Diskrétne Geometrické Štruktúry

2. Quadtree, k-d stromy

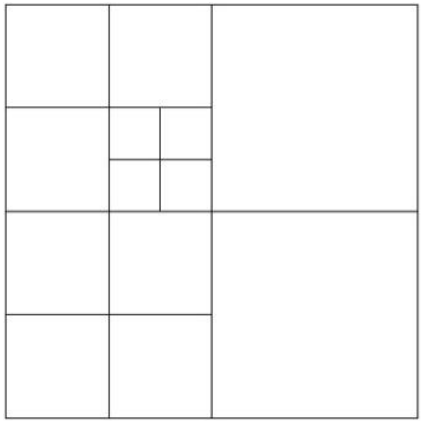
Martin Samuelčík
samuelcik@sccg.sk, www.sccg.sk/~samuelcik, I4

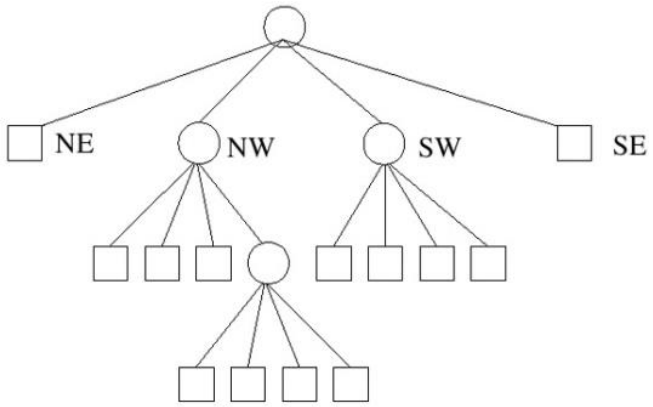
Oknová a bodová požiadavka

- Pre dané body, nájdi z nich také, ktoré patria danému d-rozmernému intervalu
- Riešenie cez viacrozmerné range stromy
  - Vyššia pamäťová náročnosť
  - Všeobecné pre rôzne dimenzie
  - Adaptívne delenie na základe daných bodov
- Nové riešenie – delenie nadrovinami v danej dimenzii – pomalšie, nižšia pamäťová náročnosť

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 rovnaké časti
- 2D





Vytvorenie quadtree

- S – množina bodov v 2D
- Na začiatku vytvoríme ohraničujúci štvorec S
- Rekurzívne delenie

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

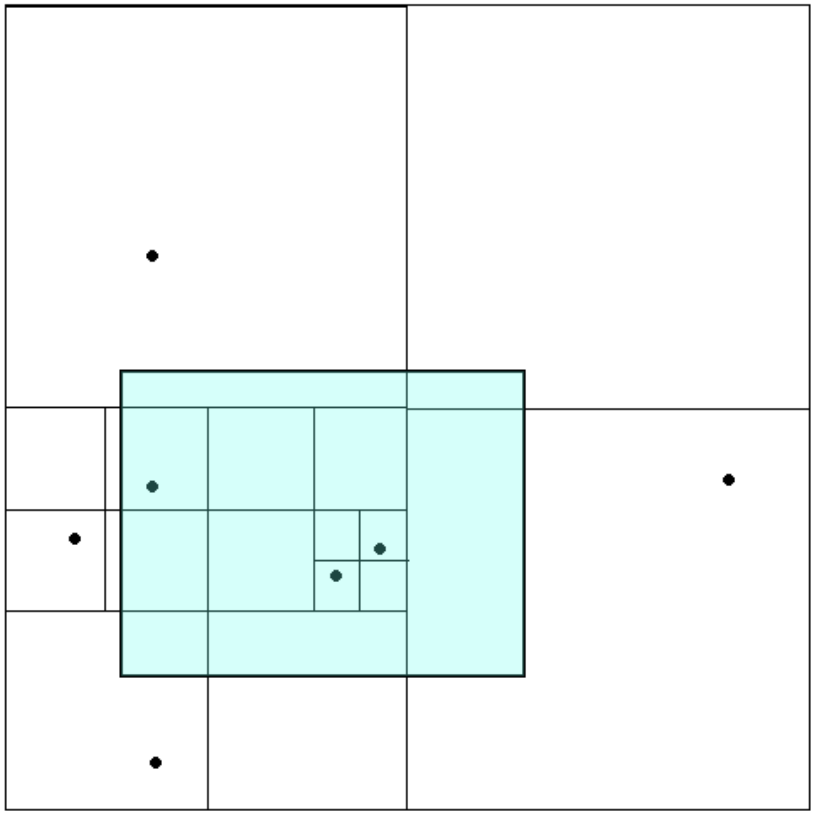
```
struct QuadTree
{
   QuadTreeNode* root;
}
```

```
QuadTreeConstruct(S)
{
   (left, right, bottom, top) = BoundSquare(S);
   QuadTree* tree = new QuadTree;
   tree->root = QuadTreeNodeConstruct(S, left, right, bottom, top);
   return tree;
}
```

```
QuadTreeNodeConstruct(P, left, right, bottom, top)
{
   v = new QuadTreeNode;
   v->left = left; v->right = right; v->bottom = bottom; v->top = top;
   v->NE = v->NW = v->SW = v->SE = v->parent = v->point = NULL;
   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;
}
```

Prehľadanie quadtree

- Nájdenie bodov ležiacich v obdĺžniku *B=[left,right,bottom,top]*



```
QuadTreeQuery(tree, B)
{
   return QuadTreeNodeQuery(tree->root, B)
}
```

```
QuadTreeNodeQuery(node, B)
{
   List result;
   if (node == NULL)
      return result;
   if (B->left > node->right || B->right < node->left ||
       B->bottom > node->top || B->top < node->bottom)
   {
      return result;
   }
   if (node->point)
      result.Add(point);

   result.Add(QuadTreeNodeQuery(v->NE, B));
   result.Add(QuadTreeNodeQuery(v->NW, B);
   result.Add(QuadTreeNodeQuery(v->SW, B);
   result.Add(QuadTreeNodeQuery(v->SE, B);
   return result;
}
```

Vlastnosti quadtree

- Hĺbka quadtree je najviac *log(s/c) + 3/2*, kde *c* je najmenšia vzdialenosť bodov z *S* a *s* je dĺžka strany počiatočného štvorca
- Quadtree hĺbky *d* s */S/=n* má O(n.(d+1)) vrcholov a dá sa vytvoriť v čase O(n.(d+1))
- Konštrukcia: $O(n^2)$
- Pamäť: $O(n^2)$
- Vyhladanie: O(n)

Vyhľadanie suseda

- Pre daný vrchol a korešpondujúci štvorec, nájdi štvorec susediaci v danom smere na rovnakej a nižšej úrovni
- Časová zložitosť O(d), d – výška stromu

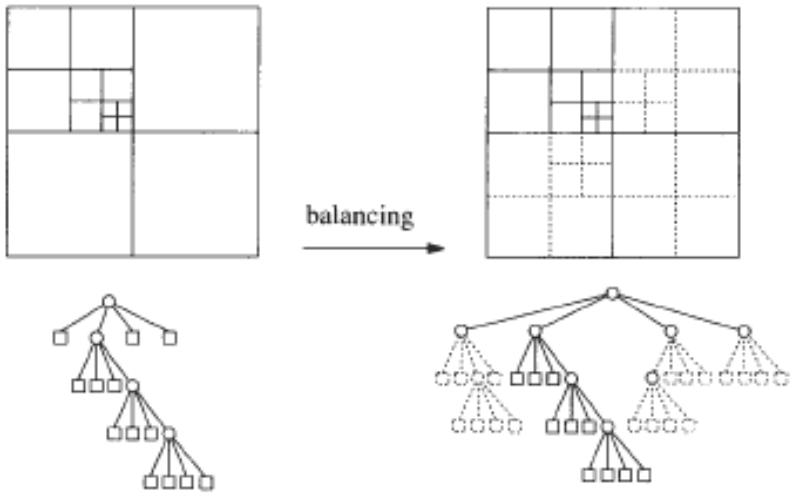
```
NorthNeighbor(v, T)
{
        if (v == T->root) return NULL;
        if (v == v->parent->SW) return v->parent->NW;
        if (v == v->parent->SE) return v->parent->NE;
        u = NorthNeighbor(v->parent, T);
        if (u == NULL || u->IsLeaf()) return u;
        if (v == v->parent->NW)
      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

```
SouthChilds(v)
{
        List result;
        if (v == NULL)
                return result;
        if (v->IsLeaf())
                result.Add(v);
        result.Add(SouthChilds(v->SE));
        result.Add(SouthChilds(v->SW));
        return result;
}
```

```
NorthNeighbors(v, T)
{
        List result;
        North = NorthNeighbor(v, T);
        if (North == NULL)
                return result;
        return SouthChilds(North);
}
```

Vyvažovanie quadtree

- Vyvážený quadtree – každé dva susedné štvorce sa líšia v rozmeroch max. o faktor 1
- 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))



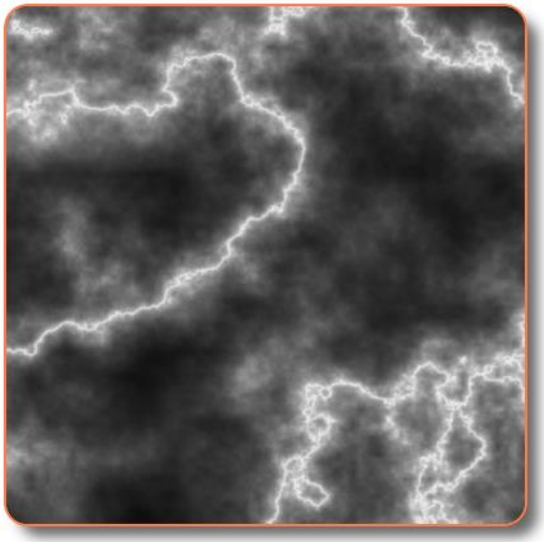
Vyvažovanie quadtree

- 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);
                }
        }
}
```

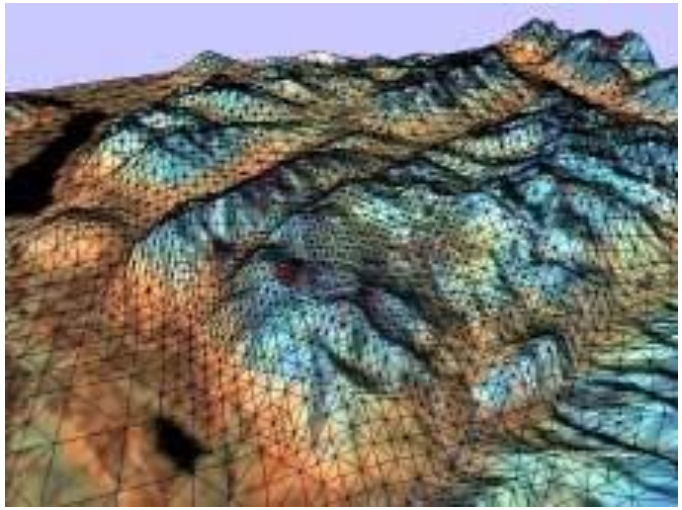
Vizualizácia terénu

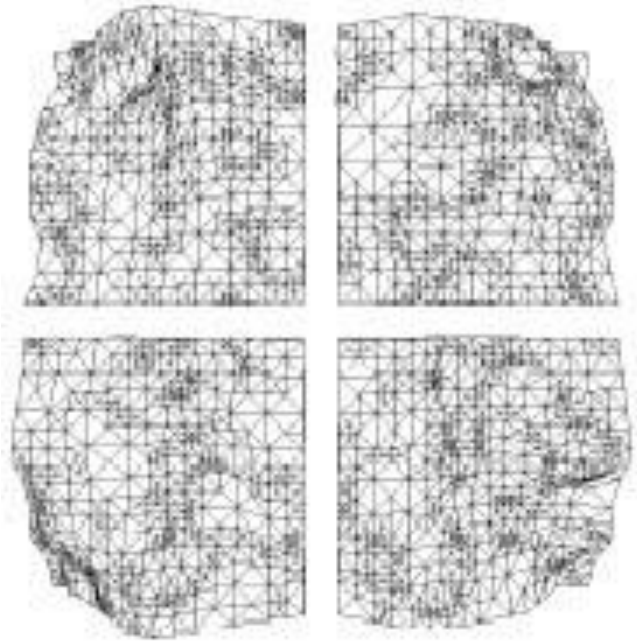
- Pri pohľade nad povrchom – niektoré časti sú blízko, niektoré ďalej - použitie LOD (Level Of Detail), každá časť terénu sa vykreslí v nejakom detaile
- Potrebná štruktúra na uskladnenie všetkých úrovní detailu pre každú časť terénu
- Terén
  - výškové pole



Vizualizácia terénu

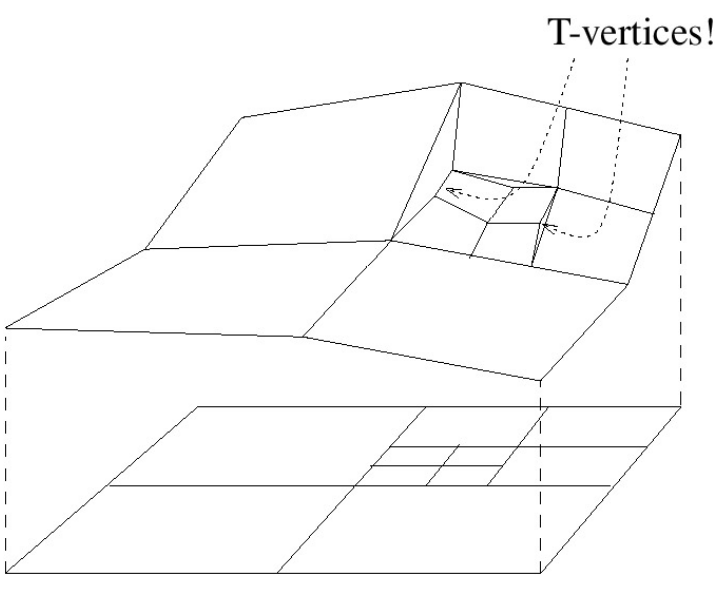
- Vytvorenie úplného quadtree nad výškovým poľom
- Pri vizualizácii sa prechádza stromom a podľa vzdialenosti sa určuje, kedy sa má prechádzanie zastaviť

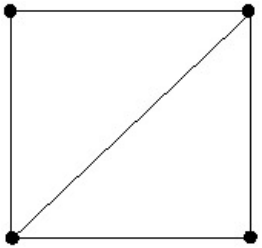


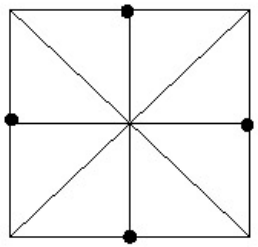


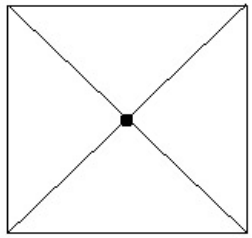
Vizualizácia terénu

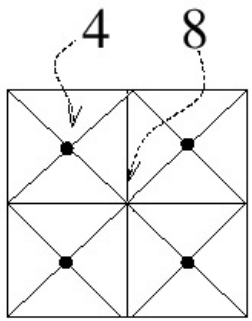
- Problémy pri prechodoch medzi úrovňami
- Riešenie pomocou triangulácie = spojenie dvoch naväzujúcich quadtrees

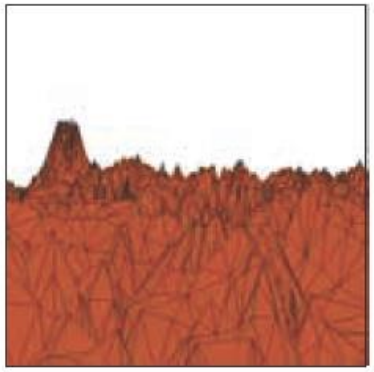


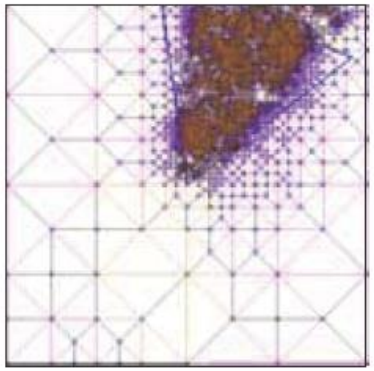




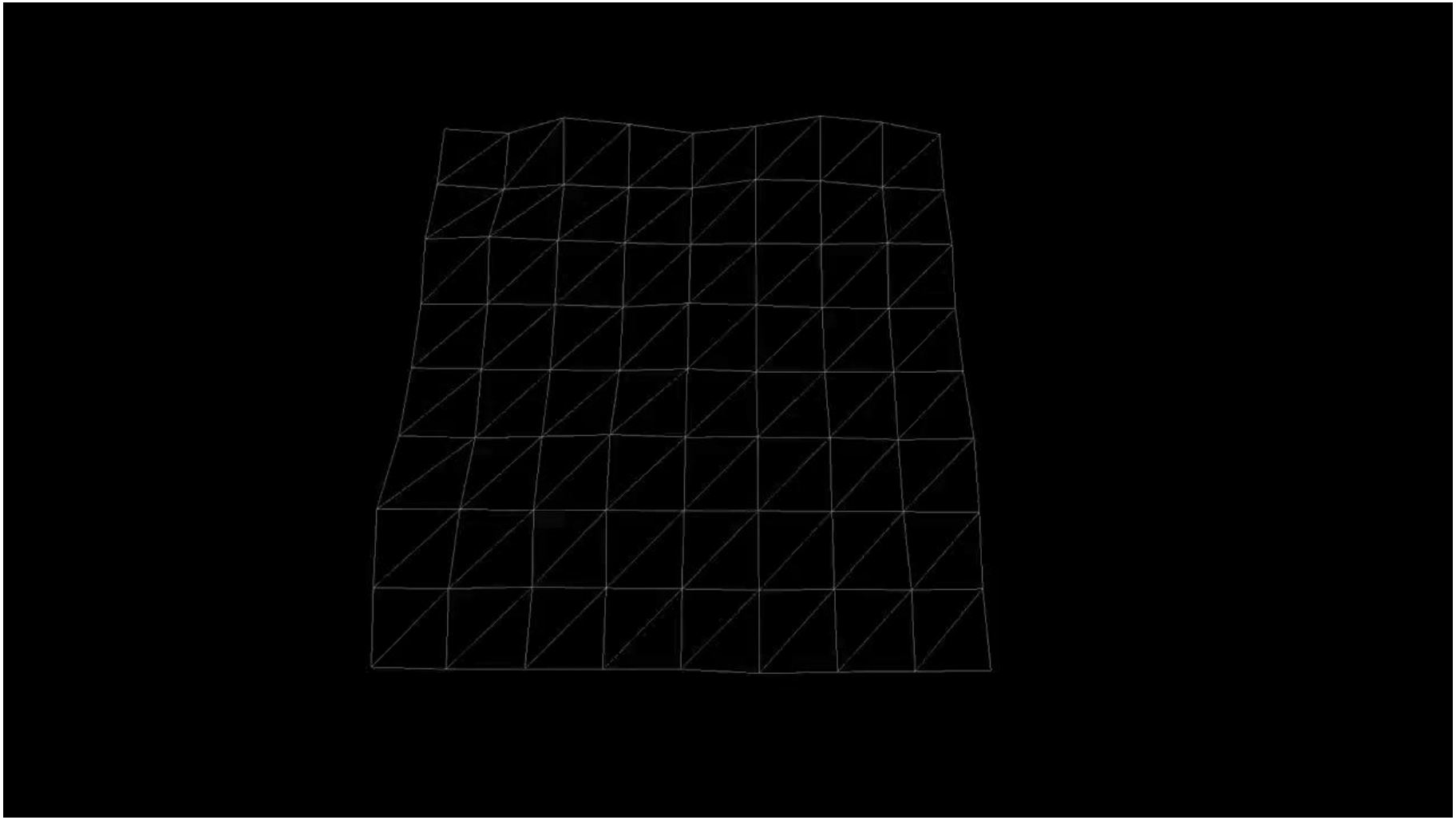








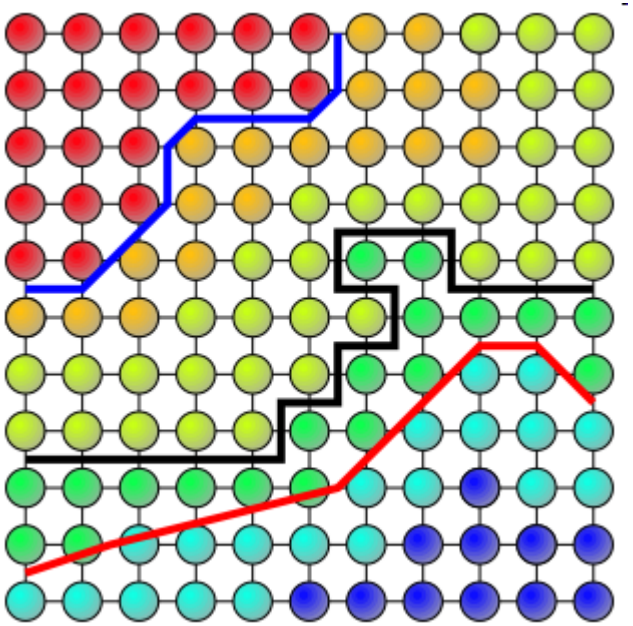
Vizualizácia terénu

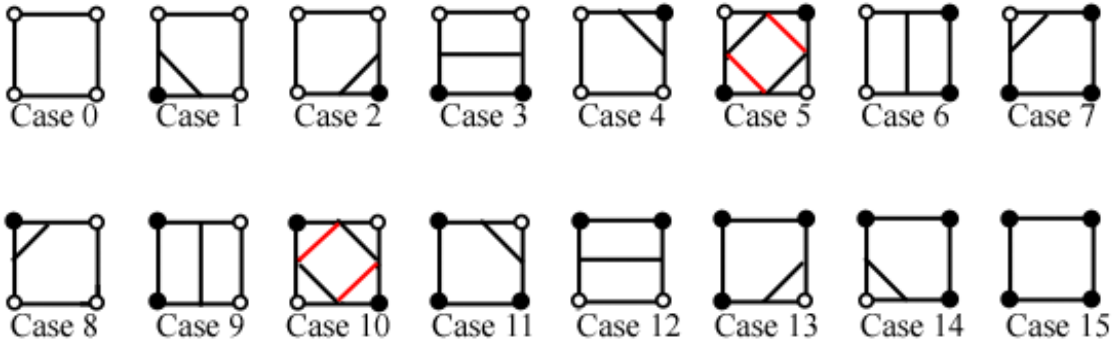


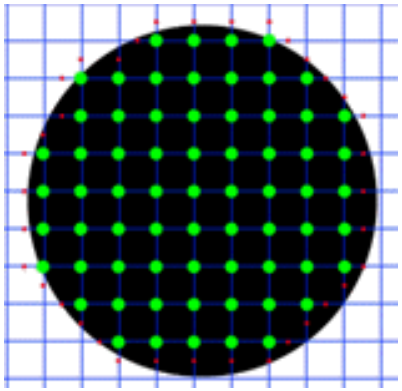
Generovanie isopovrchov

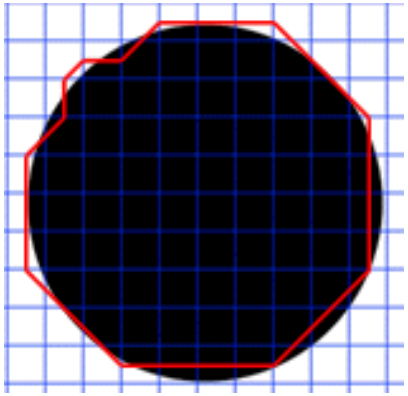
- Na uniformnej mriežke zadané intenzity, cieľ vytvoriť povrch aproximujúci určitú hladinu intenzity
- Vytvorenie kompletného quadtree, každý vrchol nesie interval intenzít v ňom obsiahnutých, pre homogénne oblasti netreba prerozdelovať
- Prehľadávanie celej štruktúry alebo posun po susedoch jednotlivých buniek – „Marching Cubes" algoritmus

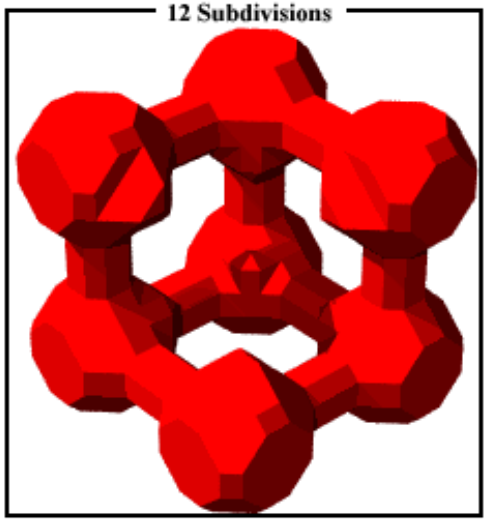
Generovanie isopovrchov

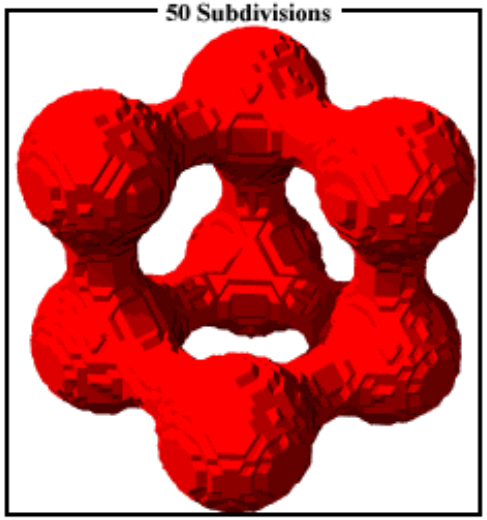






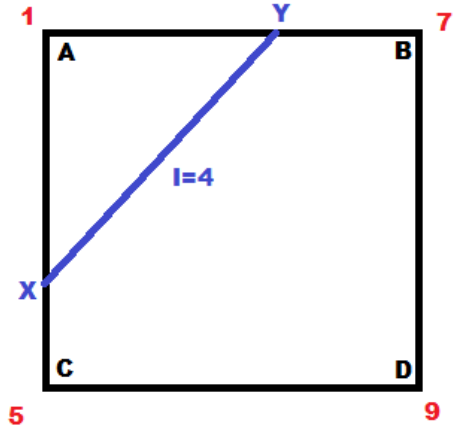






Bilineárna interpolácia

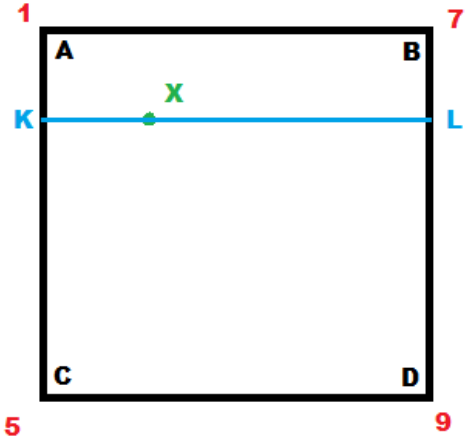
- Výpočet vrcholov úsečiek pre štvorec



$$X = \frac{|5-4|}{|5-1|}A + \frac{|4-1|}{|5-1|}C = \frac{1}{4}A + \frac{3}{4}C$$

$$Y = \frac{|7-4|}{|7-1|}A + \frac{|4-1|}{|7-1|}B = \frac{1}{2}A + \frac{1}{2}B$$

- Interpolácia intenzity vnútri štvorca



$$I_K = \frac{|Xy - Ay|}{|Cy - Ay|}5 + \frac{|Cy - Xy|}{|Cy - Ay|}1 \qquad I_L = \frac{|Xy - By|}{|Dy - By|}9 + \frac{|Dy - Xy|}{|Dy - By|}7$$

$$I_X = \frac{|Xx - Ax|}{|Bx - Ax|}I_L + \frac{|Bx - Xx|}{|Bx - Ax|}I_K$$

Quadtree pre Marching Cubes

- Dané dvojrozmerné pole intenzít
  - $P[i,j]; i = 0,\ldots,2^n ; j = 0,\ldots,2^m$

```
struct MCQuadTree
{
   MCQuadTreeNode* root;
}
```

```
struct MCQuadTreeNode
{
   float MinIntensity;
   float MaxIntensity;
   float Corner1, Corner2;
   float Corner3, Corner4;
   QuadTreeNode * parent;
   QuadTreeNode * NE;
   QuadTreeNode * NW;
   QuadTreeNode * SW;
   QuadTreeNode * SE;
}
```

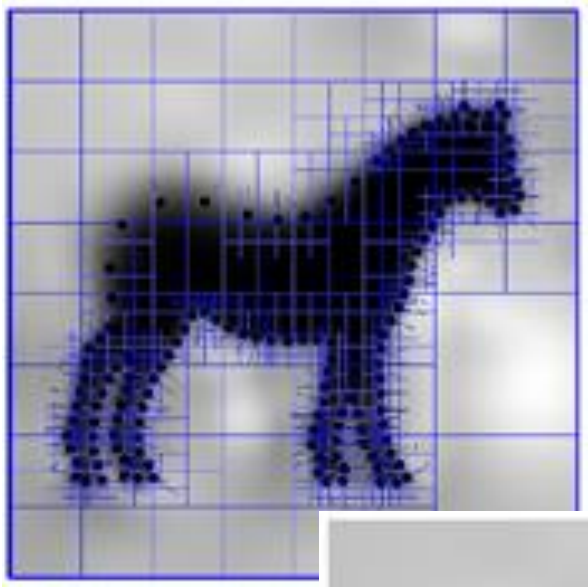
```
MCQuadTreeConstruct(P, n, m)
{
   MCQuadTree* tree = new MCQuadTree;
   tree->root = MCQuadTreeNodeConstruct(
                    P, 0, 2^n, 0, 2^m);
   return tree;
}
```

```
MCQuadTreeNodeConstruct(P, MinXIndex, MaxXIndex, MinYIndex, MaxYIndex)
{
   v = new MCQuadTreeNode;
   v->Corner1 = P[MinXIndex, MinYIndex]; v->Corner2 = P[MaxXIndex, MinYIndex];
   v->Corner3 = P[MaxXIndex, MaxYIndex]; v->Corner4 = P[MinXIndex, MaxYIndex];
   v->NE = v->NW = v->SW = v->SE = v->parent = NULL;
   (Min, Max) = GetMinMaxIntensities(P, MinXIndex, MaxXIndex,
                                                MinYIndex, MaxYIndex);
   v->MinIntensity = Min;
   v->MaxIntensity = Max;
   if (Min == Max)
      return v;
   MidXIndex = (MinXIndex + MaxXIndex) / 2;
   MidYIndex = (MinYIndex + MaxYIndex) / 2;
   v->NE = MCQuadTreeNodeConstruct(P, MidXIndex, MaxXIndex, MidYIndex, MaxYIndex);
   v->NW = MCQuadTreeNodeConstruct(P, MinXIndex, MidXIndex, MidYIndex, MaxYIndex);
   v->SW = MCQuadTreeNodeConstruct(P, MinXIndex, MidXIndex, MinYIndex, MidYIndex);
   v->SE = MCQuadTreeNodeConstruct(P, MidXIndex, MaxXIndex, MinYIndex, MidYIndex);
   v->NE->parent = v; v->NW->parent = v;
   v->SW->parent = v; v->SE->parent = v;
   return v;
}
```

Marching Cubes

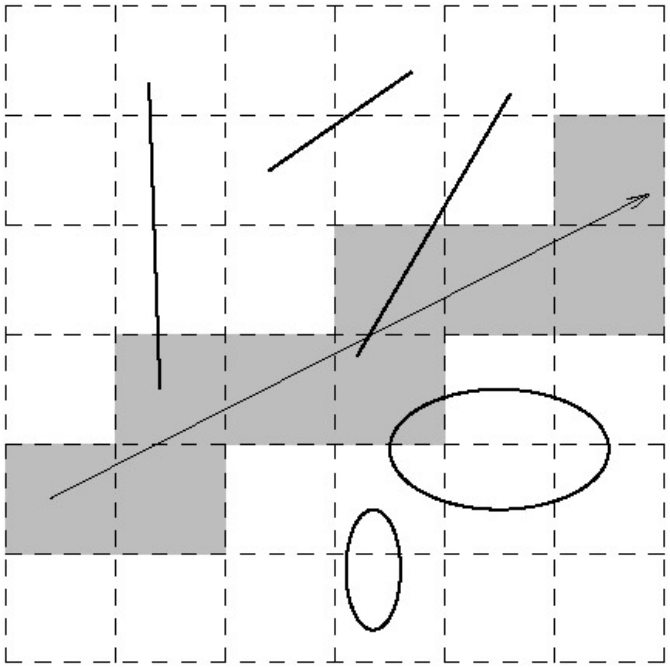
```
MarchCubes(T, intensity, left, right, bottom, top)
{
   MarchCubesNode(T->root, intensity, left, right, bottom, top);
}
```

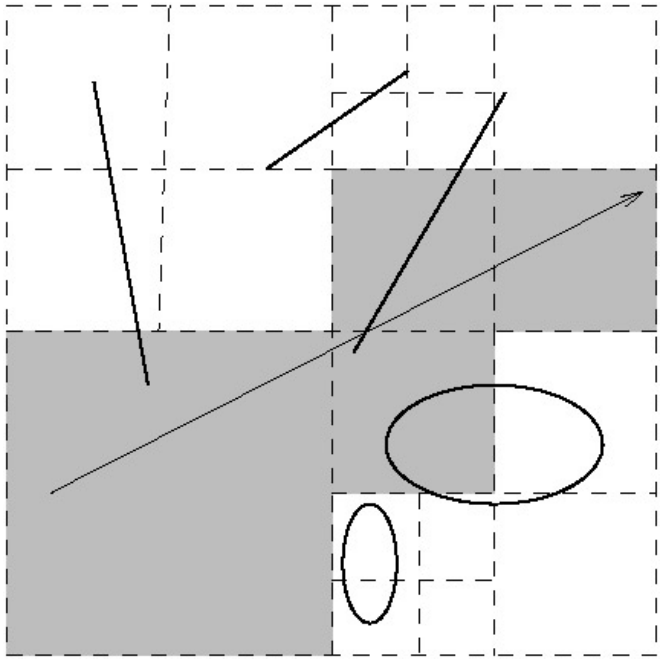
```
MarchCubesNode(v, intensity, left, right, bottom, top)
{
   if (intensity < MinIntensity || intensity > MaxIntensity)
      return;
   if (MinIntensity == MaxIntensity)
      return;
   if (v->IsLeaf())
   {
      CreateLinesInRextangle(left, right, bottom, top,
         v->Corner1, v->Corner2, v->Corner3, v->Corner4);
      return;
   }
   float midx = (left+right) / 2;
   float midy = (bottom+top) / 2;
   MarchCubesNode(v->SW, intensity, left, midx, bottom, midy);
   MarchCubesNode(v->SE, intensity, midx, right, bottom, midy);
   MarchCubesNode(v->NE, intensity, midx, right, midy, top);
   MarchCubesNode(v->NE, intensity, left, midx, midy, top);
}
```



Sledovanie lúča

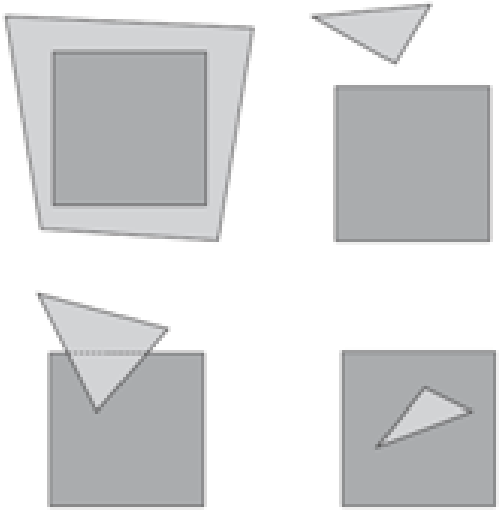
- Uloženie indexov objektov do quadtree
- Pri hľadaní prieniku lúča a objektov najprv nájdeme nasledujúcu bunku v smere lúča

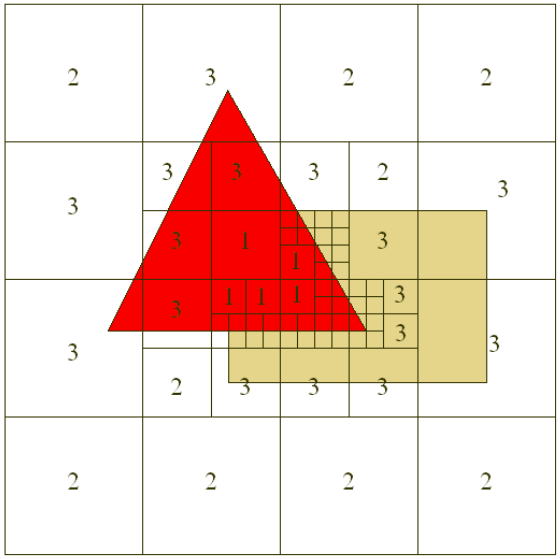




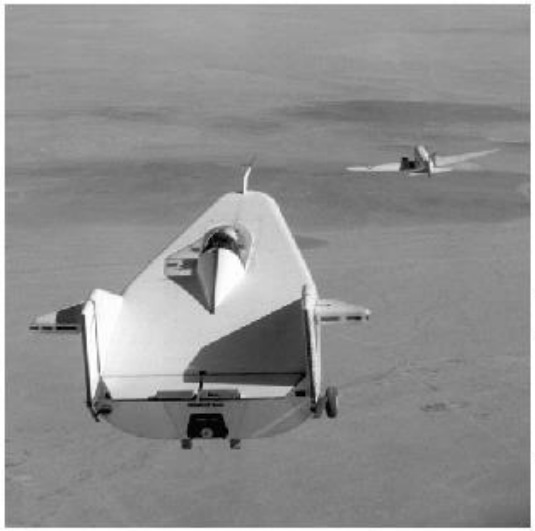
Skryté povrchy (Warnock)

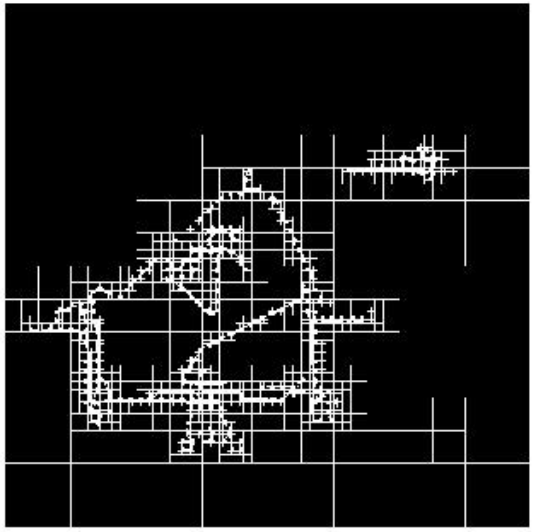
- V obrazovom priestore
- Rozdeluj pixle v okne do quadtree pokým nenastane jeden z triviálnych prípadov
- V každom liste quadtree jednoducho urči farbu pixlov podľa najbližšieho polygónu

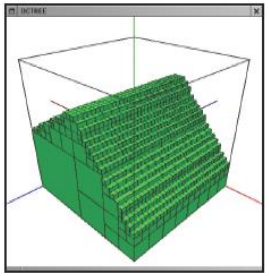


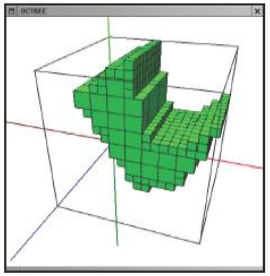


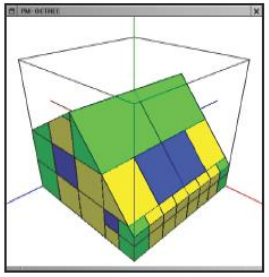
Reprezentácie

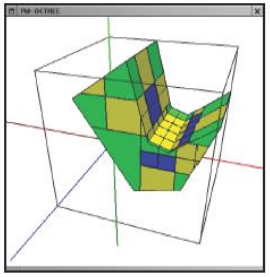


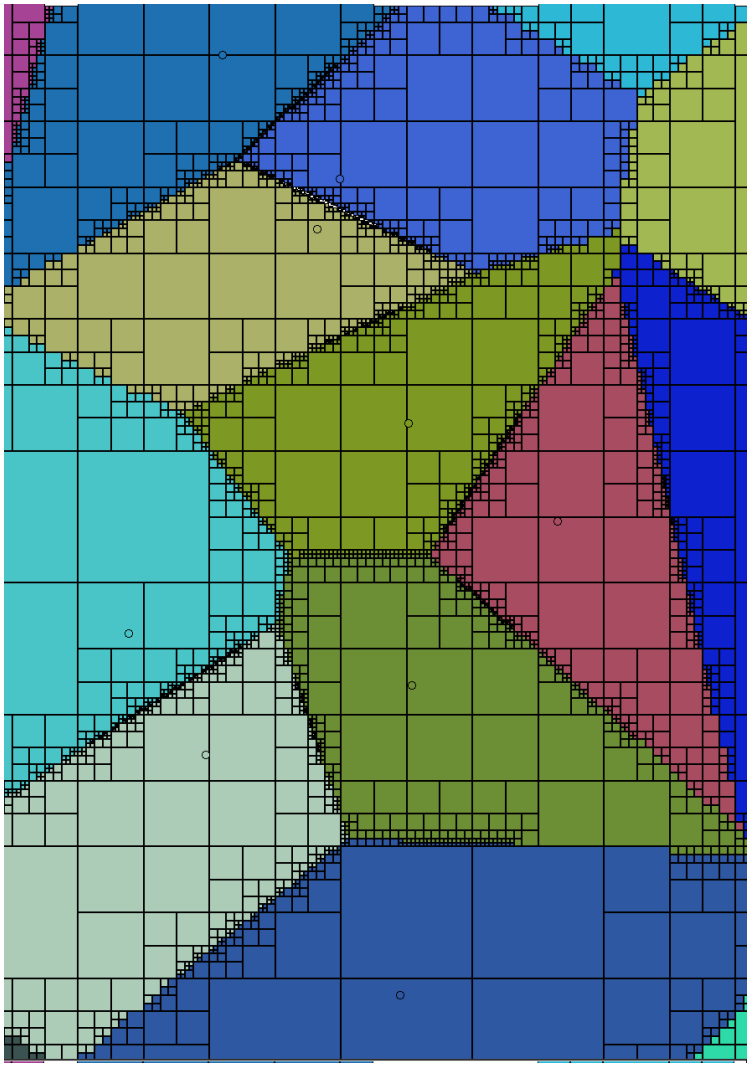






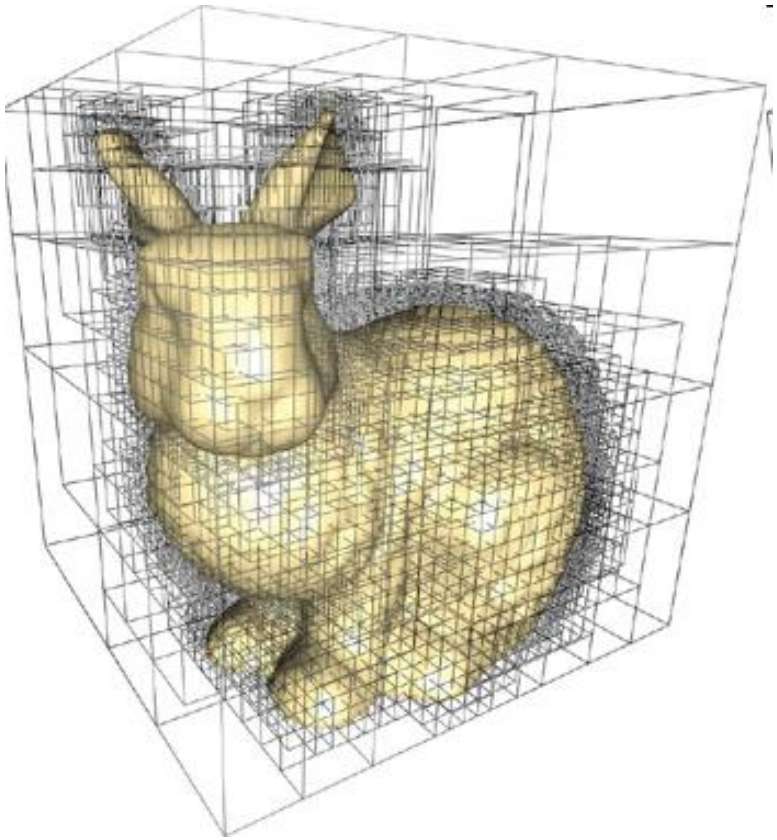


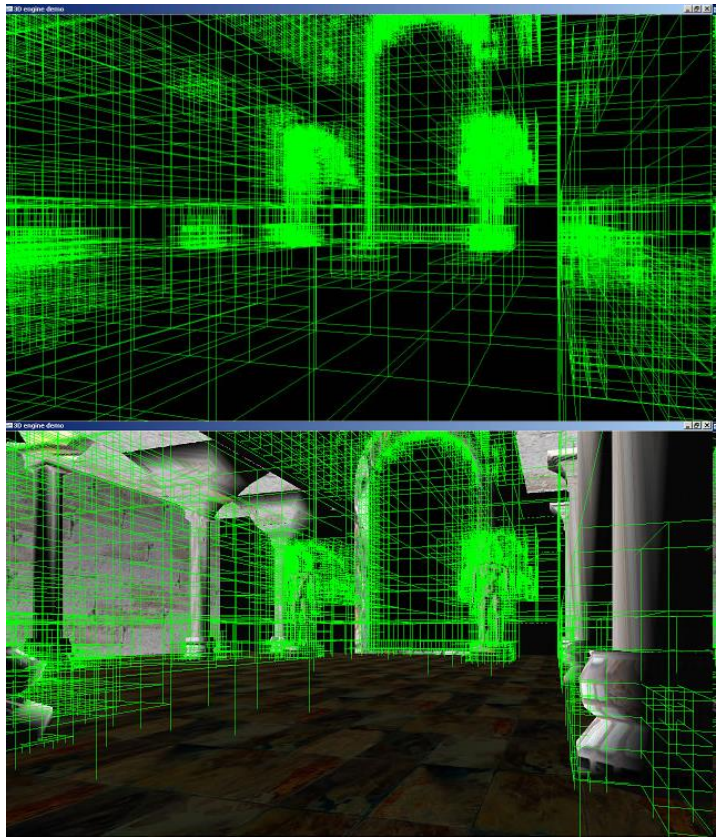




Octree

- Rozšírenie quadtree do 3D, riešenie rovnakých či podobných problémov





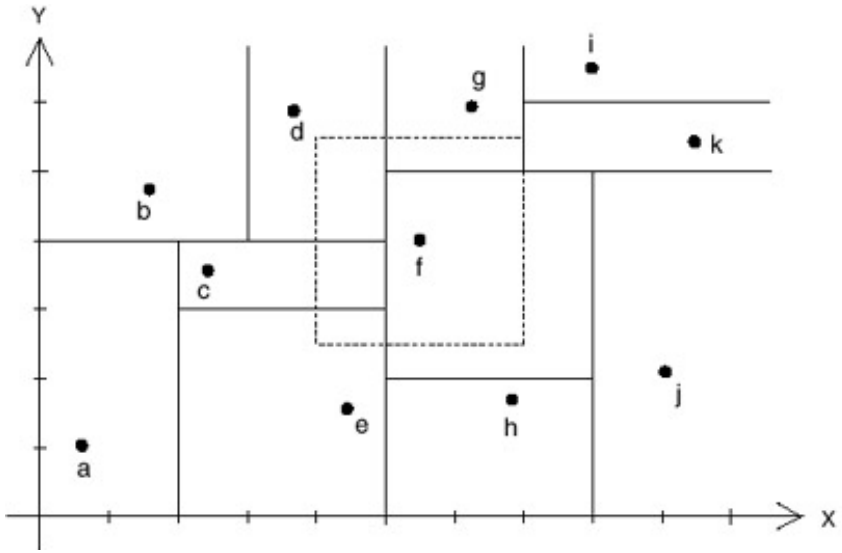
K-d strom

- Vstup: množina bodov S v R^d
- Požiadavka: d-rozmerný interval B
- Výstup: množina bodov z S, ktoré patria do množiny B
- Rekurzívna konštrukcia:

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

  - Daná množina bodov D z R^d a deliaca súradnica i
  - Ak D je prázdne, vráť nulový vrchol
  - Ak D obsahuje 1 bod, aktuálny vrchol bude list
  - Inak vypočítaj deliacu hodnotu s v i-tej súradnici a podľa tejto hodnoty rozdeľ D na dve množiny $D_{<s_i}$, $D_{>s_i}$ a pre tieto množiny rekurzívne vytvor dvoch potomkov a naplň potomkov podľa množín a so zvýšenou súradnicou i o 1

Generovanie k-d stromu

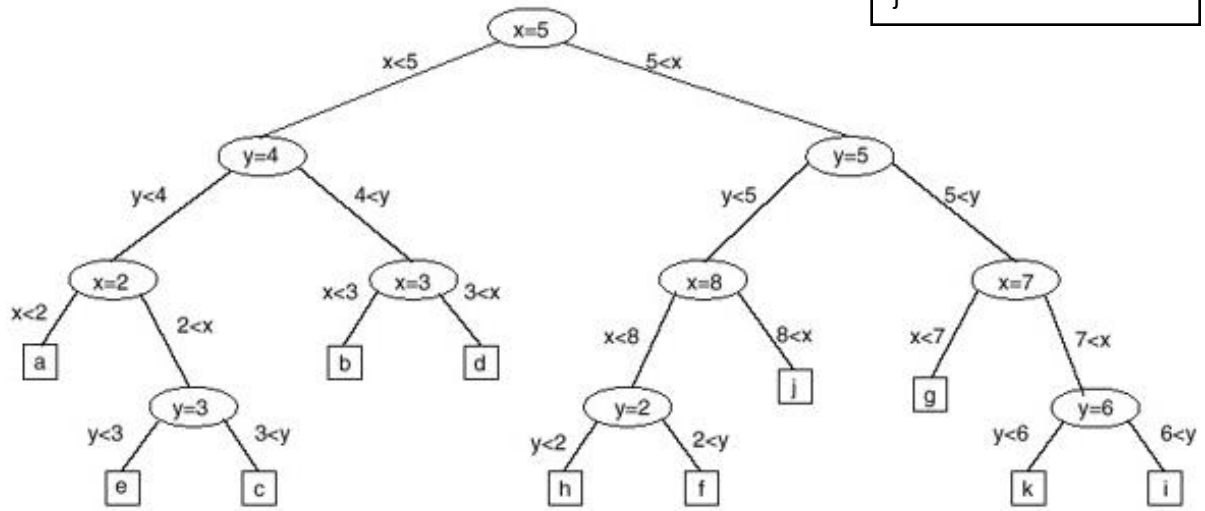



```
struct KdTreeNode
{
   float  split;
   int dim;
   Point* point;
   KdTreeNode * left;
   KdTreeNode * right;
   KdTreeNode * parent;
}
```

```
struct KdTree
{
   KdTreeNode * root;
}
```

```
KdTreeConstruct(S, d)
{
   T = new KdTree;
   T->root = KdTreeNodeConstruct(S, 0, d);
   return T;
}
```

```
KdTreeNodeConstruct(D, dim, d)
{
   if (|D| = 0)  return NULL;
   v = new KdTreeNode;
   v->dim = dim;
   if (|D| = 1)
   {
      v->point = D.Element;
      v->left = NULL;
      v->right = NULL;
      return v;
   }
   v->point = NULL;
   v->split = D.ComputeSplitValue(dim);
   D<s = D.Left(dim, v->split);
   D>s = D.Right(dim, v->split);
   j = (dim + 1) mod d;
   v->left = KdTreeNodeConstruct(D<s, j);
   v->right = KdTreeNodeConstruct(D>s, j);
   return v;
}
```



Prehľadávanie

- Pri hľadaní bodov v danom d-rozmernom intervale si pamätáme d-rozmerné intervaly prislúchajúce každému vrcholu stromu (Q)

```
KdTreeQuery(T, B)
{
   Q = WholeSpace();
   return KdTreeNodeQuery(T->root, Q, B);
}
```

```
Report(v)
{
   List L;
   if (v->IsLeaf() &&  (v->point))
   {
      L.add(v->point);
      return L;
   }
   L.add(Report(v->left));
   L.add(Report(v->right));
   return L;
}
```

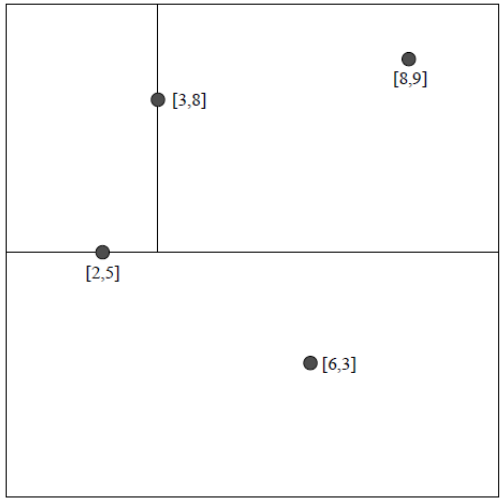
```
KdTreeNodeQuery(v, Q, B)
{
   List L;
   if (v->IsLeaf() &&  (v->point in B))
   {
      L.add(v->point);
      return L;
   }
   v_l := v->left;
   v_r := v->right;
   Q_l := Q.LeftPart(v->dim, v->split);
   Q_r := Q.RightPart(v->dim, v->split);
   if (Q_l in B)
      L.add(Report(v->left));
  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

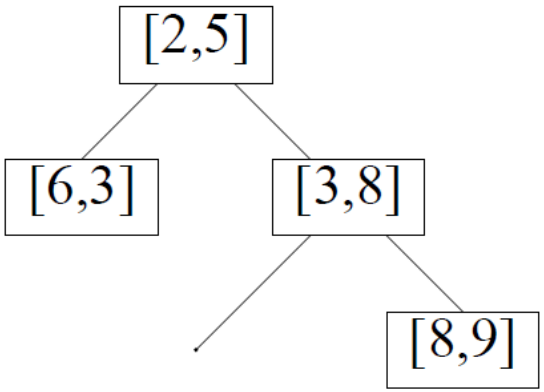
Vlastnosti k-d stromov

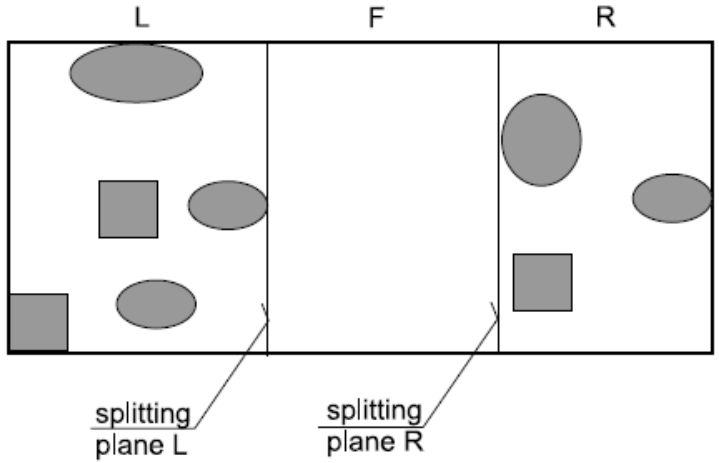
- Ak pri delení sú $D_{<s_j}$, $D_{>s_j}$ približne rovnaké (použije sa medián), potom je strom vyvážený
- Vyvážený k-d strom v R^d môže byť vytvorený v čase O(n.log(n)) a spotrebuje O(n) pamäte
- Požiadavka na prehľadanie k-d stromu v R^d má časovú náročnosť $O(n^{(1-1/d)} + k)$, kde k je počet nájdených bodov
- V najhoršom prípade vysoká časová náročnosť (pri zlom rozdeľovaní), očakávaná náročnosť je O(log(n) + k)
- Vloženie bodu: O(log(n))
- Odstránenie bodu: O(log(n))

K-d stromy

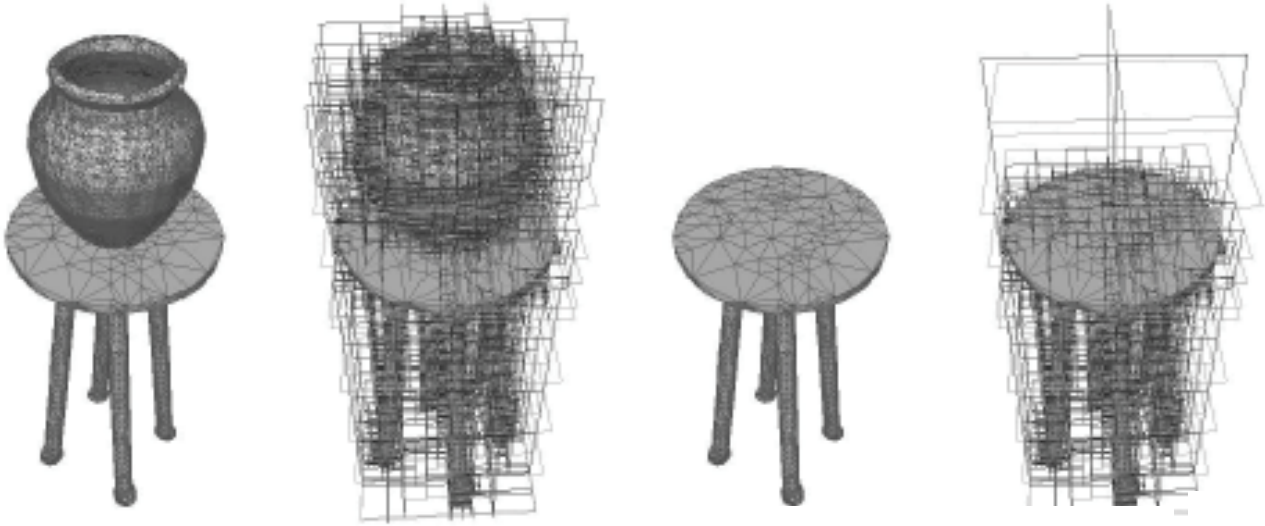
- Variácie stromov: body nielen v listoch, neperiodická zmena deliacej nadroviny, rôzne spôsoby delenia a ukončenia, dve deliace nadroviny

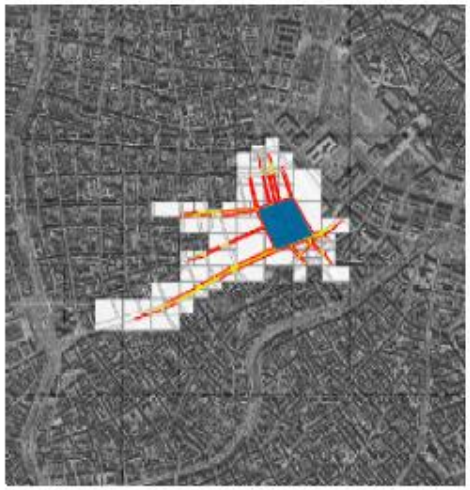




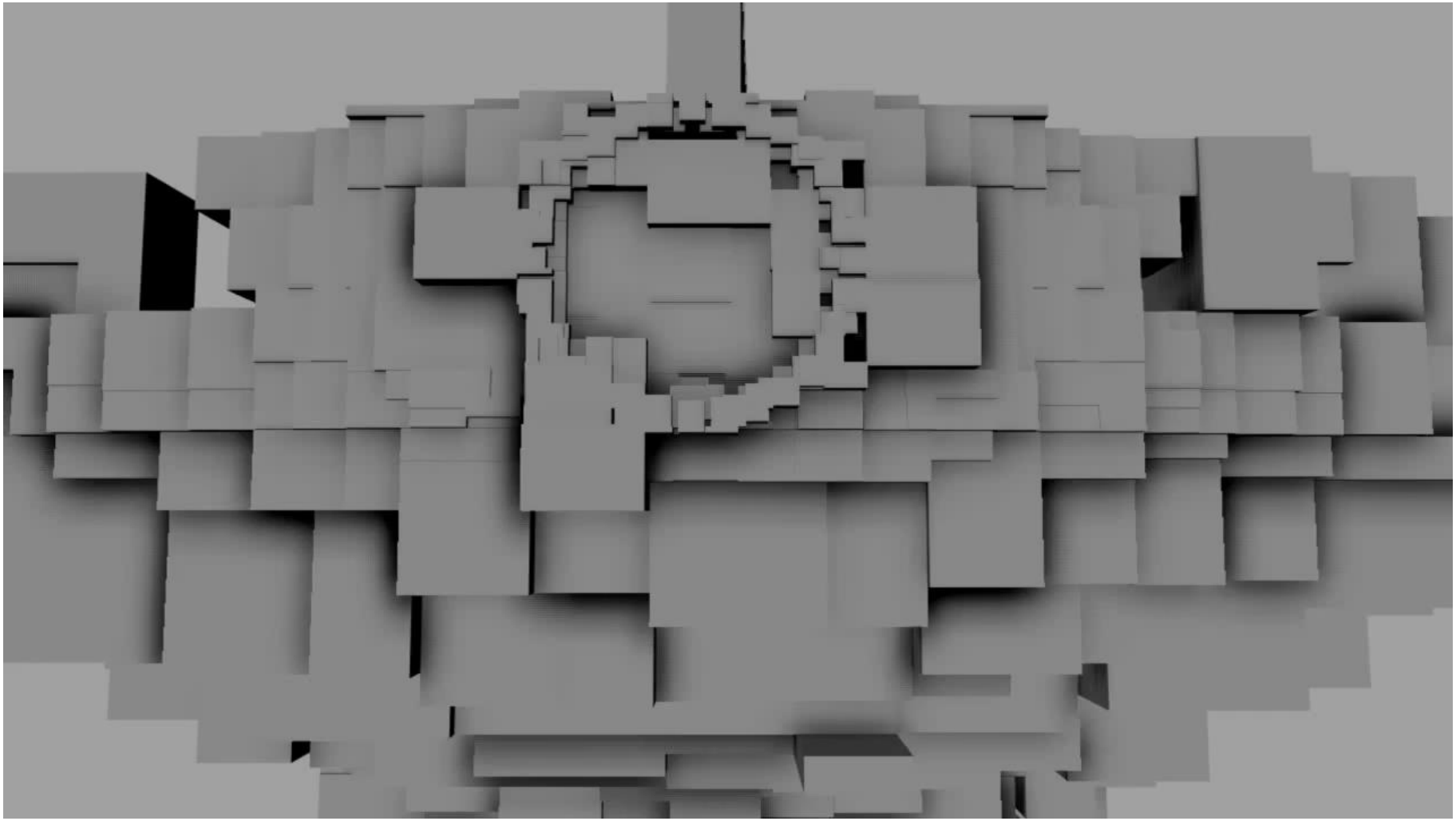


K-d stromy



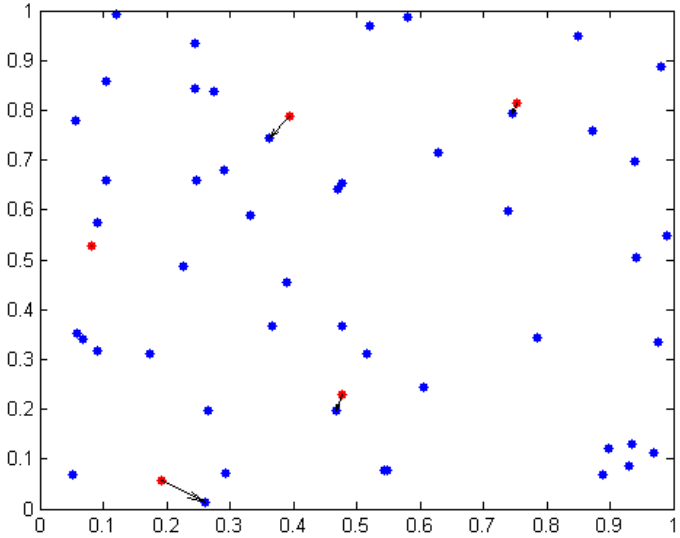


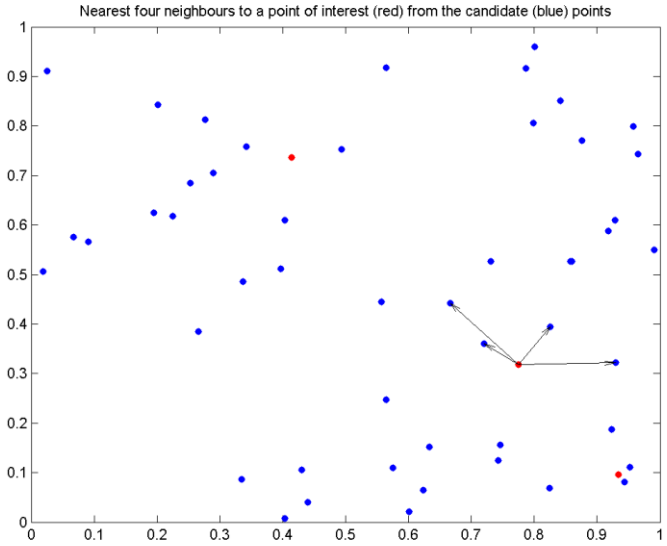
K-d stromy



Hladanie najbližšieho suseda

- Pre množinu bodov S z R^d a jeden bod P z R^d, treba nájsť taký bod Q z S aby vzdialenosť $|PQ|$ bola minimálna
- Rozšírenie v podobe nájdenia k najbližších susedov, poprípade približných susedov





Hľadanie najbližšieho suseda

```
FindNearestPoint(T, near_node, P)
{
   nearest_node = current_node = near_node;
   while (current_node != NULL)
   {
      hyperplane_distance = Distance(P, current_node->parent->dim, current_node->parent->split);
      if (hyperplane_distance < Distance(P, nearest_node))
      {
         if (current_node == current_node->parent->left)
            nearest_node = SearchSubtree(current_node->parent->right, nearest_node, P);
         else
            nearest_node = SearchSubtree(current_node->parent->left, nearest_node, P);
      }
      current_node = current_node->parent;
   }; return nearest_node;
}
```

```
SearchSubtree(v, nearest_node, P)
{
   List nodes;
   nodes.Add(v);
   current_nearest = nearest_node;
   while (nodes.size() > 0)
   {
      current_node = nodes.PopFirst();
      if (current_node->IsLeaf() &&  (current_node->point))
      {
         if (Distance(current_node->point, P) < Distance(current_nearest->point, P))
            current_nearest = current_node;
         continue;
      }
      hyperplane_distance = Distance(P, current_node->dim, current_node->split);
      if (hyperplane_distance > Distance(current_nearest->point, P))
      {
          if (InLeftPart(P, current_node->dim, current_node->split)) nodes.Add(current_node->left);
          else nodes.Add(current_node->right);
      }
      else
      {
         nodes.Add(current_node->left);
         nodes.Add(current_node->right);
       }
   }
   return current_nearest;
}
```

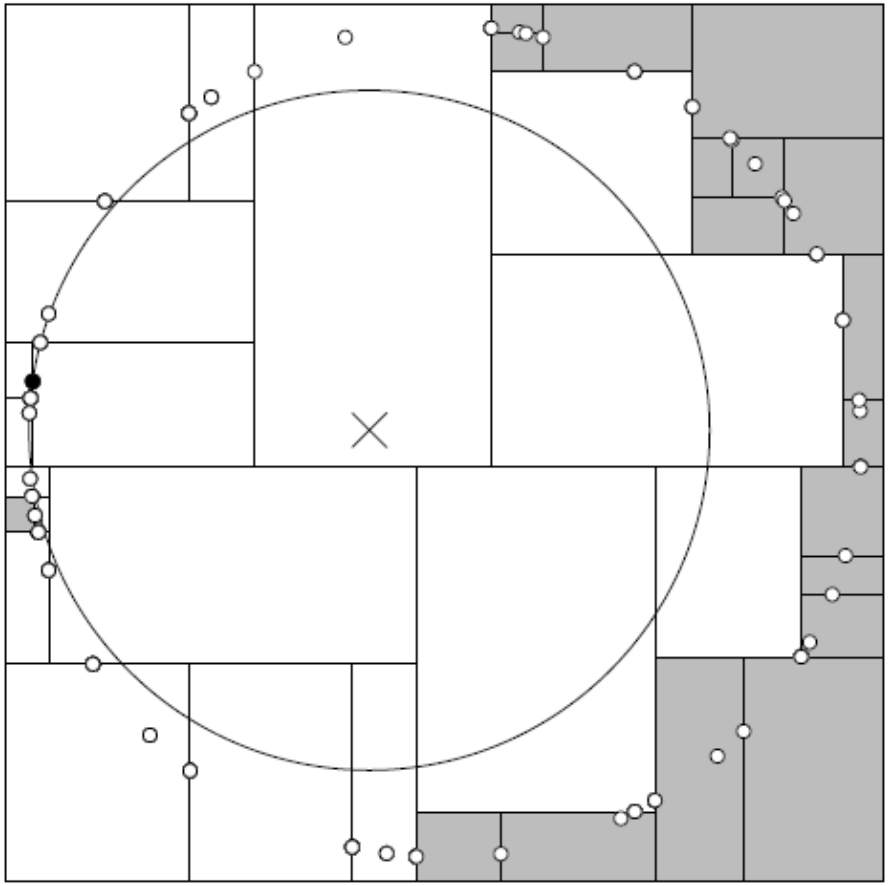
```
FindNearPoint(v, P)
{
   if (v->IsLeaf() &&  (v->point))
      return v;
   if (InLeftPart(P, v->dim, v->split))
      return FindNearPoint(v->left, P);
   else
      return FindNearPoint(v->right, P);
}
```

```
KdTreeNearestNeighbor(T, P)
{
   near = FindNearPoint(T->root, P);
   return FindNearestPoint(T, near, P);
}
```

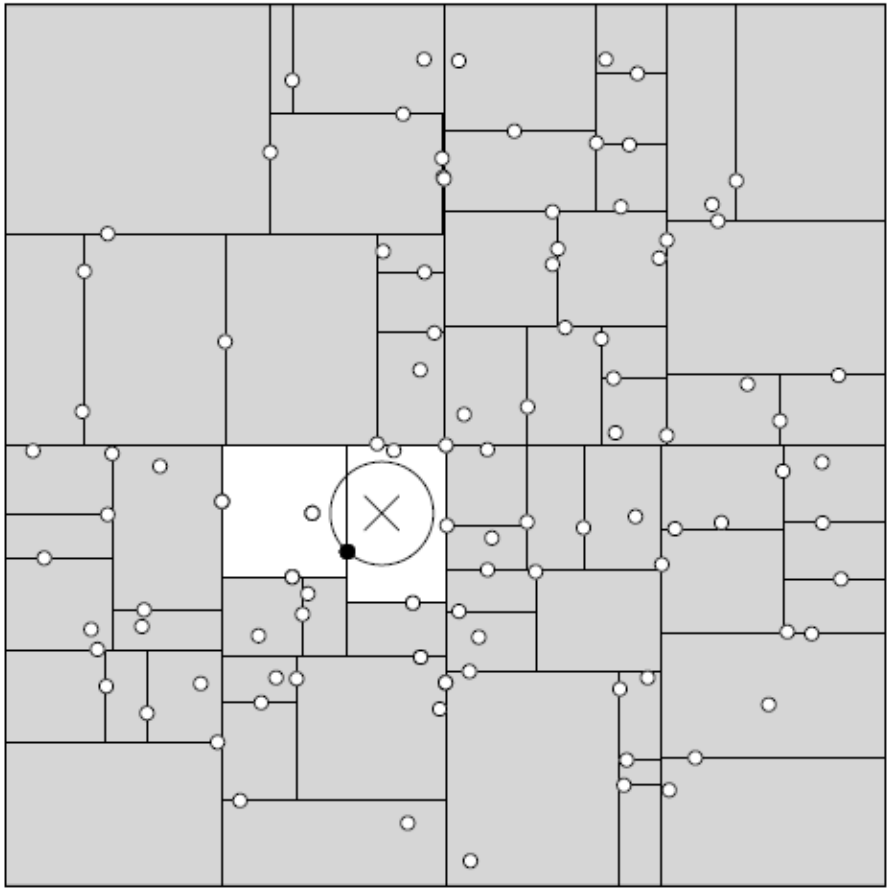
Hľadanie najbližšieho suseda

- Najprv nájdeme bod (list k-d stromu) ktorý by mal byť dosť blízko P
- Od tohoto listu prechádzame až ku koreňu stromu a hľadáme ešte bližšie body v opačných podstromoch
- Časová náročnosť: $O(d.\ n^{(1-1/d)})$
- Pre náhodne rozmiestnené body je očakávaná časová zložitosť O(log(n))
- http://dl.acm.org/citation.cfm?id=355745
- http://dimacs.rutgers.edu/Workshops/MiningTutorial/pindyk-slides.ppt
- http://www.ri.cmu.edu/pub_files/pub1/moore_andrew_1991_1/moore_andrew_1991_1.pdf

Hľadanie najbližšieho suseda



Mnoho navštívených vrcholov



Málo navštívených vrcholov

k najbližších susedov

- Rozšírenie predchádzajúceho algoritmu
- Namiesto sféry obsahujúcej iba 1 najbližší bod pracujeme so sférou, ktorá obsahuje *k* aktuálne najbližších bodov
- Pokiaľ sféra odsahuje menej ako *k* bodov, jej polomer je nekonečný
- Pri prvom prechode stromom nenájdeme 1 potencionálne najbližší bod, ale *k* potencionálne najbližších bodov

Hľadanie k najbližších susedov

```
Struct SearchRecord
{
   vector<KdTreeNode> points;
   float radius;
}
```

```
KdTreeNearestNeighbors(T, P, k)
{
   SearchRecord result;
   FindNearPoints(T->root, P, k, &result);
   FindNearestPoints(T, P, k, &result);
   return result;
}
```

```
FindNearPoints(v, P, k, result)
{
   if (v->IsLeaf() &&  (v->point))
   {
      result->points.Add(v);
      result->UpdateRadius(P);
      return;
   }
   if (InLeftPart(P, v->dim, v->split))
   {
      FindNearPoints(v->left, P, k, result);
      if (result->points.size < k)
         FindNearPoints(v->right, P, k, result);
   }
   else
   {
      FindNearPoints(v->right, P, k, result);
      if (result->points.size < k)
         FindNearPoints(v->left, P, k, result);
   }
}
```

Hľadanie k najbližších susedov

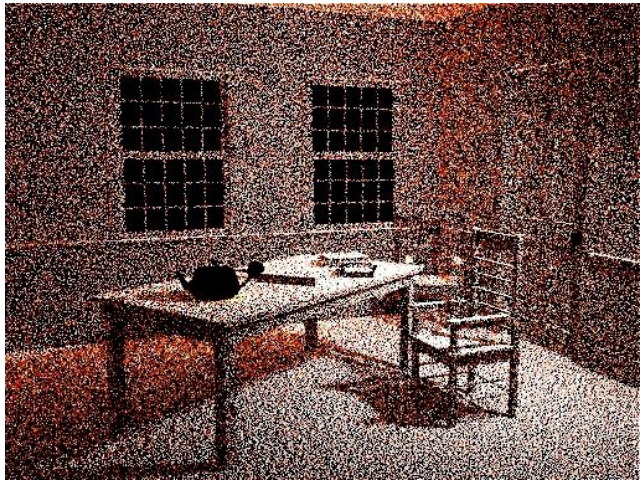
```
FindNearestPoints(T, P, k, result)
{
   current_node = result->points[0];
   while (current_node != NULL)
   {
      hyperplane_distance = Distance(P, current_node->parent->dim,
                                                 current_node->parent->split);
      if (hyperplane_distance < result->radius)
      {
         if (current_node == current_node->parent->left)
            SearchSubtree(current_node->parent->right, P, k, result);
         else
            SearchSubtree(current_node->parent->left, P, k, result);
       }
       current_node = current_node->parent;
   };
   return;
}
```

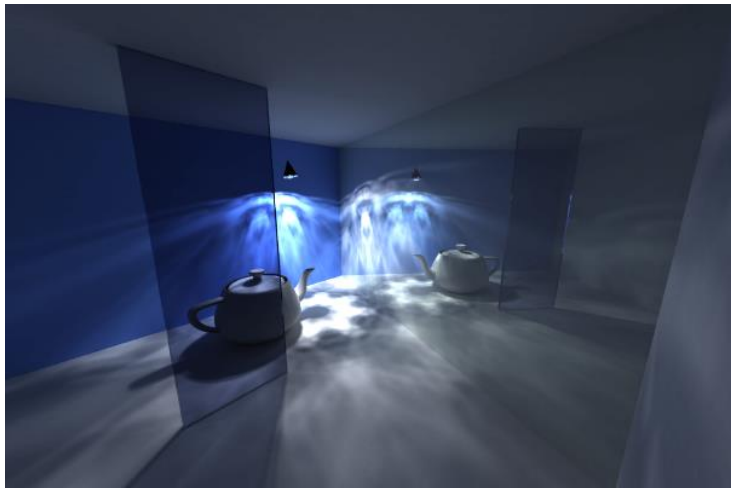
```
SearchSubtree(v, P, k, result)
{
   List nodes;
   nodes.Add(v);

   while (nodes.size() > 0)
   {
      current_node = nodes.PopFirst();
      if (current_node->IsLeaf() &&  (current_node->point))
      {
         if (Distance(current_node->point, P) < result->radius)
         {
            result->points.AddNewAndRemove(current_node, P, k);
            result->UpdateRadius(P);
         }
         continue;
      }
      hyperplane_distance = Distance(P, current_node->dim, current_node->split);
      if (hyperplane_distance > result->radius)
      {
          if (InLeftPart(P, current_node->dim, current_node->split))
             nodes.Add(current_node->left);
          else
             nodes.Add(current_node->right);
      }
      else
      {
         nodes.Add(current_node->left);
         nodes.Add(current_node->right);
       }
   }
   return;
}
```

Photon mapping

- 1 prechod:
  - strieľanie fotónov zo zdrojov svetla
  - zisťovanie nárazov a odrazov fotónov so scénou
  - ukladanie bodov nárazu do mapy
- 2 prechod:
  - rendering z pohľadu kamery
  - použitie dát z mapy (nájdenie k najbližších fotónov pre povrchový bod) pre výpočet globálnej iluminácie
- Štruktúra pre mapu fotónov = k-d strom



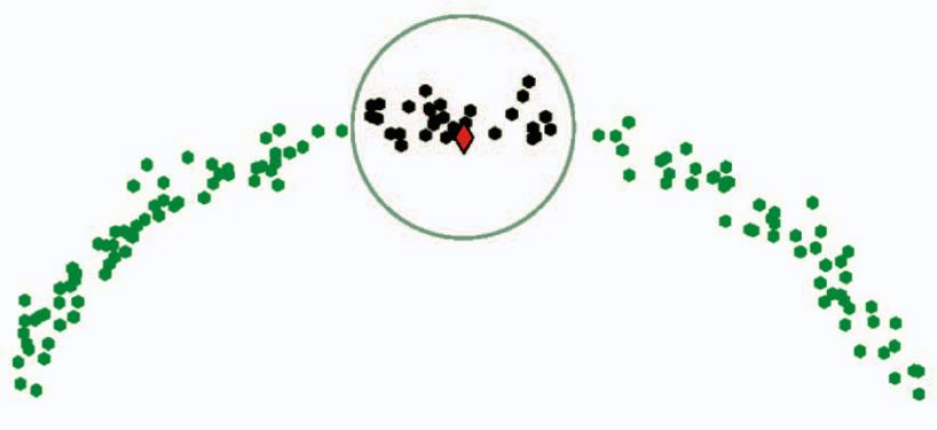


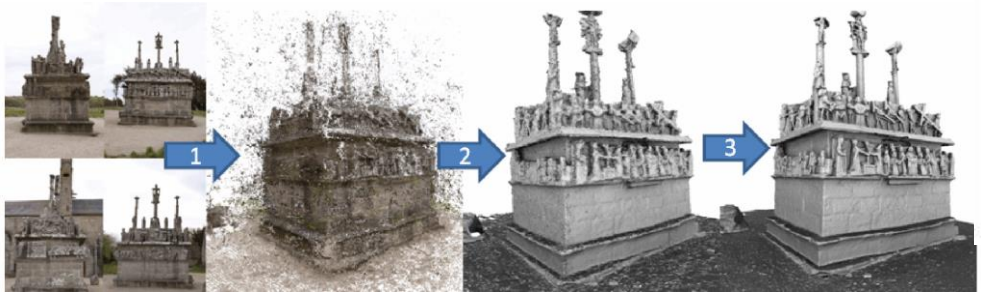
Mračná bodov

- Mnoho spôsobov generovania: laserové skenovanie, Kinect, štruktúrované svetlo
- Rekonštrukcia povrchov – potreba hľadania najbližších bodov

Rekonštrukcia povrchov

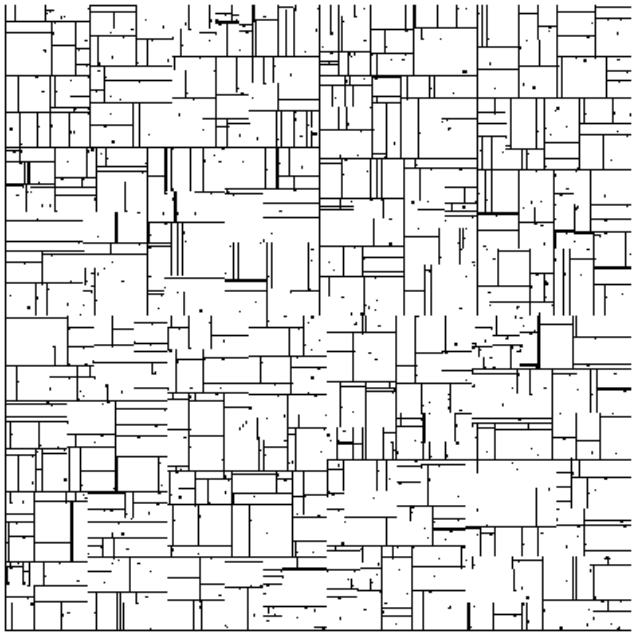
- Hľadanie bodov ktoré ležia v danej guli
- Malá modifikácia predchádzajúceho alg.

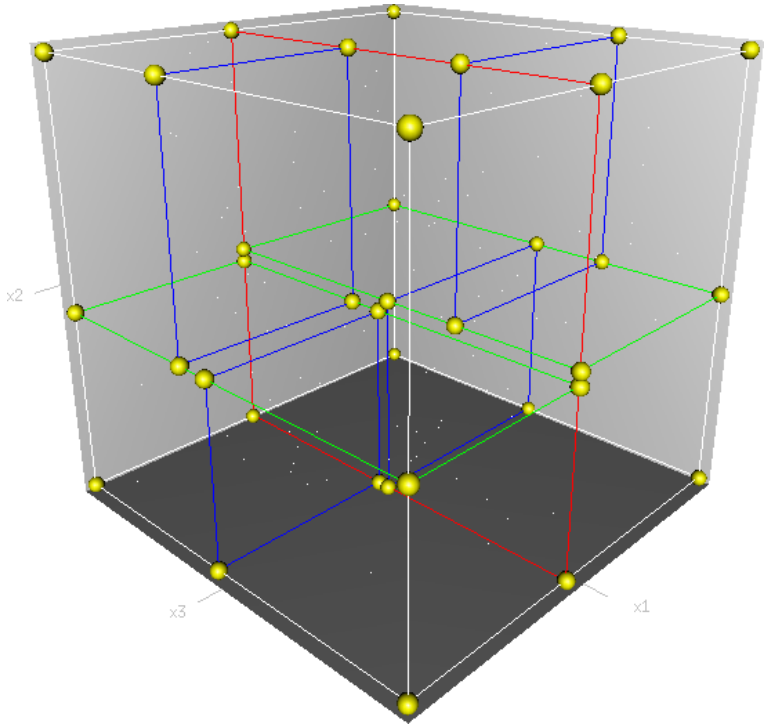




Databázy

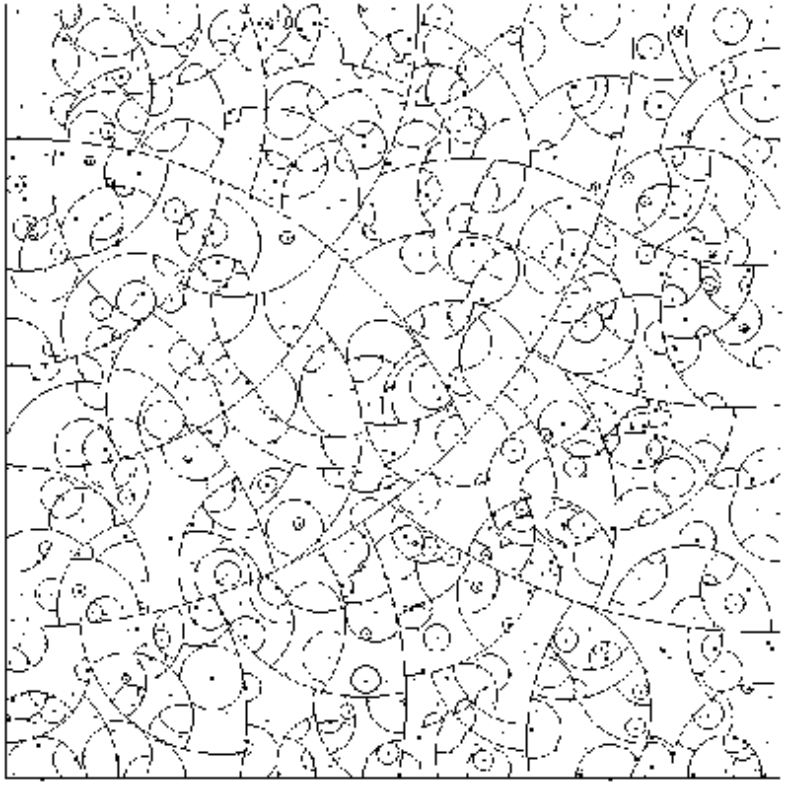
- Záznam v databáze = d-rozmerný vektor
- Pre daný záznam, nájdi najpodobnejší záznam v databáze = nájdenie najbližšieho suseda v databáze





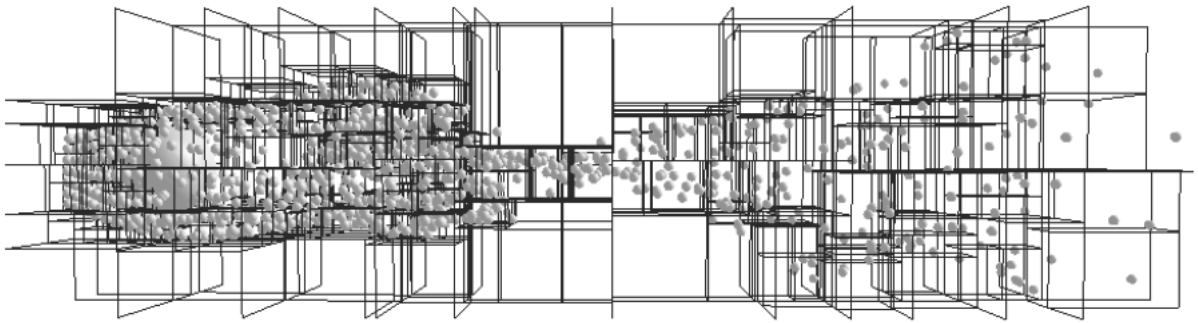
vp (Vantage-point) strom

- Binárny strom, v každom vrchole je určený bod **P** a polomer *r*, v ľavom podstrome sú vrcholy vzdialené od **P** nanajvýš *r*, v pravom podstrome zvyšné body
- Výber **P** – náhodný bod
- Výber *r* – medián vzdialeností **P** od ostatných bodov



Sledovanie lúča

- K-d strom je najlepšie delenie priestoru pre raytracing (minimalizuje počet rátaní priesečníka lúč-objekt)
- http://dcgi.felk.cvut.cz/home/havran/phdthesis.html
- Možnosť adaptívneho delenia podľa povrchu
- Sekvenčné hladanie pozdĺž lúča



Sledovanie lúča

- Rozdeľovanie vo vrcholoch
  - Spatial median
  - Object median
  - Smer – najväčšia variácia
  - V strede v smere "najdlhšej" dimenzie
  - Ohodnocovacie techniky
    - Ohodnotenie rozdelenia na základe pravdepodobnosti zásahu lúčom (ordinary surface area heuristic)

$$C_{v^G} = \frac{1}{SA(\mathcal{AB}(v^G))}[SA(\mathcal{AB}(lchild(v^G))).(N_L + N_{SP}) + SA(\mathcal{AB}(rchild(v^G))).(N_R + N_{SP})],$$

Otázky?