

Diskrétné Geometrické Štruktúry

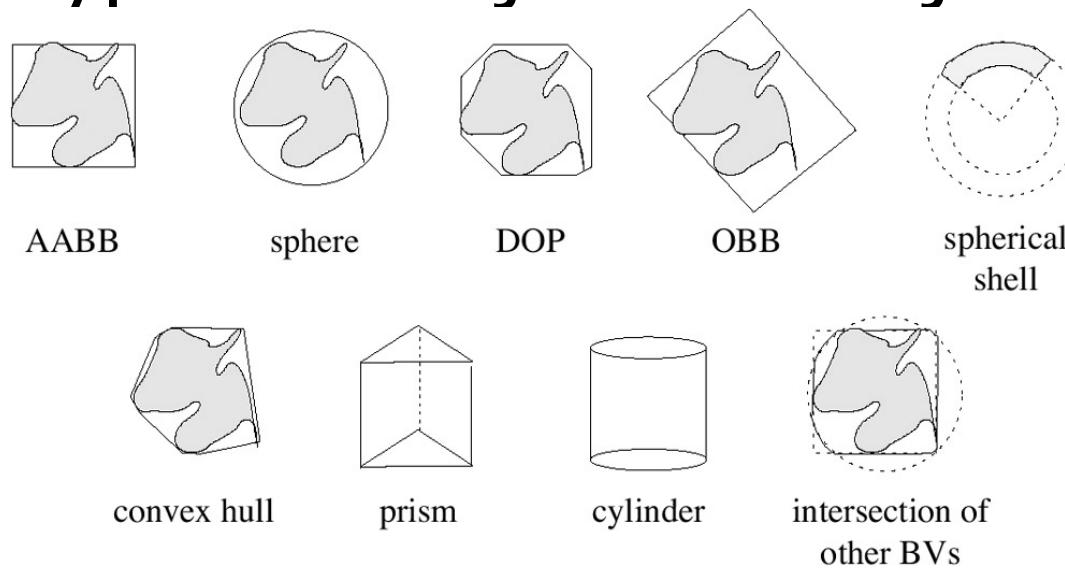
4. Ohraničujúce Objemy

Martin Samuelčík

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

Ohraničujúce objemy

- Pomocná množina ohraničujúca jeden alebo viac objektov
- Jednoduchý tvar objemu, nevhodné na reprezentáciu
- Prenos výpočtu z objektu na objem



Ohraničujúce objemy

- <http://www.geometrictools.com/index.html>
- Vhodnosť podľa algoritmov
 - Vytvorenie objemu z množiny bodov, ...
 - Presnosť aproximácie
 - Prienik s iným ohraničujúcim objemom
 - Prienik s priamkou
 - Príslušnosť bodu k objemu
 - Transformácie objektu (rotácia, škálovanie posunutie)
 - Vytvorenie objemu z dvoch menších objemov

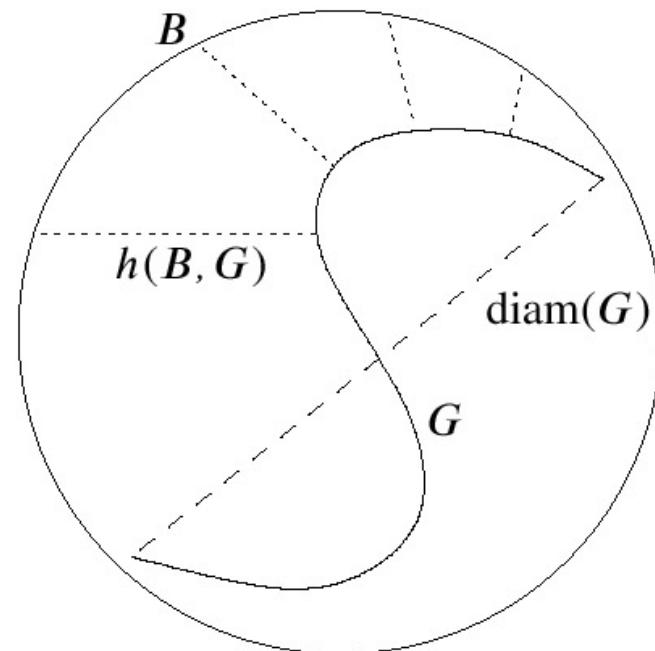
Presnosť' aproximácie

- Ohodnenie blízkosti ohraničujúceho objemu k objektu
- Použitie Haussdorfovej vzdialenosťi
- Záleží aj na veľkosti objemu

$$h(B, G) = \max_{b \in B} \min_{g \in G} d(b, g)$$

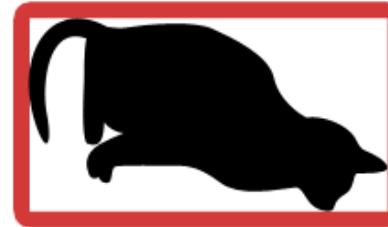
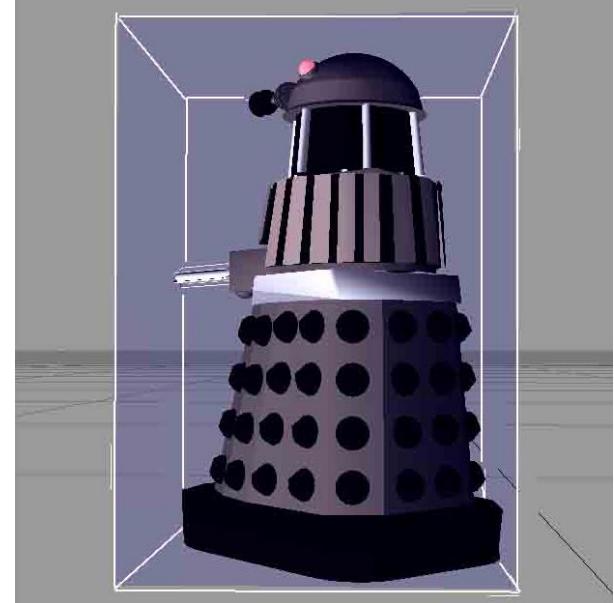
$$\text{diam}(G) = \max_{g, f \in G} d(g, f)$$

$$\tau := \frac{h(B, G)}{\text{diam}(G)}.$$



AABB

- Axis-aligned bounding box
- d-rozmerný interval
- Jednoduché vybudovanie objemu
- Jednoduché prieniky
- Zložitá transformácia
- Horšia aproximácia, až 0.5



AABB

- Vytvorenie – nájdenie min-max hodnôt
- Vzájomný prienik – porovnanie hodnôt
- Transformácia - nutnosť prepočítania

```
struct AABB
{
    float fMinX, fMaxX;
    float fMinY, fMaxY;
    float fMinZ, fMaxZ;
}
```

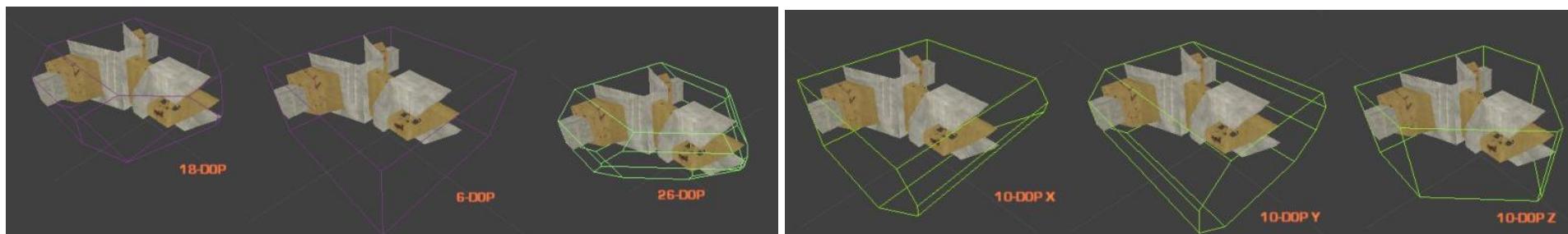
```
AABBIntersection(AABB v1, AABB v2)
```

```
{  
    if (v1.fMaxX < v2.fMinX) return false;  
    if (v1.fMinX > v2.fMaxX) return false;  
    if (v1.fMaxY < v2.fMinY) return false;  
    if (v1.fMinY > v2.fMaxY) return false;  
    if (v1.fMaxZ < v2.fMinZ) return false;  
    if (v1.fMinZ > v2.fMaxZ) return false;  
    return true;  
}
```

```
CreateAABB(vector<Point3D> points)
{
    AABB result;
    result.fMinX = result.fMinY = result.fMinZ = MAX_FLOAT;
    result.fMaxX = result.fMaxY = result.fMaxZ = MIN_FLOAT;
    for (int i = 0;i < points.size();i++)
    {
        if (points[i].x < result.fMinX) result.fMinX = points[i].x;
        if (points[i].x > result.fMaxX) result.fMaxX = points[i].x;
        if (points[i].y < result.fMinY) result.fMinY = points[i].y;
        if (points[i].y > result.fMaxY) result.fMaxY = points[i].y;
        if (points[i].z < result.fMinZ) result.fMinZ = points[i].z;
        if (points[i].z > result.fMaxZ) result.fMaxZ = points[i].z;
    }
    return result;
}
```

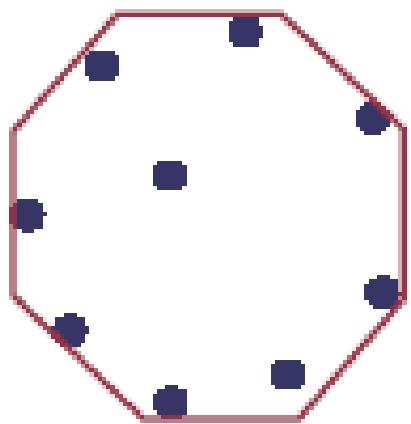
k-DOP

- Discrete Oriented Polytope, Polytop – polygon, polyhedron...
- Prienik k polpriestorov, konvexný obal je najmenší k-DOP
- Najčastejšie sa definuje k / 2 smerov a k nim sa nájdu min a max hodnoty v každom smere
- AABB je k-DOP so smermi danými súradnicovými osami

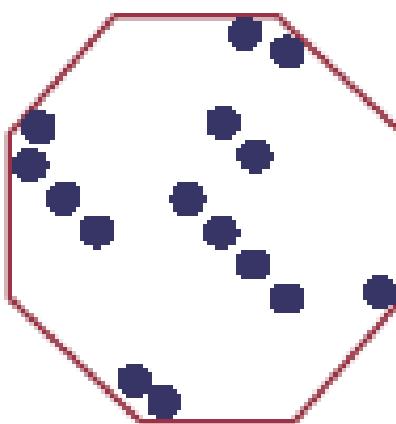


k-DOP

- Lepšia aproximácia, záleží na objekte
- 2D: $k = 4, 8$; 3D: $k = 6, 18, 26$
- Podobné zložitosti algoritmov ako pre AABB



good choice



quite good choice

```
struct kDOP
{
    static vector<Vector3> directions;
    Point3 center;
    vector<float> min_values;
    vector<float> max_values;
}
```

k-DOP

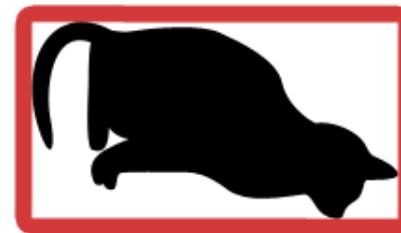
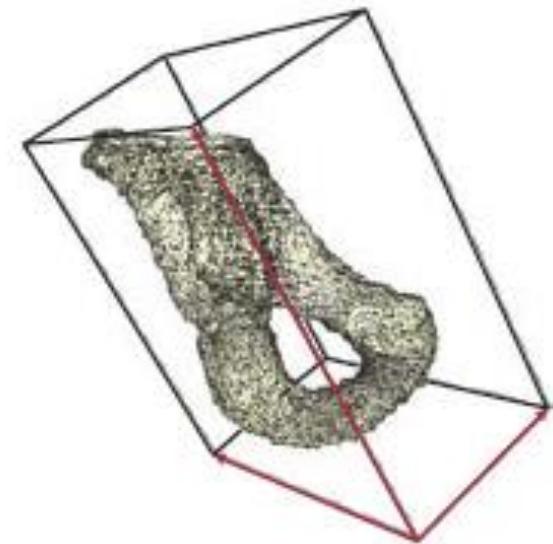
```
CreatekDOP(vector<Point3D> points)
{
    kDOP result;
    result.center.x = result.center.y = result.center.z = 0;
    for (int i = 0;i < points.size();i++)
    {
        result.center.x += points[i].x / points.size();
        result.center.y += points[i].y / points.size();
        result.center.z += points[i].z / points.size();
    }
    for (int j = 0;j < result.directions.size();j++)
    {
        result.min_values[j].push_back(MAX_FLOAT);
        result.max_values[j].push_back(MIN_FLOAT);
    }
    for (int i = 0;i < points.size();i++)
    {
        Point3 X = points[i];
        for (int j = 0;j < result.directions.size();j++)
        {
            Vector3 v = result.directions[j]; Point3 O = result.center;
            float t = (X.x*v.x+X.y*v.y+X.z*v.z-O.x*v.x-O.y*v.y-O.z*v.z);
            t = t / (v.x*v.x+v.y*v.y+v.z*v.z);
            if (t < result.min_values[j]) result.min_values[j] = t;
            if (t > result.max_values[j]) result.max_values[j] = t;
        }
    } return result;
}
```

```
kDOPIntersection(kDOP v1, kDOP v2)
{
    Point3 O1 = v1.center; Point3 O2 = v2.center;
    for (int j = 0;j < v1.directions.size();j++)
    {
        Vector3 v = v1.directions[j];
        float t = (O2.x*v.x+O2.y*v.y+O2.z*v.z-O1.x*v.x-O1.y*v.y-O1.z*v.z);
        t = t / (v.x*v.x+v.y*v.y+v.z*v.z);
        if (v1.min_values[j] > (v2.max_values[j] + t)) return false;
        if (v1.max_values[j] < (v2.min_values[j] + t)) return false;
    }
    return true;
}
```

```
PointInsidekDOP(kDOP v,Point3 P)
{
    Point3 O = v.center;
    for (int j = 0;j < v.directions.size();j++)
    {
        Vector3 v = v.directions[j];
        float t = (P.x*v.x+P.y*v.y+P.z*v.z-O.x*v.x-O.y*v.y-O.z*v.z);
        t = t / (v.x*v.x+v.y*v.y+v.z*v.z);
        if (v.min_values[j] > t) return false;
        if (v.max_values[j] < t) return false;
    }
    return true;
}
```

OBB

- Oriented Bounding Box
- Rotovaný d-rozmerný interval
- Presnejšia aproximácia
- Časovo náročnejšie testy
- Ľahšia transformácia



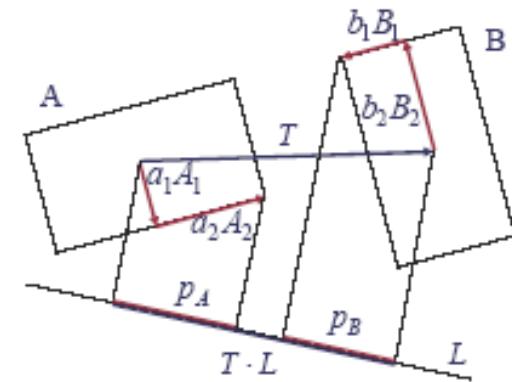
OBB

- Prienik dvoch OBB – nájdenie „separating axis“
 - priamka na ktorej sa projekcie polytopov nepretínajú
- Dva objekty A a B sa nepretínajú ak existuje priamka p taká, že priemety A a B na priamku p sa nepretínajú
- Smer p je definovaný orientáciou stien A a B a vektorovým súčinom hrán z A a B (v 3D)

$$p_A = |a_1 A_1 L| + |a_2 A_2 L|$$

$$p_B = |b_1 B_1 L| + |b_2 B_2 L|$$

$$\exists L : |T \cdot L| > p_A + p_B \quad \text{or} \quad \exists L \in \{A_1, A_2, B_1, B_2\} : |T \cdot L| > p_A + p_B$$



OBB

```
struct OBB
{
    Point3 center;
    Vector3 v1, v2, v3;
    float fMin1, fMin2, fMin3;
    float fMax1, fMax2, fMax3;
}
```

```
ComputeDirections(vector<Point3> points)
{
    Point3 center;
    center.x = center.y = center.z = 0;
    for (int i = 0;i < points.size();i++)
    {
        center.x += points[i].x / points.size();
        center.y += points[i].y / points.size();
        center.z += points[i].z / points.size();
    }
    Matrix3 C; C[0][0] = C[1][0] = C[2][0] = C[0][1] = C[1][1] = ... = 0;
    for (int i = 0;i < points.size();i++)
    {
        C[0][0] += (points[i].x - center.x)*(points[i].x - center.x) / points.size();
        C[1][0] += (points[i].y - center.y)*(points[i].x - center.x) / points.size();
        C[2][0] += (points[i].z - center.z)*(points[i].x - center.x) / points.size();
        C[0][1] += (points[i].x - center.x)*(points[i].y - center.y) / points.size();
        ....
    }
    return C.Eigenvectors();
}
```

```
CreateOBB(vector<Point3> points)
{
    OBB result;
    result.center.x = result.center.y = result.center.z = 0;
    for (int i = 0;i < points.size();i++)
    {
        result.center.x += points[i].x / points.size();
        result.center.y += points[i].y / points.size();
        result.center.z += points[i].z / points.size();
    }
    (result.v1, result.v2, result.v3) = ComputeDirections(points);
    result.fMin1 = result.fMin2 = result.fMin3 = MAX_FLOAT;
    result.fMax1 = result.fMax2 = result.fMax3 = MIN_FLOAT;
    for (int i = 0;i < points.size();i++)
    {
        Point3 X = points[i];
        float t1 = (X.x*v1.x+X.y*v1.y+X.z*v1.z);
        t1 = t1 / (v1.x*v1.x+v1.y*v1.y+v1.z*v1.z);
        if (t1 < result.fMin1) result.fMin1 = t1;
        if (t1 > result.fMax1) result.fMax1 = t1;
        // .... Same for other two directions
    }
    return result;
}
```

OBB

```
PointInsideOBB(Point3 P, OBB v)
{
    Point3 O = v.origin;
    float t = (P.x*v.v1.x+P.y*v.v1.y+P.z*v.v1.z-O.x*v.v1.x-O.y*v.v1.y-O.z*v.v1.z);
    t = t / (v.v1.x*v.v1.x+v.v1.y*v.v1.y+v.v1.z*v.v1.z);
    if (t < v.fMin1) return false;
    if (t > v.fMax1) return false;
    t = (P.x*v.v2.x+P.y*v.v2.y+P.z*v.v2.z-O.x*v.v2.x-O.y*v.v2.y-O.z*v.v2.z);
    t = t / (v.v2.x*v.v2.x+v.v2.y*v.v2.y+v.v2.z*v.v2.z);
    if (t < v.fMin2) return false;
    if (t > v.fMax2) return false;
    t = (P.x*v.v3.x+P.y*v.v3.y+P.z*v.v3.z-O.x*v.v3.x-O.y*v.v3.y-O.z*v.v3.z);
    t = t / (v.v3.x*v.v3.x+v.v3.y*v.v3.y+v.v3.z*v.v3.z);
    if (t < v.fMin3) return false;
    if (t > v.fMax3) return false;
    return true;
}
```

```
RayPlaneIntersection(Ray r, Vector3 normal, float plane4)
{
    Point3 result;
    float t = -plane4 - normal.x*r.start.x - normal.y*r.start.y - normal.z*r.start.z;
    t = t / (normal.x*v.x+normal.y*v.y+normal.z*v.z);
    result.x = r.start.x + t * v.dir.x;
    result.y = r.start.y + t * v.dir.y;
    result.z = r.start.z + t * v.dir.z;
    return result;
}
```

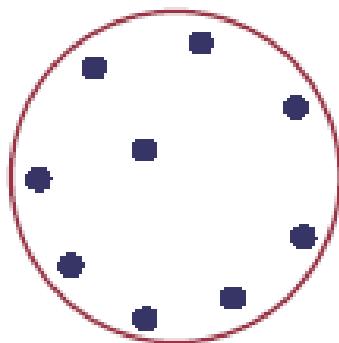
```
struct Ray
{
    Point3 start;
    Vector3 dir;
}
```

```
RayOBBIntersection(Ray r, OBB v)
{
    Point3 plane;
    plane.x = v.origin.x + v.fMin1*v.v1.x;
    plane.y = v.origin.y + v.fMin1*v.v1.y;
    plane.z = v.origin.z + v.fMin1*v.v1.z;
    float d = -v.v1.x*plane.x-v.v1.y*plane.y-v.v1.z*plane.z;
    Point3 I = RayPlaneIntersection(r, v.v1, d);
    if (PointInsideOBB(I, v))
        return true;

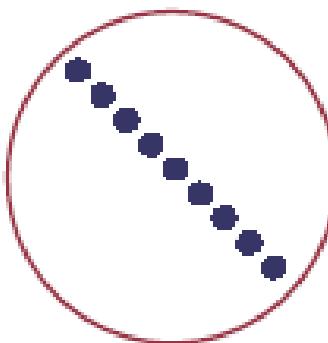
    plane.x = v.origin.x + v.fMax1*v.v1.x;
    plane.y = v.origin.y + v.fMax1*v.v1.y;
    plane.z = v.origin.z + v.fMax1*v.v1.z;
    float d = -v.v1.x*plane.x-v.v1.y*plane.y-v.v1.z*plane.z;
    I = RayPlaneIntersection(r, v.v1, d);
    if (PointInsideOBB(I, v))
        return true;
    // same for remaining two directions v2, v3
    .....
    return false;
}
```

Sféra, elipsa

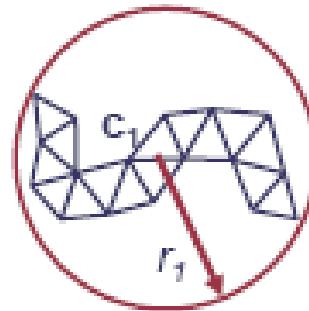
- Vhodná pre transformácie
- Ľahké výpočty, prieniky, transformácie
- Vytvorenie: najmenšia ohraničujúca kružnica, kružnica ohraničujúca AABB, OBB



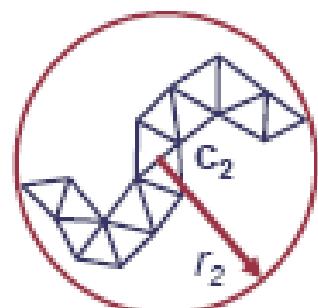
good choice



bad choice



$$(c_1 - c_2)(c_1 - c_2) > (r_1 + r_2)^2$$



Sféra

```
struct BoundingSphere
{
    Point3 center;
    float radius;
}
```

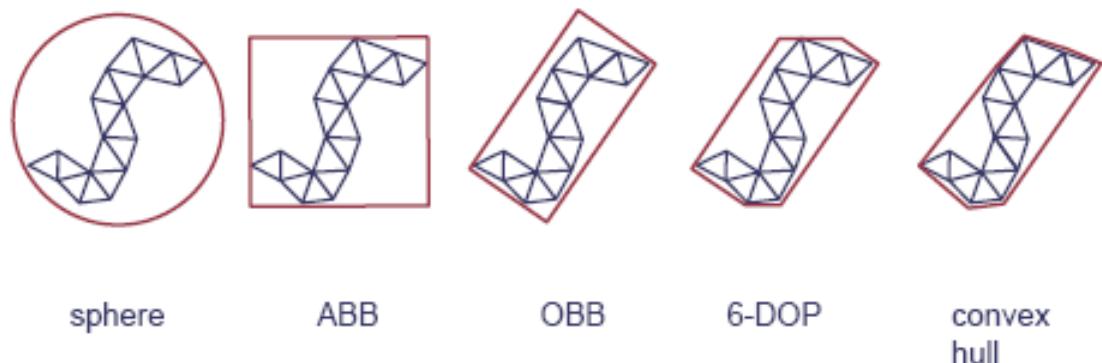
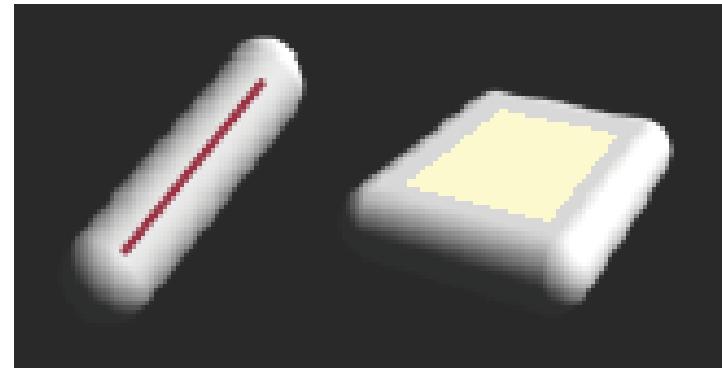
```
CreateBoundingSphere(vector<Point3> points)
{
    BoundingSphere result;
    result.center.x = result.center.y = result.center.z = 0;
    for (int i = 0;i < points.size();i++)
    {
        result.center.x += points[i].x / points.size();
        result.center.y += points[i].y / points.size();
        result.center.z += points[i].z / points.size();
    }
    result.radius = 0;
    for (int i = 0;i < points.size();i++)
    {
        float t = (points[i].x - result.center.x)^2 +
                  (points[i].y - result.center.y)^2 +
                  (points[i].z - result.center.z)^2;
        if (t > result.radius) result.radius = t;
    }
    result.radius = sqrt(result.radius);
    return result;
}
```

```
BoundingSphereIntersection(BoundingSphere v1, BoundingSphere v2)
{
    float d = (v1.center.x - v2.center.x)^2 +
              (v1.center.y - v2.center.y)^2 +
              (v1.center.z - v2.center.z)^2;
    if ((v1.radius + v2.radius) > sqrt(d))
        return true;
    else
        return false;
}
```

```
RayBoundingSphereIntersection(BoundingSphere s, Ray r)
{
    Point3 S = s.center;
    Vector3 v = r.dir;
    Point3 O = r.start;
    float dis = v.x*(O.x-S.x)+v.y*(O.y-S.y)+v.z*(O.z-S.z);
    dis = dis*dis;
    dis = dis - (v.x*v.x+v.y*v.y+v.z*v.z)*
            ((O.x-S.x)^2+(O.y-S.y)^2+(O.z-S.z)^2-s.radius^2);
    if (dis < 0)
        return false;
    else
        return true;
}
```

Ďalšie objemy

- Elipsoidy
- Cylindre, hranoly
- Konvexné obaly
- Rozšírené sférické objemy
- Kombinácie

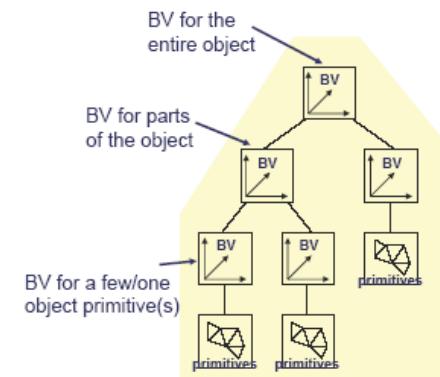


tighter approximation

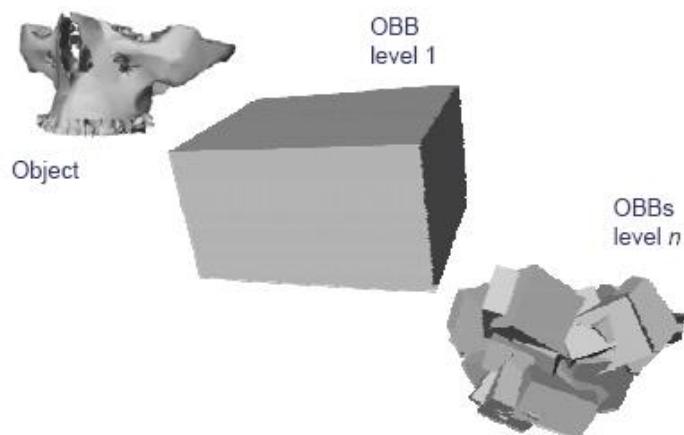
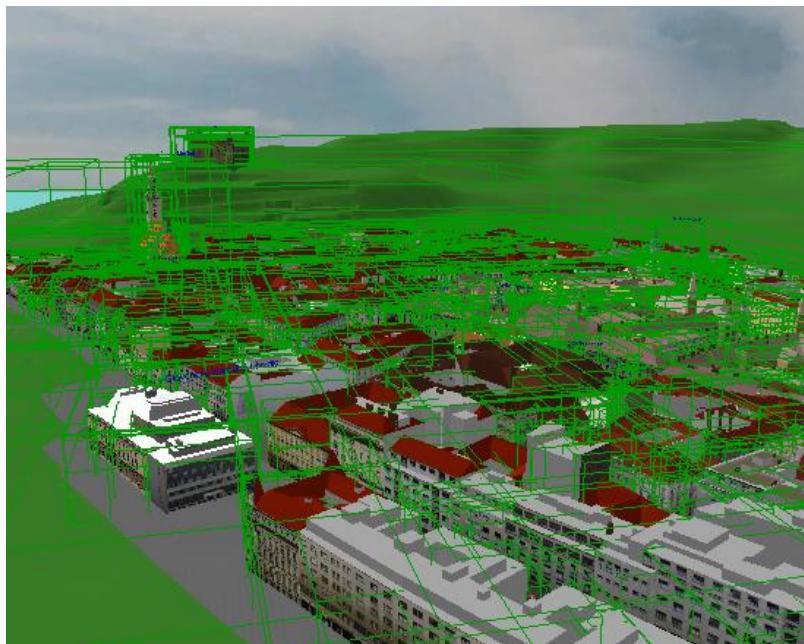
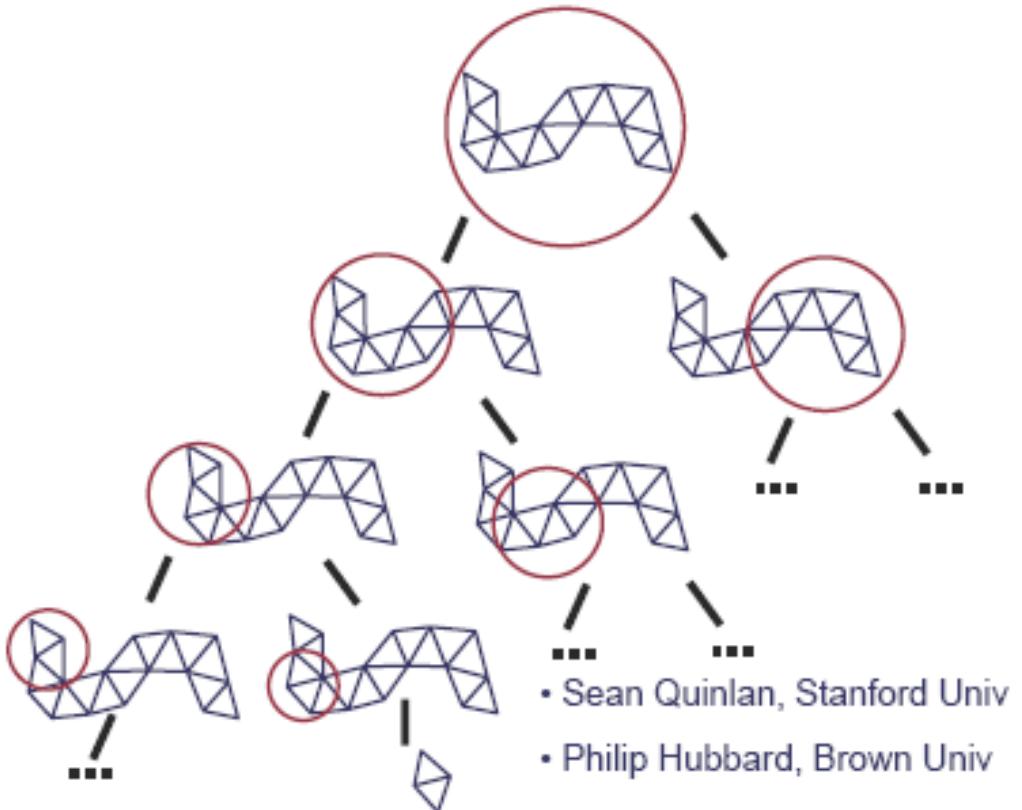
decreasing complexity and computational expenses for overlap test

Bounding Volumes Hierarchy

- Strom ohraničujúcich objektov
- Nech $O = \{o_1, \dots, o_n\}$ je množina objektov
- Potom je BVH pre množinu O definovaný:
 - ak $|O| \leq e$, tak $\text{BVH}(O)$ je list ktorý obsahuje O a ohraničujúci objem O
 - ak $|O| > e$, tak $\text{BVH}(O)$ je vrchol s m potomkami v_i , $i=1,\dots,m$, kde v_i sú $\text{BVH}(O_i)$ nad množinami O_i , $O = \cup O_i$, $\text{BVH}(O)$ obsahuje aj ohraničujúci objem O



Príklady BVH



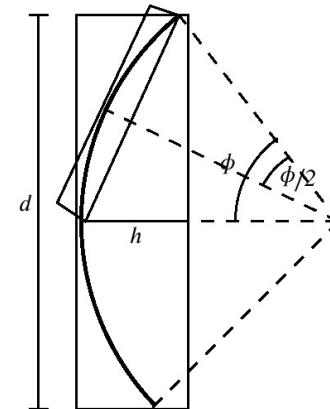
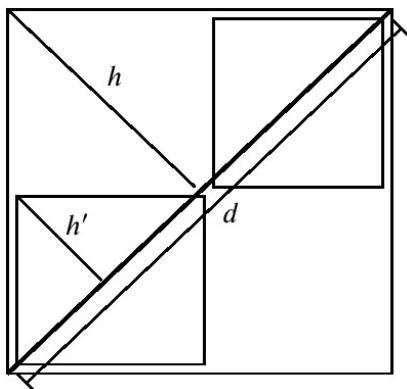
Tesnosť' aproximácie BVH

- Volumetrická tesnosť'
- $C(v)$ potomkovia v , $L(v)$ listy v

$$\tau := \frac{\text{Vol}(v)}{\sum_{v' \in C(v)} \text{Vol}(v')}.$$

$$\tau := \frac{\text{Vol}(v)}{\sum_{v' \in L(v)} \text{Vol}(v')},$$

- Pri AABB sa tesnosť' veľmi nemení, závisí na orientácii geometrie
- Pri OBB rastie, závisí na krivosti



Konštrukčné kritériá

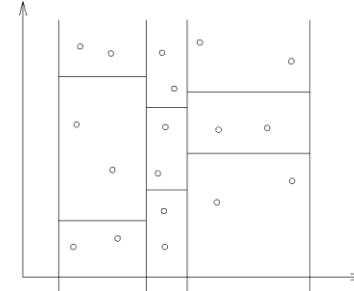
- Vyhľadanie v strome
- Delenie objektov alebo objemov
- Ako deliť objekty, objemy
- Minimalizácia redundancie (či sa objekt nachádza vo viacerých vrcholoch)
- Počet primitív v listoch
- Podľa použitia stromu
- Ohodnotenie vrcholov stromu

Konštrukcia BVH

- Bottom-up:
 - na začiatku máme jednotlivé objekty
 - vytvor ohraňujúci objem pre každý objekt
 - rekurzívne zoskupuj ohraňujúce objemy do väčších celkov
 - ukončenie ked' zostane jeden veľký objem

Spôsoby zoskupovania

- Greedy:
 - usporiadaj objemy podľa vzdialenosí od seba
 - v usporiadaní vyber prvých k objemov a vytvor pre ne nového predka v hierarchii
- Tiling:
 - pre každý objem vypočítaj stred
 - rozdel' priestor na $\sqrt[d]{n/k}$ rezov tak, aby v každom bol rovnaký počet primitív (podľa x a potom podľa y)
 - vznikne rozdelenie priestoru na diely s k prvkami, objemy v jednej bunke spojíme do jedného objemu
 - pokračujeme o úroveň vyššie



Konštrukcia BVH

- Vkladanie

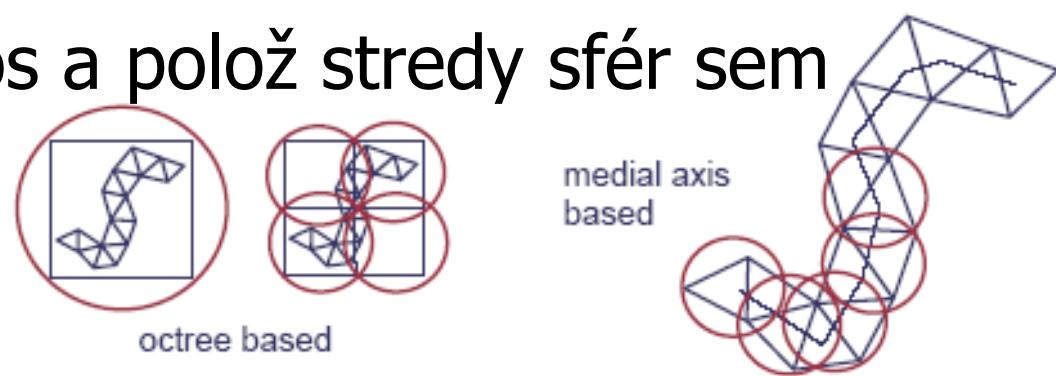
```
void BVH_Insert (B)
{
    while (|B| > 0)
    {
        b = B.pop; v = root; // vybratie dobreho b je dolezite
        while (!v.IsLeaf())
        {
            choose child v', so that insertion of b into v' causes minimal increase in the
            costs of the total tree;
            v = v';
        }
    }
}
```

Konštrukcia BVH

- Top-down
 - začneme s celou scénou, objektom
 - vytvoríme ohraňujúci objem pre scénu
 - objem (objekt) rozdelíme na k časti
 - prerozdelujeme
 - skončíme keď objem obsahuje podprahový počet primitív

Konštrukcia Sférických BVH

- Vytvor top-down BVH pomocou octree štruktúry
- Hubbard, O'Sullivan:
 - approximuj objekty pomocou sfér a vytvor strom bottom-up zoskupovaním sfér
 - pri tvorbe vytváraj homogénnu štruktúru vrcholov pre odstránenie redundancie
 - vypočítaj strednú os a polož stredy sfér sem



Ohodnotenia

- Raytracing: ohodnocovacia funkcia na základe plochy, $a = x, y, z, \dots$

$$k^\alpha = \min_{j=0 \dots n} \left\{ \frac{\text{Area}(b_1, \dots, b_j)}{\text{Area}(B)} j + \frac{\text{Area}(b_{j+1}, \dots, b_n)}{\text{Area}(B)} (n - j) \right\}$$

- Frustum culling: objem namiesto povrchu
- Collision detection: porovnávanie viacerých stromov a častí stromov, minimalizovať $C(A, B)$

$$C(A, B) = 4 + \sum_{i,j=1,2} P(A_i, B_j) \cdot C(A_i, B_j),$$

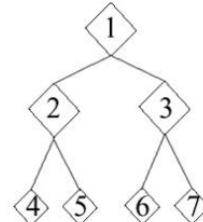
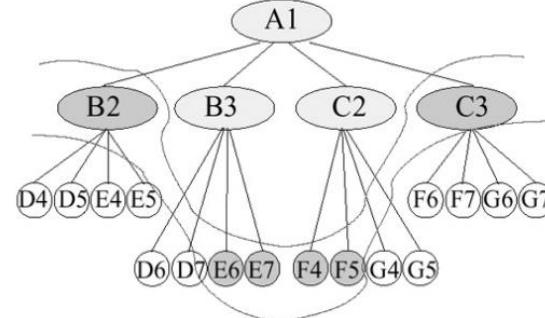
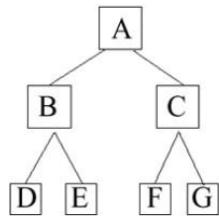
$$C(A, B) \approx 4(1 + P(A_1, B_1) + \dots + P(A_2, B_2)).$$

$$P(A_1, B_1) \approx \frac{\text{Vol}(A_1) + \text{Vol}(B_1)}{\text{Vol}(A) + \text{Vol}(B)}.$$

Detekcia kolízií

- Kontrola či sa nejaké objekty pretínajú → kontrola prieniku ich primitív
- Prechod dvoma stromami
- Vytváranie *bounding volume test tree* (BVTT)
- Možnosť využitia koherencie

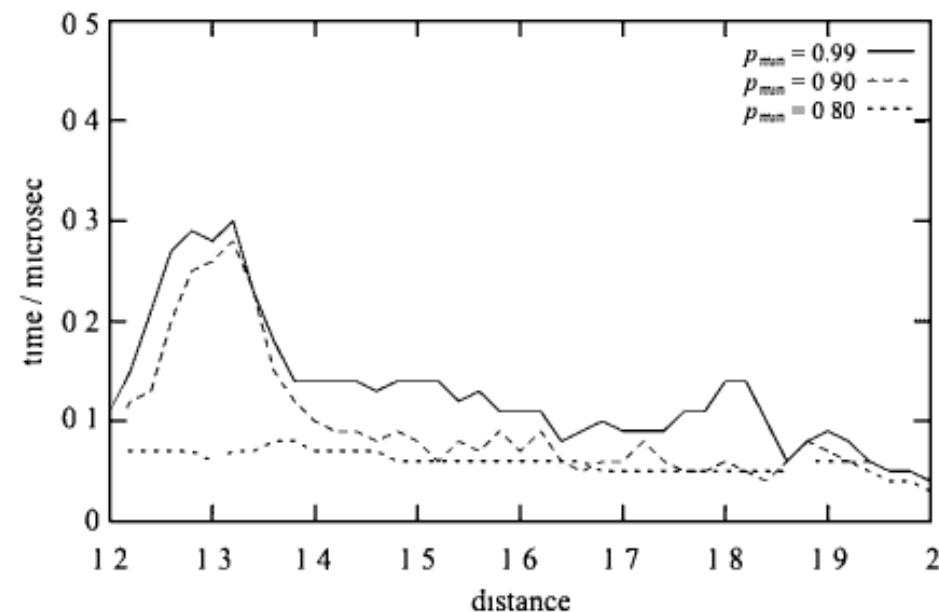
```
traverse(A,B)
{
    if (A ∩ B == 0)
        return NULL;
    if (A.IsLeaf() && B.IsLeaf())
        return A.primitives ∩ B.primitives;
    for (all children A[i] and B[j])
        traverse(A[i],B[j]);
}
```



Stochastická detekcia kolízií

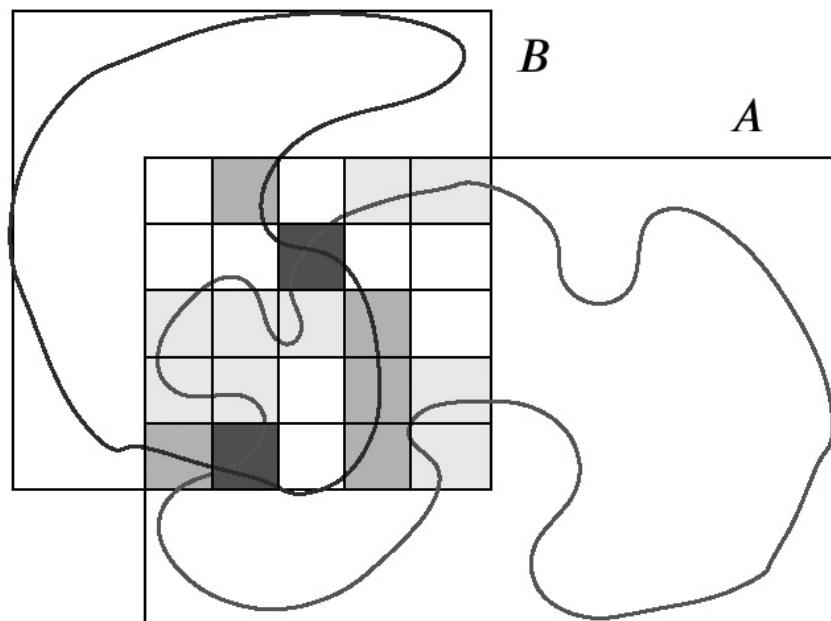
- Ohodnotíme časti stromu, ktoré sa budú spracuvávať s vyššou prioritou
- Pravdepodobnosť prieniku → externý opis správania sa primitív v objeme

```
traverse(A, B)
{
    q.insert(A, B, 1);
    while (!q.isEmpty())
    {
        (A, B) = q.pop();
        for (all children Ai and Bj)
        {
            p = P [Ai, Bj];
            if (p > thresh)
                return "collision";
            else if (p > 0)
                q.insert(Ai, Bj, p);
        }
    }
}
```



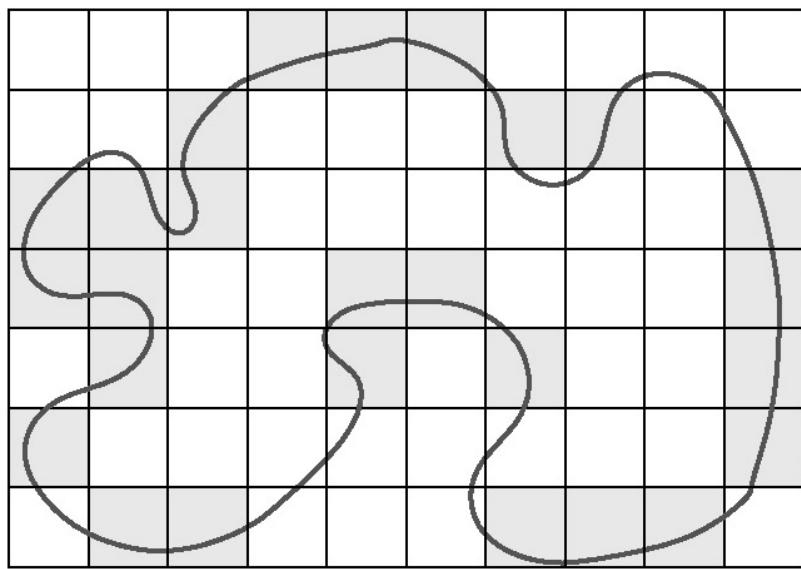
Pravdepodobnosť prieniku

- Rozdelíme $A \cap B$ na regulárnu mriežku
- V každej bunke určíme obsadenosť bunky množinou A a B
- Spočítaj počet kolíznych buniek



Stochastické riešenie

- V čase tvorby ohraničujúceho objemu
- Rozdelíme objem regulárrou mriežkou
- Zrátame počet dostatočne obsadených buniek a uložíme tento počet s objemom - s_A



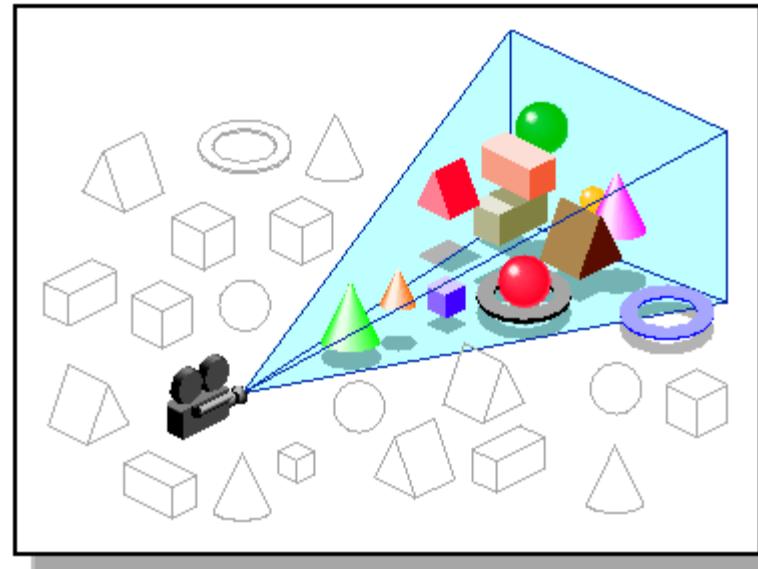
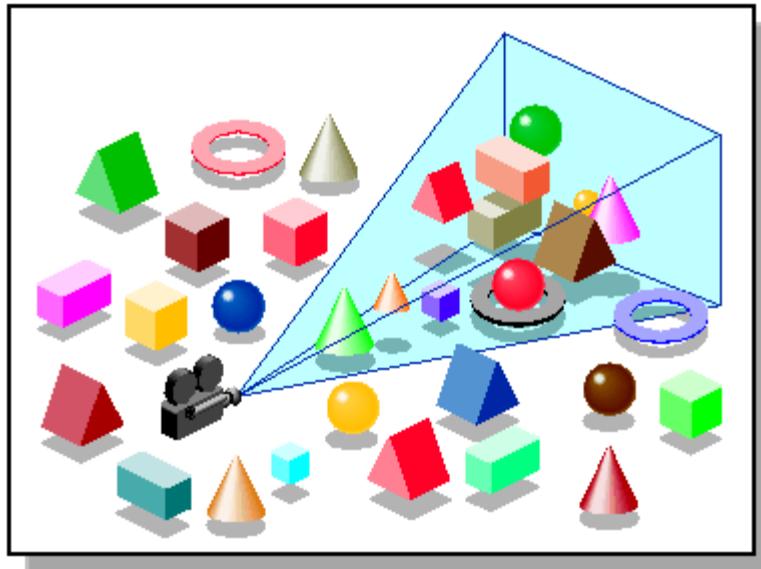
$$s'_A = s_A \frac{\text{Vol}(A)}{\text{Vol}(A \cap B)},$$

$$P[c(A \cap B) \geq x] = 1 - \sum_{t=0}^{x-1} \frac{\binom{s_A}{t} \binom{s - s_A}{s_B - t}}{\binom{s}{s_B}}.$$

Pravdepodobnosť, že v prieniku objemov je aspoň x dostatočne plných buniek

Frustum culling

- Nevykreslovanie neviditeľných objektov, ktoré su mimo pohľadového objemu
- Testuj postupne prienik ohraničujúcich objemov v hierarchii a pohľadového objemu

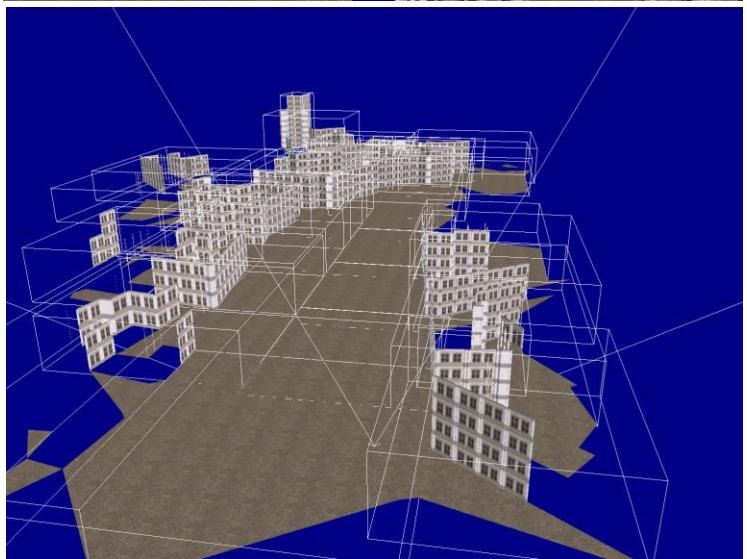
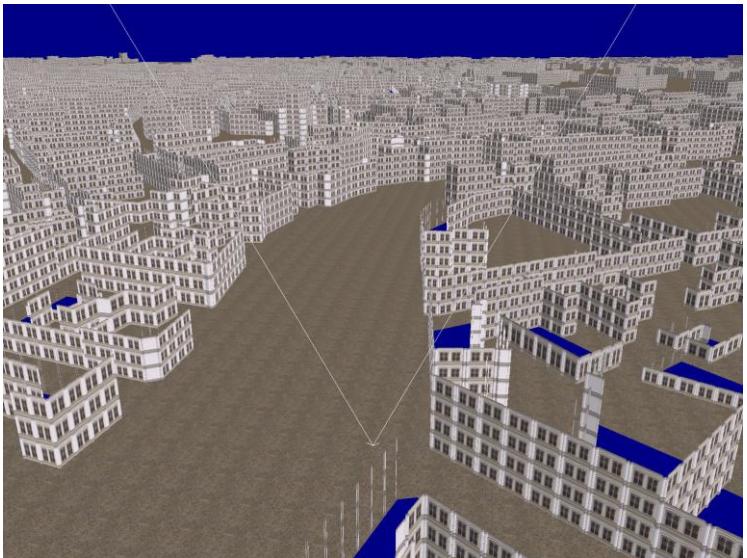


Zdroj: www.sgi.com

Occlusion culling

- Nevykreslovanie neviditeľných objektov
- Použitie rozšírenia, ktoré určí počet vykreslených pixelov daného objektu
- Postup:
 - Vykreslenie veľkých blízkych objektov
 - Vykreslenie ohr. objemov ostatných objektov, ak sa objem vykreslí do nenulového počtu pixelov, vykreslý sa aj celý objekt
 - Najlepšie postupujeme od predu dozadu (BSP)

Occlusion culling

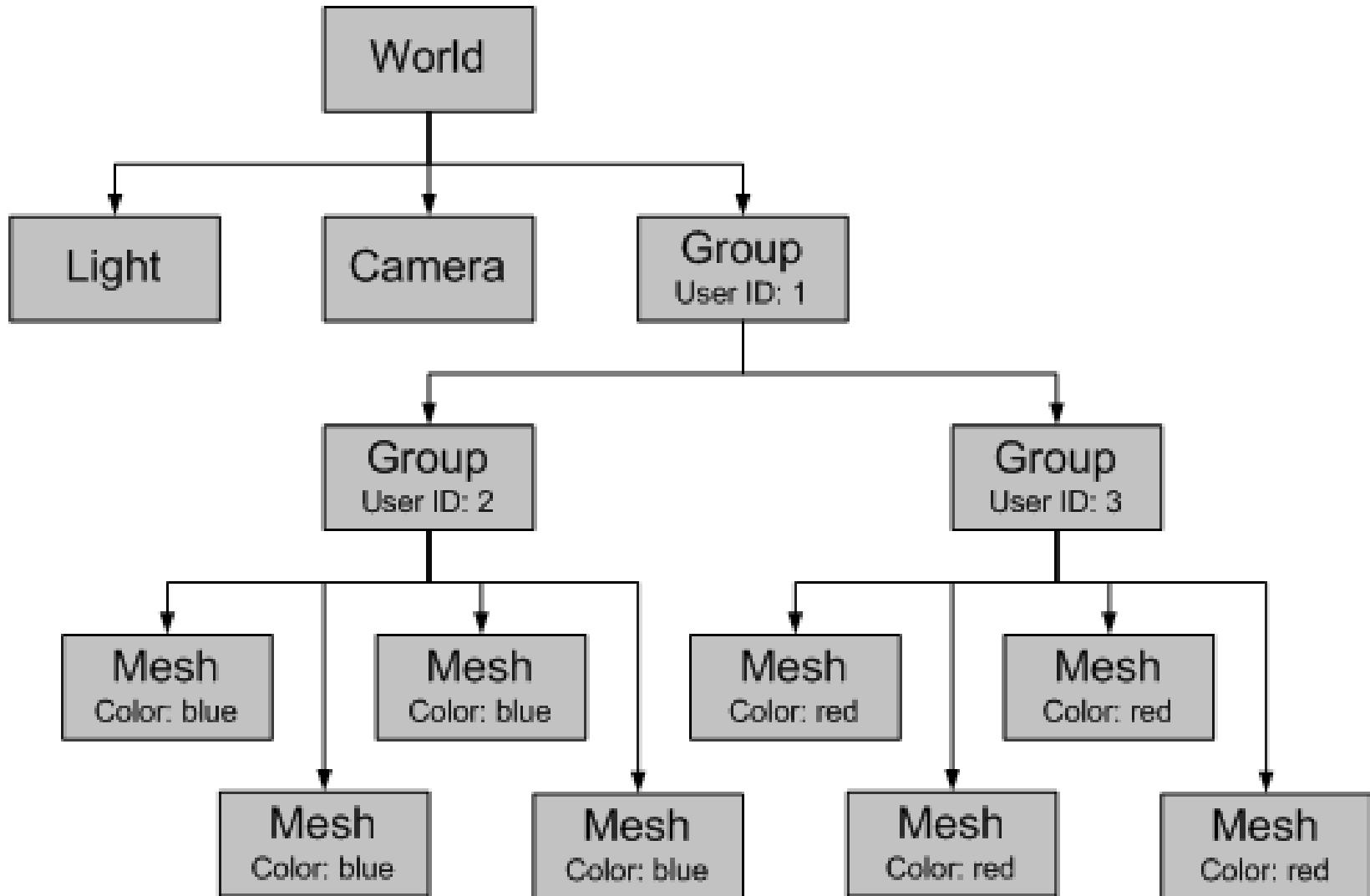


Source: <http://www.cg.tuwien.ac.at>

Graf scény

- Založený na hierarchiách v scéne
- Rýchlejší prístup k dátam
- Môže byť kombinovaný s ohraňujúcimi objemami
- Dynamické scény: bottom-up update
- Každý vrchol má mnoho atribútov:
 - lokálna, globálna transformácia
 - renderovacie atribúty
 - ohraňujúci objem

Graf scény





Otázky?