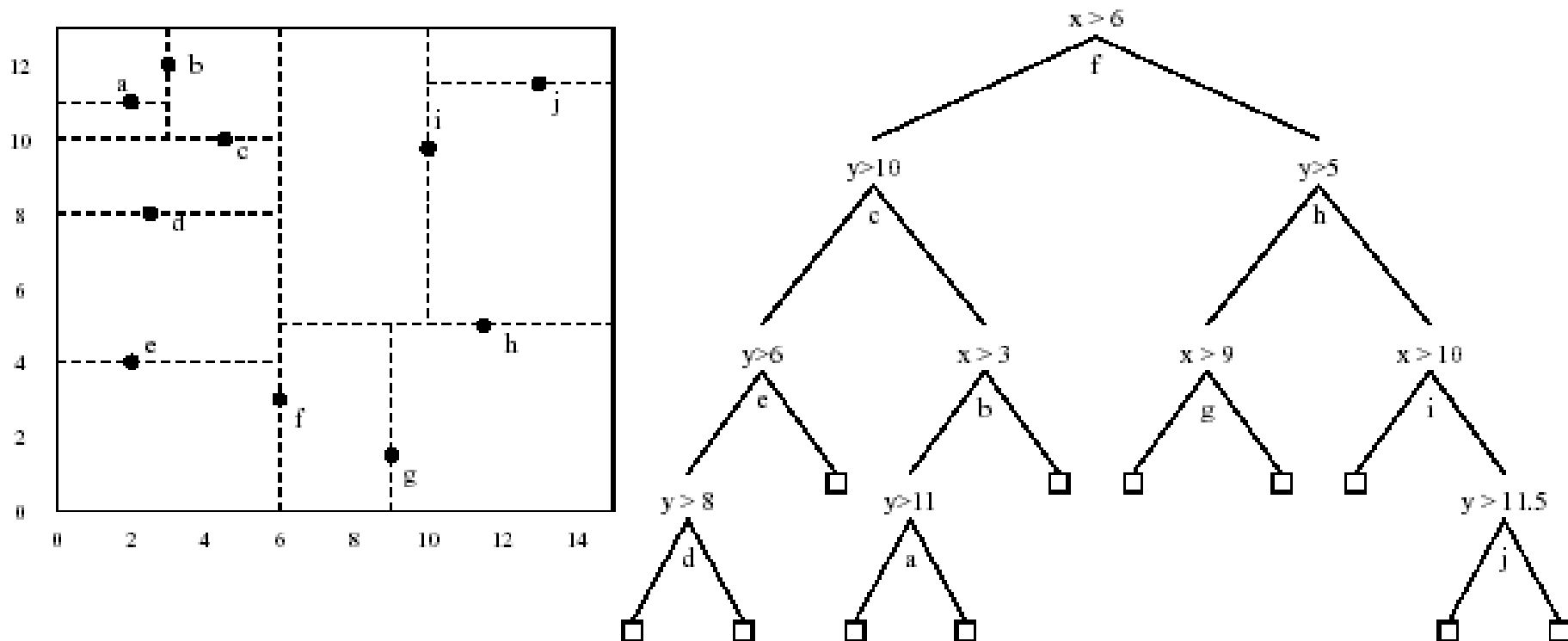
k -мерное дерево (k -d tree)

- Заданы точки $(2,3)$, $(5,4)$, $(9,6)$, $(4,7)$, $(8,1)$, $(7,2)$
- Строим k -d дерево рекурсивно разбивая плоскость поочередно прямыми $x = k$, $y = k$
- За k принимается медиана среди координат X или Y точек



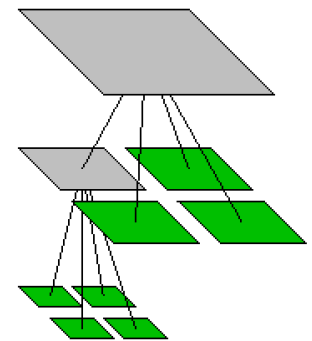
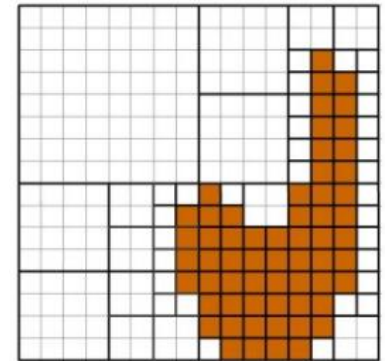
k -мерное дерево (k -d tree)

- k -мерное дерево (k -d tree) – это бинарное дерево разбиения пространства для упорядочивания точек в k -мерном пространстве

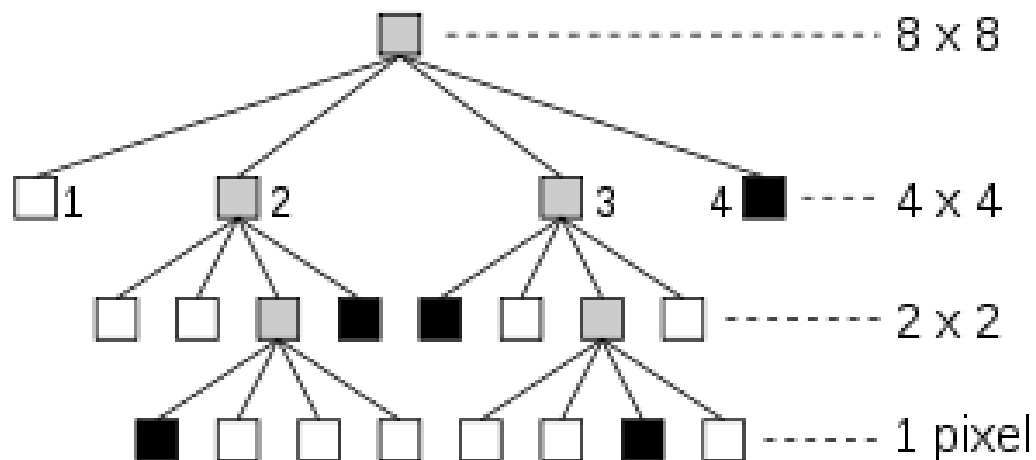
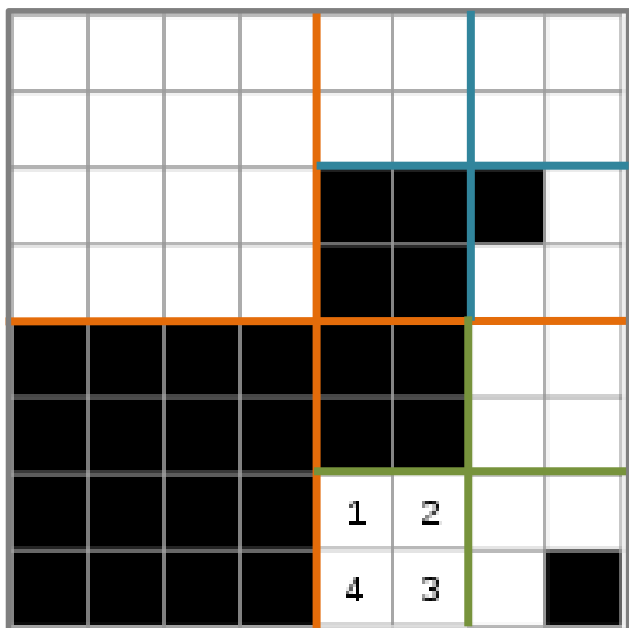


Дерево квадрантов (quadtree)

- **Дерево квадрантов (quadtree)** – это дерево, в котором каждый внутренний узел содержит 4 дочерних элемента (квадранта)
- Дерево Quadtree – используется для разбиения 2D пространства
- Авторы: Raphael Finkel, Jon Bentley, 1974
- Задачи
 - Представление изображений
 - Обнаружение столкновений (ближайших точек/объектов)
 - Хранение данных для табличных или матричных вычислений



Дерево квадрантов (quadtree)



Октодерево (octree)

- **Октодерево (octree)** – это трехмерный аналог дерева квадрантов

