Diskrétne geometrické štruktúry

4.

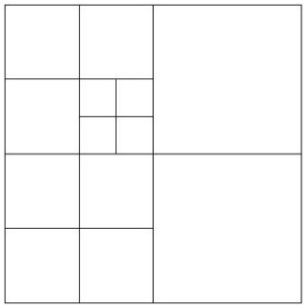
Martin Florek

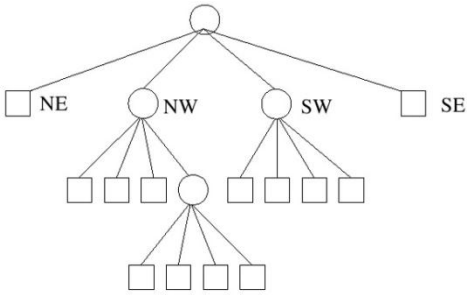
florek@sccg.sk

www.sccg.sk/~florek

Quadtree

- Každý vnútorný vrchol má práve štyroch potomkov
- Vrchol predstavuje najčastejšie štvorec alebo obdĺžnik, môžu byť aj iné tvary
- Štyria potomkovia vrcholu predstavujú rozdelenie vrcholu na 4 časti (subdivision)





Množina bodov

- *P* – množina bodov v 2D
- Rozdelenie až kým štvorec neobsahuje max. jeden bod

```
QuadTreeNodeConstruct(P, left, right, bottom, top)
{
        v = new QuadTreeNode;
        V->left = left; v->right = right; v->bottom = bottom; v->top = top;
        if (|P| == 0) return v;
        if (|P| == 1)
        {
                v->point = P.first;
                return v;
        }
        xmid = (left + right)/2; ymid = (bottom + top)/2;
        (NE, NW, SW, SE) = P.Divide(midx, midy);
        v->NE = QuadTreeNodeConstruct(NE, xmid, right, ymid, top);
        v->NW = QuadTreeNodeConstruct(NW, left, xmid, ymid, top);
        v->SW = QuadTreeNodeConstruct(SW, left, xmid, bottom, ymid);
        v->SE = QuadTreeNodeConstruct(SE, xmid, right, bottom, ymid);
        v->NE->parent = v; v->NW->parent = v;
        v->SW->parent = v; v->SE->parent = v;
        return v;
}
```

```
struct QuadTreeNode
{
        Point point;
        float left, right, bottom, top;
        QuadTreeNode * parent;
        QuadTreeNode * NE;
        QuadTreeNode * NW;
        QuadTreeNode * SW;
        QuadTreeNode * SE;
}
```

Vlastnosti

- Hĺbka quadtree je najviac Log*(s/c)* + *3/2*, kde *c* je najmenšia vzdialenosť bodov z *P* a *s* je dĺžka strany počiatočného štvorca
- Quadtree hĺbky *d* s |*P*|=*n* má O*(n.(d+1))* vrcholov a dá sa vytvoriť v čase O*(n.(d+1))*

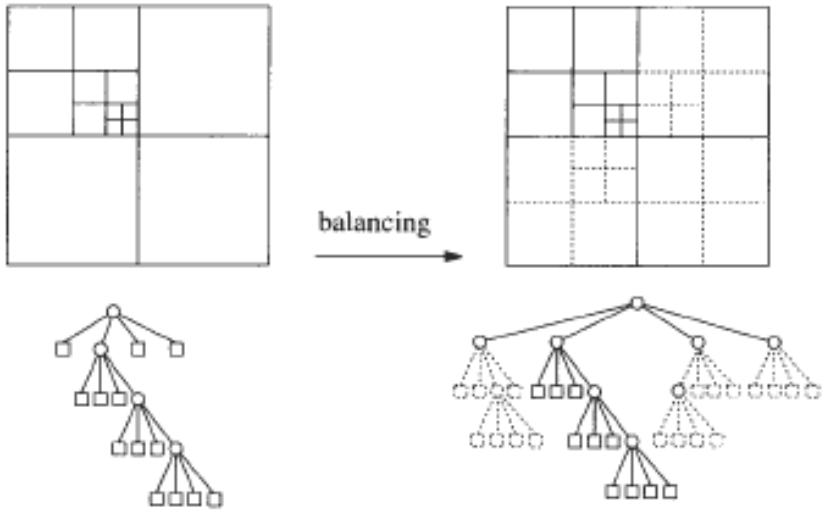
Vyhľadanie suseda

- Pre daný vrchol a korešpondujúci štvorec, nájdi štvorec susediaci v danom smere na rovnakej úrovni
- Časová zložitosť O(*d+1*)

```
NorthNeighbor(v, T)
{
        if (v == T->root) return NULL;
        if (v == v->parent->SW) return v->parent->NW;
        if (v == v->parent->SE) return v->parent->NE;
        u = NorthNeighbor(v->parent, T);
        if (u == NULL || u->IsLeaf()) return u;
        if (v == v->parent->NW)
                return u->SW;
        else
                return u->SE;
}
```

Vyvažovanie quadtree

- Vyvážený quadtree – každé dva susedné štvorce sa líšia v rozmeroch max. o faktor 2
- Jednoduché pridávanie prázdnych podstromov
- Ak *T* má *m* vrcholov, tak jeho vyvážená verzia má O(*m*) vrcholov a dá sa vytvoriť v čase O(*m*(*d*+*1*))



Vyvažovanie quadtree 2

- CheckDivide – zistenie, či treba vrchol *v* rozdeliť, hľadajú sa susedia a či príslušný potomkovia susedov sú listy
- Divide – rozdelenie vrcholu na štyroch potomkov a presunutie bodu do jedného z nich
- CheckNeighbours – ak v susedoch vzniklo nevyváženie, tak sa susedia pridajú do zoznamu *L*

```
BalanceQuadTree(T)
{
        if (v == T->root) return NULL;
        L = T.ListLeafs();
        while (!L.IsEmpty())
        {
                v = L.PopFirst();
                if (CheckDivide(v))
                {
                        Divide(v);
                        L.add(v->NE); L.add(v->NW);
                        L.add(v->SE); L.add(v->SW);
                        CheckNeighbors(v, L);
                }
        }
}
```

Vkladanie a mazanie

- Nájdeme list, ktorý obsahuje vkladaný bod
- Ak po vložení obsahuje štvorec dva body, prerozdeľuj
- Pri mazaní je niekedy potrebné vymazať aj niektorých potomkov vnútorných vrcholov, ktoré neobsahujú žiadne body

Požiadavky

- Simple: nájdenie, či sa bod nachádza v množine bodov
- Range: vyhľadanie bodov, ktoré sa nachádzajú v danej množine
- Boolean: hľadanie prieniku objektu s definovanou množinou objektov

```
PointsInRectangle(R, v)
{
        List L;
        if (v == NULL) return L;
        if (!CheckIntersection(R, v)) retun L;
        L.add(PointsInRectangle(R, v->NW));
        L.add(PointsInRectangle(R, v->NE));
        L.add(PointsInRectangle(R, v->SW));
        L.add(PointsInRectangle(R, v->SE));
        if (Inside(v->point, R)) L.add(v->point);
        return L;
}
```

Druhy quadtrees

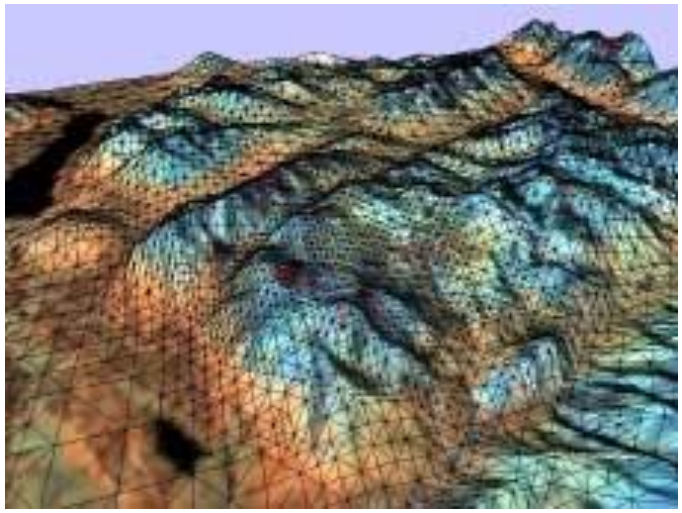
- Naplnenie geometrickými primitívami: bod, úsečka, konvexný objekt, všeobecná množina
- Delenie: v strede, v najlepšom z daných bodov, na základe distribúcie objektov,
- Rozšírenie v 3D – octree – 8 potomkov

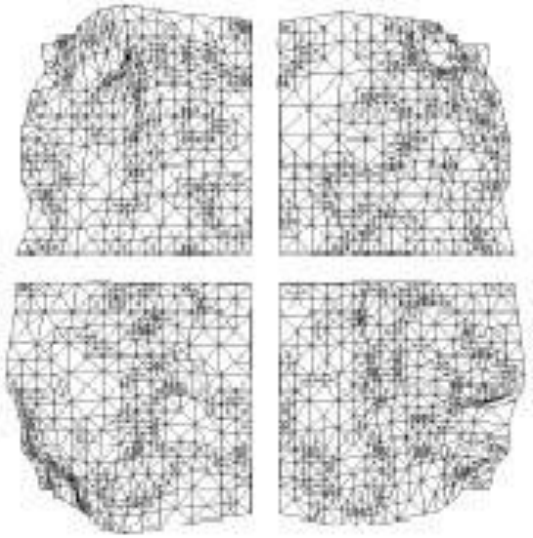
Vizualizácia terénu

- Terén – výškové pole
- Pri pohľade nad povrchom – niektoré časti sú blízko, niektoré ďalej
- Použitie LOD, každá časť terénu sa vykreslí v nejakom detaile
- Potrebná štruktúra na uskladnenie všetkých úrovní detailu pre každú časť terénu

Vizualizácia terénu 2

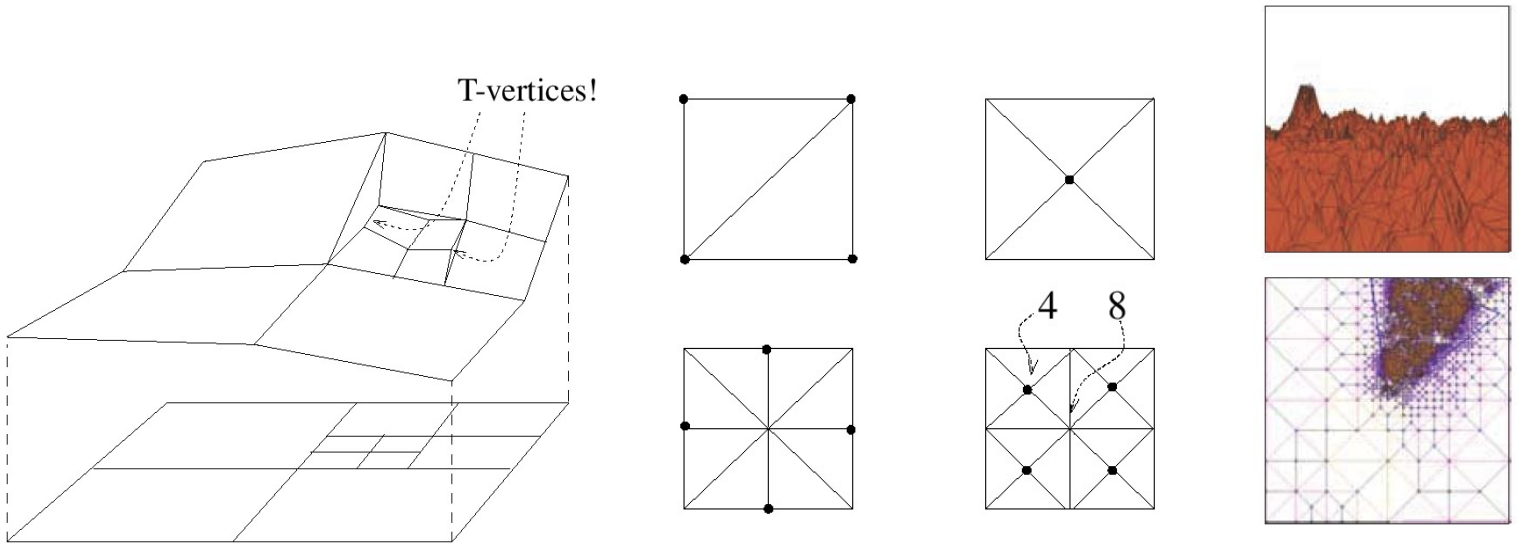
- Vytvorenie úplného quadtree nad výškovým poľom
- Pri vizualizácii sa prechádza stromom a podľa vzdialenosti sa určuje, kedy sa má prechádzanie zastaviť





Vizualizácia terénu 3

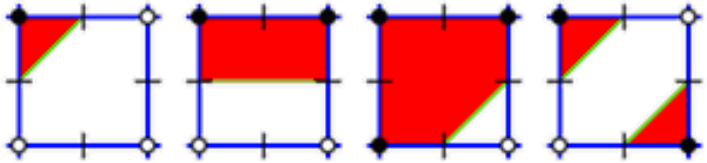
- Problémy pri prechodoch medzi úrovňami
- Riešenie pomocou triangulácie = spojenie dvoch naväzujúcich quadtrees

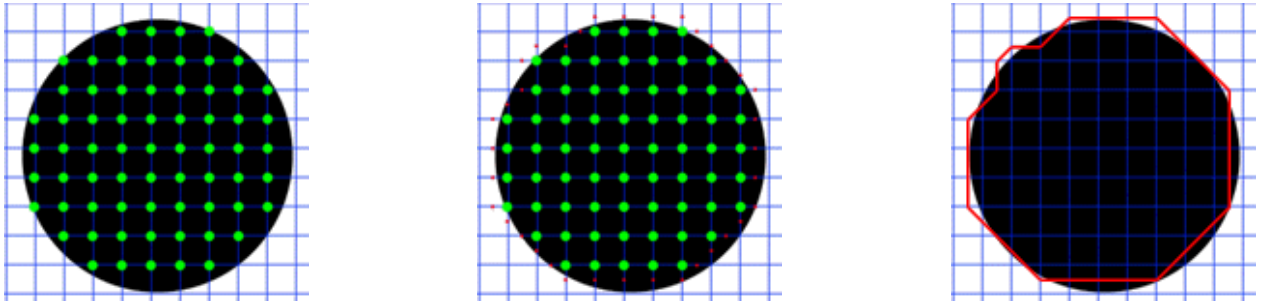


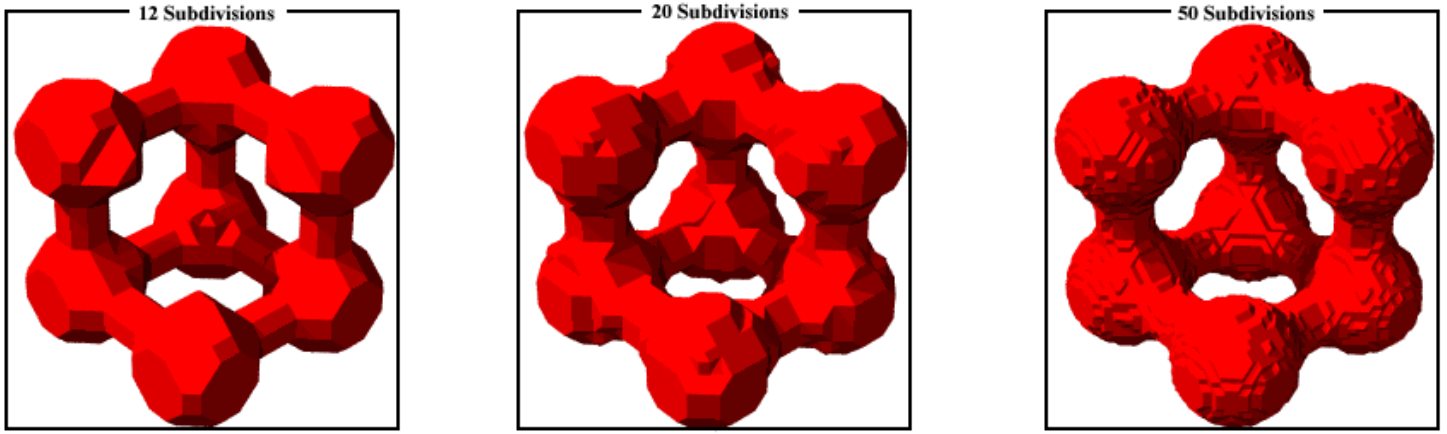
Generovanie isopovrchov

- Na uniformnej mriežke zadané intenzity, cieľ vytvoriť povrch aproximujúci určitú hladinu intenzity
- Vytvorenie kompletného quadtree, každý vrchol nesie interval intenzít v ňom obsiahnutých
- Prehľadávanie celej štruktúry alebo posun po susedoch jednotlivých buniek
- Marching cubes algoritmus

Generovanie isopovrchov 2

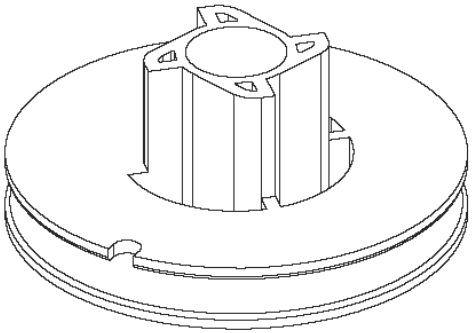


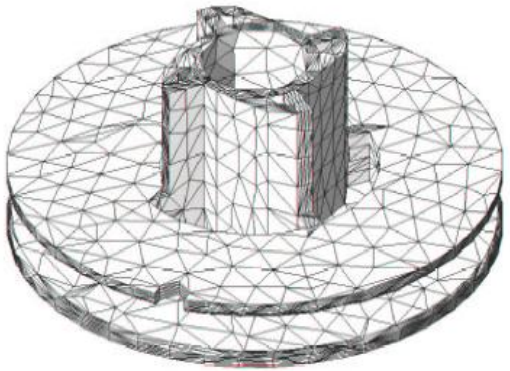


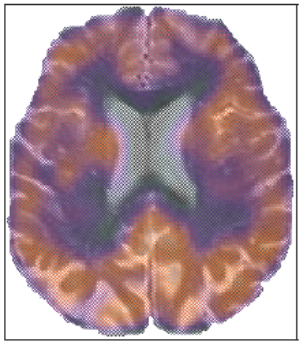


Generovanie mešov

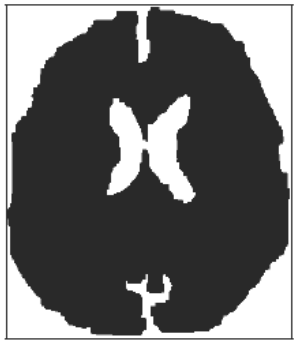
- Triangulácia množiny bodov alebo danej množiny bodov a hrán
- Rozdelenie na quadtree a riešenie triangulácie v listoch
- Vyvážený quadtree – zlepšená triangulácia
- V 3D - tetrahedralizácia



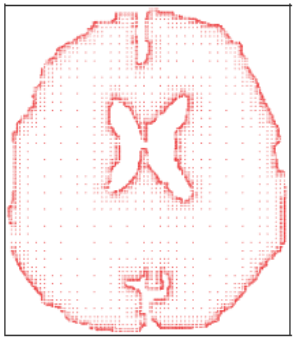




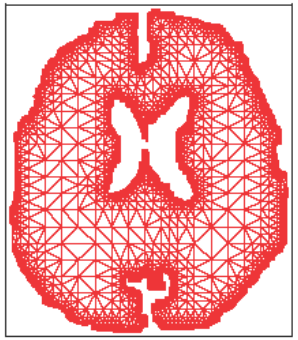
(a)



(b)



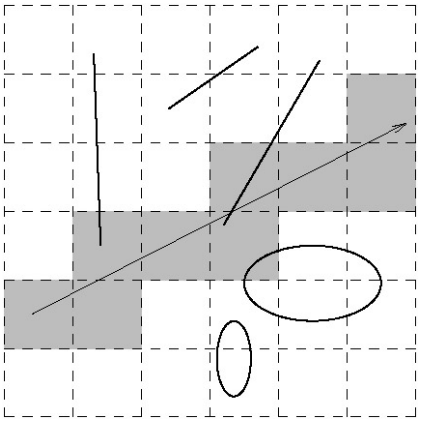
(c)

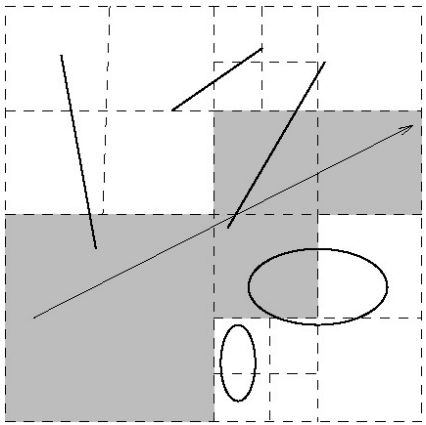


(d)

Sledovanie lúča

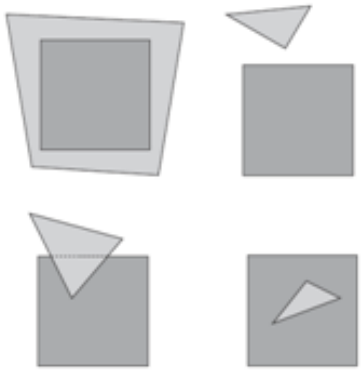
- Uloženie indexov objektov do quadtree
- Pri hľadaní prieniku lúča a objektov nájdeme nasledujúce bunky v smere lúča
- Pohľadový objem

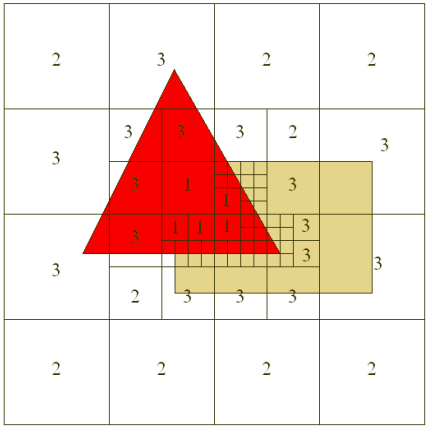




Odstraňovanie skrytých povrchov (Warnock)

- V obrazovom priestore
- Rozdeluj pixle v okne do quadtree pokým nenastane jeden z triviálnych prípadov
- V každom liste quadtree jednoducho urči farbu pixlov podľa najbližšieho polygónu





K-d stromy

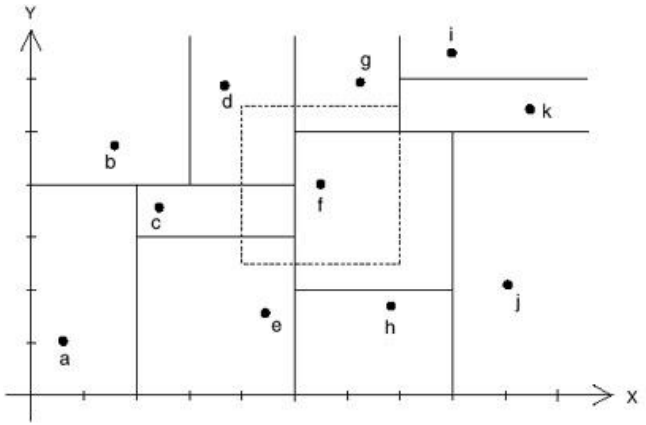
- Vstup: množina bodov *S* v R^d
- Požiadavka: *d*-rozmerný AABB *B*
- Výstup: množina bodov z *S*, ktoré patria do množiny *B*
- Rekurzívna konštrukcia:
  - Daná množina bodov *D* z R^d a deliaca súradnica *i*
  - Ak *D* je prázdne, vráť nulový vrchol
  - Vypočítaj deliacu hodnotu *s* v *i*-tej súradnici
  - Inak pre vrchol *v* vytvor dvoch potomkov a naplň potomkov podľa množín a so zvýšenou súradnicou o 1

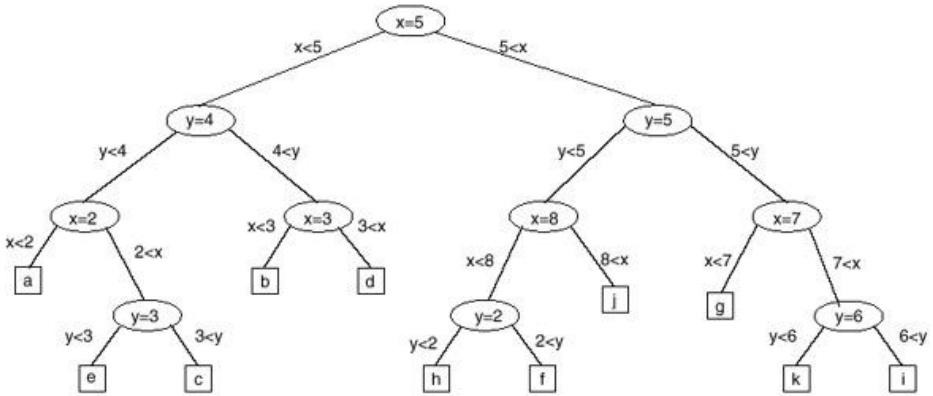
$$D_{<s_i} = \{(x_1, \ldots, x_i, \ldots, x_n) \in D | x_i < s\},$$
$$D_{>s_i} = \{(x_1, \ldots, x_i, \ldots, x_n) \in D | x_i > s\};$$

Generovanie

- Vyvážený K-d strom v rovine môže byť vytvorený v čase O(*n.*log *n*) a spotrebuje O(*n*) pamäte
- Vyvážený K-d strom v R^d môže byť vytvorený v čase O(*n.*log *n*) a spotrebuje O(*n*) pamäte

```
KdTreeConstruct(D, i)
{
        if (|D| = 0)  return NULL;
        v = new KdTreeNode;
        if (|D| = 1)
        {
                v.element = D.Element;
                v->left = NULL;
                v->right = NULL;
        }
        else
        {
                s = D.SplitValue(i);
                v->split = s;
                v->dim = i;  // axis X or Y…
                D<s = D.Left(i, s);
                D>s = D.Right(i, s);
                j = (i mod d) + 1;  // set next axis
                v->left = KdTreeConstruct(D<s, j);
                v->right = KdTreeConstruct(D>s, j);
        }
        return v;
}
```





Požiadavka

- Požiadavka na prehľadanie K-d stromu v R^d má časovú náročnosť $O(n^{(1-1/d)} + k)$, kde k je počet nájdených bodov

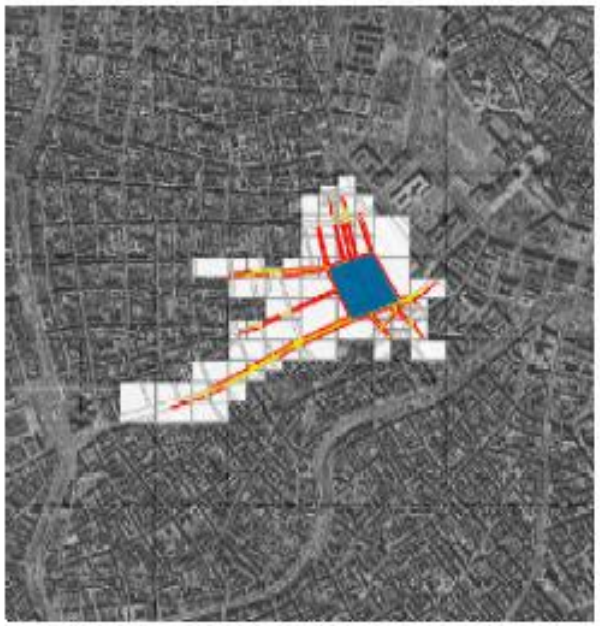
```
KdTreeQuery(v, Q, B)
{
        List L;
        if (v->IsLeaf() &&  (v->element in B) L.add(v->element);
        else
        {
                v_l = v->left;
                v_r = v->right;
                Q_l = Q.LeftPart(v->split);
                Q_r = Q.RightPart(v->split);
                if (Q_l in B) L.add(v->left->Report);  // add all points from Q_l
                else if (Q_l ∩ B != 0) L.add(KdTreeQuery(v->left,Q_l ,B));
                if (Q_r in B) L.add(v->right->Report); // add all points from Q_r
                else if (Q_r ∩ B ! = 0) L.add(KdTreeQuery(v->right, Q_r, B));
        }
        return L;
}
```

K-d stromy 2

- Rozdelenie priestoru
- Malá pamäťová náročnosť
- V najhoršom prípade vysoká časová náročnosť (pri zlom rozdeľovaní), očakávaná náročnosť je $O(\lg N + k)$
- Rôzne spôsoby rozdeľovania
- Jednoduchý insert, zložitý delete
- Široké spektrum použitia

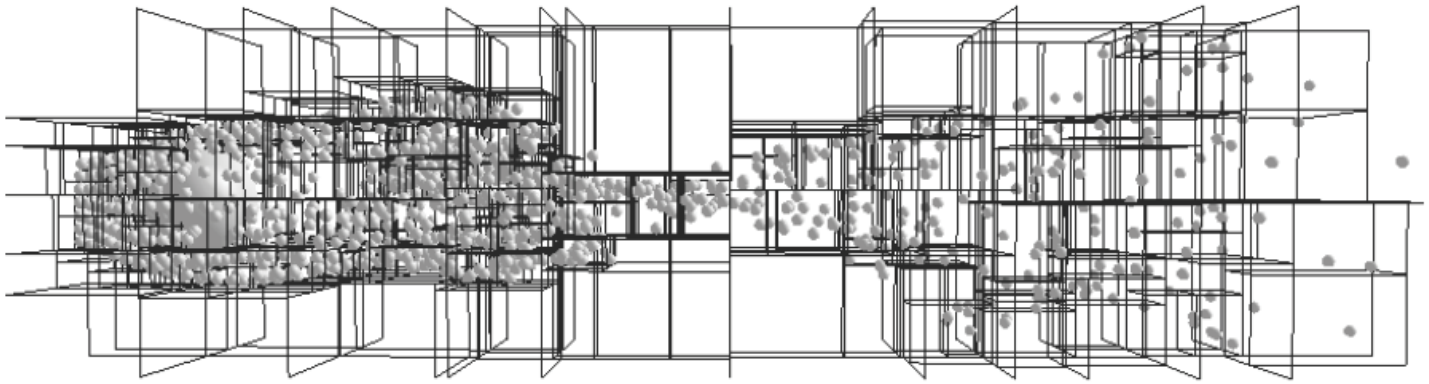
K-d stromy 3

- Použitie pre podobné problémy ako v prípade quadtrees, octrees
- Lepšia adaptabilita



Raytracing

- K-d strom je najlepšie delenie priestoru pre raytracing (minimalizuje počet rátaní priesečníka lúč-objekt) http://www.cgg.cvut.cz/u/havran/phdthesis.html
- Možnosť adaptívneho delenia podľa povrchu vrcholov stromu



koniec (-:

florek@sccg.sk