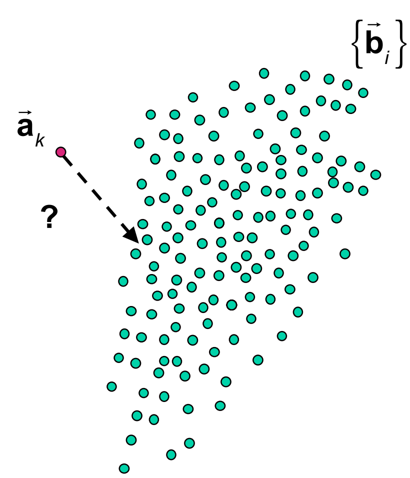
Finding point-pairs

- Given an **a**, find a corresponding **b** on the surface.
- Then one approach would be to search every possible triangle or surface point and then take the closest point.
- The key is to find a more efficient way to do this

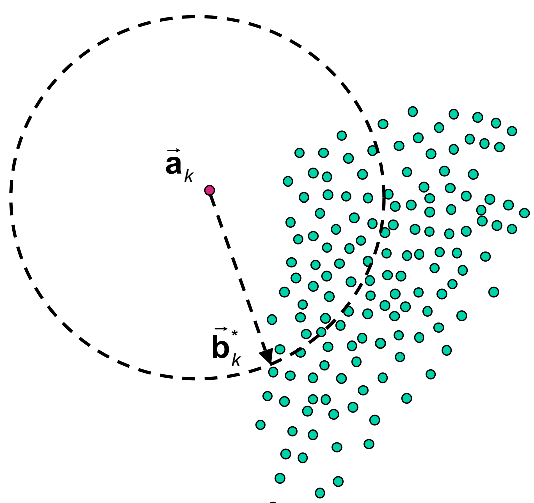
Suppose surface is represented by dense cloud of points



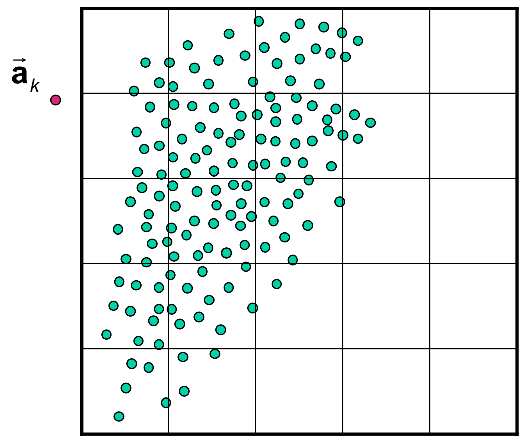
Find Closest Point from Dense Cloud

- Basic approach is to divide space into regions. Suppose that we have one point $\mathbf{b}_k$* that is a possible match for a point $\mathbf{a}_k$**.** The distance Δ*=|| $\mathbf{b}_k$* **-** $\mathbf{a}_k$|| obviously acts as an <u>upper</u> bound on the distance of the closest point to the surface.
- Given a region **R** containing many possible points $\mathbf{b}_j$, if we can compute a <u>lower</u> bound Δ_L on the distance from **a** to <u>any</u> point in **R,** then we need only consider points inside **R** if Δ_L < Δ*.

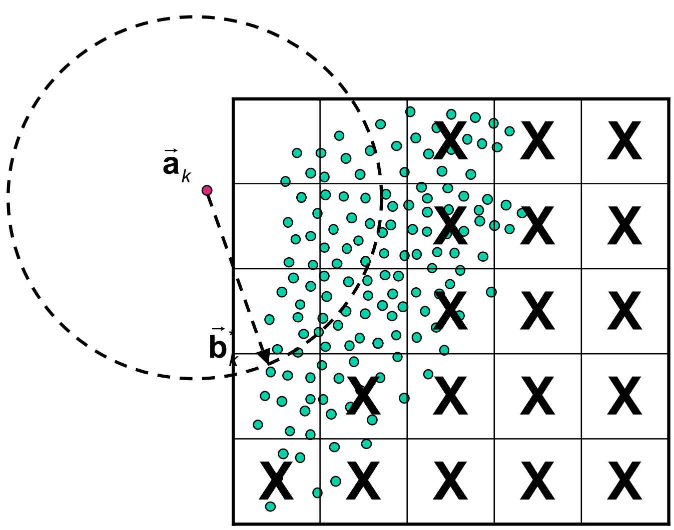
Given a match, is there anything closer?



Divide cloud into cells



Can exclude everything outside circle



Find Closest Point from Dense Cloud

- There are many ways to implement this idea
  - Simply partitioning space into many buckets
  - Octrees, KD trees, covariance trees, etc.

Approaches to closest triangle finding

1. (Simplest) Construct linear list of triangles and search sequentially for closest triangle to each point.
2. (Only slightly harder) Construct bounding spheres around each triangle and use these to reduce the number of careful checks required.
3. (Faster if have lots of points) Construct hierarchical data structure to speed search.
4. (Better but harder) Rotate each level of the tree to align with data.

Simple Search

// Triangle i has corners $[\vec{\mathbf{p}}_i, \vec{\mathbf{q}}_i, \vec{\mathbf{r}}_i]$

// Surrounding sphere i has radius ρ_i center $\vec{\mathbf{q}}_i$

$bound = \infty$;

for i=1 to N do

{ if $\left\| \vec{\mathbf{q}}_i - \vec{\mathbf{a}} \right\| - \rho_i \le bound$ then

{ $\vec{\mathbf{h}} = \text{FindClosestPoint}(\vec{\mathbf{a}}, [\vec{\mathbf{p}}_i, \vec{\mathbf{q}}_i, \vec{\mathbf{r}}_i])$;

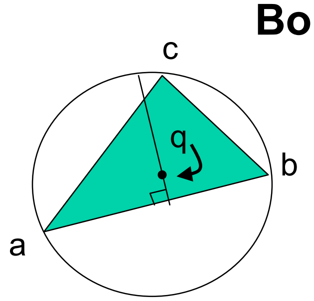
if $\left\| \vec{\mathbf{h}} - \vec{\mathbf{a}} \right\| < bound$ then

{ $\vec{\mathbf{c}} = \vec{\mathbf{h}}$; $bound = \left\| \vec{\mathbf{h}} - \vec{\mathbf{a}} \right\|$; };

};

};

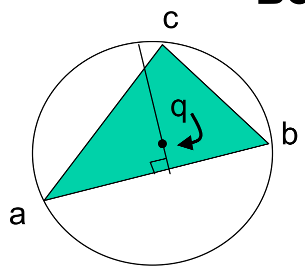
Bounding Sphere



Suppose you have a point $\vec{\mathbf{p}}$ and are trying to find the closest triangle $(\vec{\mathbf{a}}_k, \vec{\mathbf{b}}_k, \vec{\mathbf{c}}_k)$ to $\vec{\mathbf{p}}$. If you have already found a triangle $(\vec{\mathbf{a}}_j, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ with a point $\vec{\mathbf{r}}_j$ on it, when do you need to check carefully for some triangle k?

Answer: if $\vec{\mathbf{q}}_k$ is the center of a sphere of radius ρ_k enclosing $(\vec{\mathbf{a}}_k, \vec{\mathbf{b}}_k, \vec{\mathbf{c}}_k)$, then you only need to check carefully if $\left\| \vec{\mathbf{p}} - \vec{\mathbf{q}}_k \right\| - \rho_k < \left\| \vec{\mathbf{p}} - r_j \right\|$.

Bounding Sphere



Assume edge $(\vec{\mathbf{a}},\vec{\mathbf{b}})$ is the longest.

Then the center $\vec{\mathbf{q}}$ of the sphere will obey

$$\left(\vec{\mathbf{b}}-\vec{\mathbf{q}}\right)\cdot\left(\vec{\mathbf{b}}-\vec{\mathbf{q}}\right)=\left(\vec{\mathbf{a}}-\vec{\mathbf{q}}\right)\cdot\left(\vec{\mathbf{a}}-\vec{\mathbf{q}}\right)$$
$$\left(\vec{\mathbf{c}}-\vec{\mathbf{q}}\right)\cdot\left(\vec{\mathbf{c}}-\vec{\mathbf{q}}\right)\le\left(\vec{\mathbf{a}}-\vec{\mathbf{q}}\right)\cdot\left(\vec{\mathbf{a}}-\vec{\mathbf{q}}\right)$$
$$\left(\vec{\mathbf{b}}-\vec{\mathbf{a}}\right)\times\left(\vec{\mathbf{c}}-\vec{\mathbf{a}}\right)\cdot\left(\vec{\mathbf{q}}-\vec{\mathbf{a}}\right)=0$$

Simple approach: Try $\vec{\mathbf{q}}=\left(\vec{\mathbf{a}}+\vec{\mathbf{b}}\right)/\,2$.

If inequality holds, then done.

Else solve the system to get $\vec{\mathbf{q}}$ (next page).

The radius $\rho=\left\|\vec{\mathbf{q}}-\vec{\mathbf{a}}\right\|$.

Bounding Sphere

Assume edge $(\vec{\mathbf{a}},\vec{\mathbf{b}})$ is the longest side of triangle.

Compute $\vec{\mathbf{f}}=\left(\vec{\mathbf{a}}+\vec{\mathbf{b}}\right)/\,2$.

Define

$$\vec{\mathbf{u}}=\vec{\mathbf{a}}-\vec{\mathbf{f}};\vec{\mathbf{v}}=\vec{\mathbf{c}}-\vec{\mathbf{f}}$$
$$\vec{\mathbf{d}}=(\vec{\mathbf{u}}\times\vec{\mathbf{v}})\times\vec{\mathbf{u}}$$

Then the sphere center $\vec{\mathbf{q}}$ lies somewhere along the line

$$\vec{\mathbf{q}}=\vec{\mathbf{f}}+\lambda\vec{\mathbf{d}}$$

with $(\lambda\vec{\mathbf{d}}-\vec{\mathbf{v}})^2\le(\lambda\vec{\mathbf{d}}-\vec{\mathbf{u}})^2$. Simplifying gives us

$$\lambda\ge\frac{\vec{\mathbf{u}}^2-\vec{\mathbf{v}}^2}{2\vec{\mathbf{d}}\bullet\left(\vec{\mathbf{v}}-\vec{\mathbf{u}}\right)}=\gamma$$

If $\gamma\le 0$, then just pick λ=0. Else pick λ=γ.

FindClosestPoint(a,[p,q,r])

Many approaches. One is to solve the system

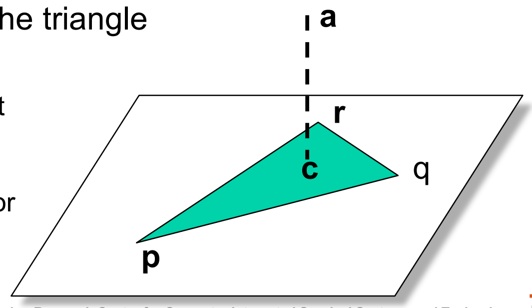
$$\mathbf{a}-\mathbf{p} \approx \lambda(\mathbf{q}-\mathbf{p})+\mu(\mathbf{r}-\mathbf{p})$$

in a least squares sense for λ and μ. Then compute

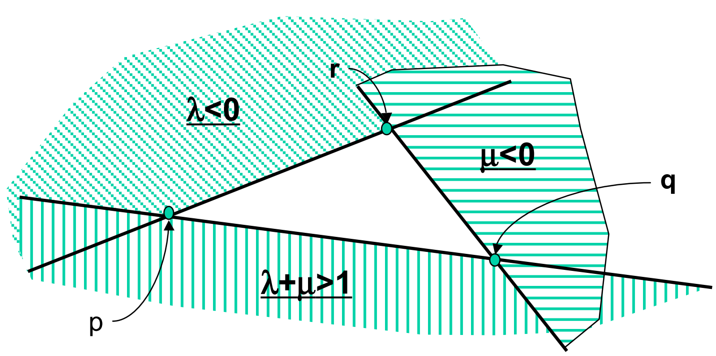
$$\mathbf{c} = \mathbf{p}+\lambda(\mathbf{q}-\mathbf{p})+\mu(\mathbf{r}-\mathbf{p})$$

If $\lambda \geq 0, \mu \geq 0, \lambda+\mu \leq 1$, then **c** lies within the triangle and is the closest point. Otherwise, you need to find a point on the border of the triangle

Hint: For efficiency, work out the least squares problem explicitly. You will have to solve a 2 x 2 linear system for λ, μ

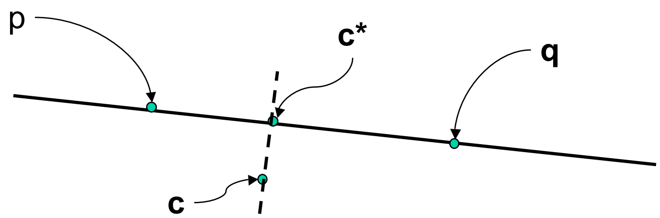


Finding closest point on triangle



Region	Closest point
$\lambda<0$	*ProjectOnSegment(***c,r,p***)*
$\mu<0$	*ProjectOnSegment(***c,p,q***)*
$\lambda+\mu>1$	*ProjectOnSegment(***c,q,r***)*

ProjectOnSegment(**c**,**p**,**q**)

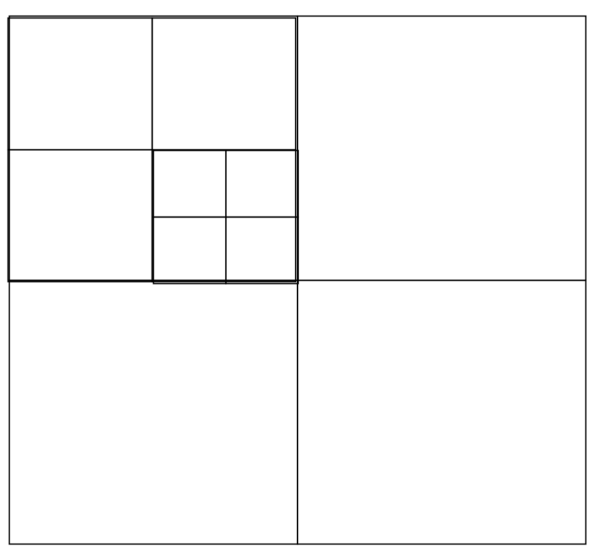


$$\lambda = \frac{(\mathbf{c}-\mathbf{p})\bullet(\mathbf{q}-\mathbf{p})}{(\mathbf{q}-\mathbf{p})\bullet(\mathbf{q}-\mathbf{p})}$$

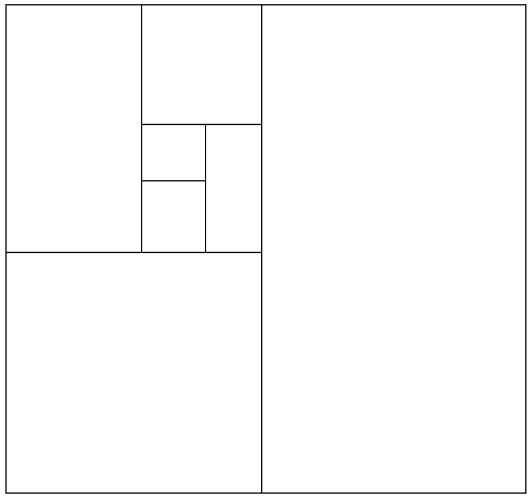
$$\lambda^* = Max(0, Min(\lambda, 1))$$

$$\mathbf{c}^* = \mathbf{p} + \lambda^* (\mathbf{q}-\mathbf{p})$$

Hierarchical cellular decompositions



Hierarchical cellular decompositions



Constructing tree of bounding spheres

```
class BoundingSphere {
  public:
    Vec3 Center;          // Coordinates of center
    double Radius;        // radius of sphere
    Thing* Object;        // some reference to the thing
                          // bounded
    };
```

Constructing octree of bounding spheres

```
class BoundingBoxTreeNode {
    Vec3 Center;        // splitting point
    Vec3 UB;            //  corners of box
    Vec3 LB;
    int HaveSubtrees;
    int nSpheres;
    double MaxRadius;          // maximum radius of sphere in box
    BoundingBoxTreeNode* SubTrees[2][2][2];
    BoundingSphere** Spheres;
        :
        :
    BoundingBoxTreeNode(BoundingSphere** BS, int nS);
    ConstructSubtrees();
    void FindClosestPoint(Vec3 v, double& bound, Vec3& closest);
    };
```


Constructing octree of bounding spheres

```
BoundingBoxTreeNode(BoundingSphere** BS, int nS)
{   Spheres = BS; nSpheres = nS;
    Center = Centroid(Spheres, nSpheres);
    MaxRadius = FindMaxRadius(Spheres,nSpheres);
    UB = FindMaxCoordinates(Spheres,nSpheres);
    LB = FindMinCoordinates(Spheres,nSpheres);
    ConstructSubtrees();
    };
```

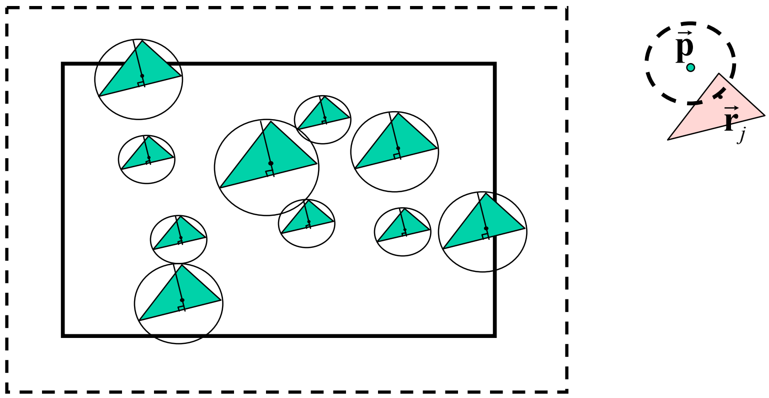
Constructing octree of bounding spheres

```
ConstructSubtrees()
{  if (nSpheres<= minCount || length(UB-LB)<=minDiag)
      { HaveSubtrees=0; return; };
  HaveSubtrees = 1;
  int nnn,npn,npp,nnp,pnn,ppn,ppp,pnp;
        // number of spheres in each subtree
  SplitSort(Center, Spheres, nnn,npn,npp,nnp,pnn,ppn,ppp,pnp);
  Subtrees[0][0][0] = BoundingBoxTree(Spheres[0],nnn);
  Subtrees[0][1][0] = BoundingBoxTree(Spheres[nnn],npn);
  Subtrees[0][1][1] = BoundingBoxTree(Spheres[nnn+npn],npp);
                  :
                  :
  }
```

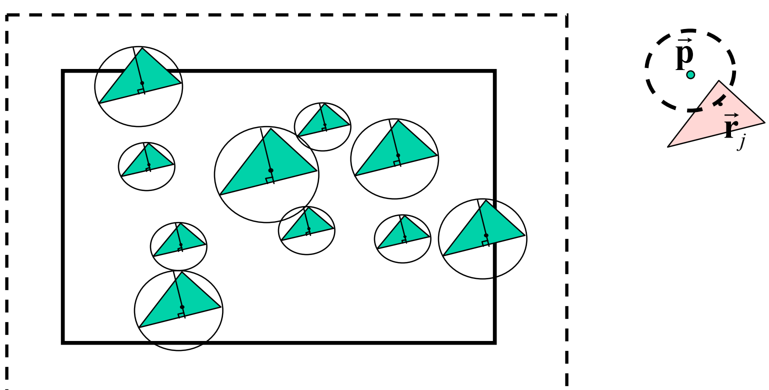

Constructing octree of bounding spheres

```
SplitSort(Vec3 SplittingPoint, BoundingSphere** Spheres,
            int& nnn, int& npn, … ,int& pnp)
{  // reorder Spheres(…) into eight buckets according to
   // comparison of coordinates of Sphere(k)->Center
   // with coordinates of splitting point.  E.g., first bucket has
   //       Sphere(k)->Center.x < SplittingPoint.x
   //       Sphere(k)->Center.y < SplittingPoint.y
   //       Sphere(k)->Center.z < SplittingPoint.z
   // This can be done "in place" by suitable exchanges.
   // Set nnn = number of spheres with all coordinates less than
   // splitting point, etc.
}
```

Searching an octree of bounding spheres



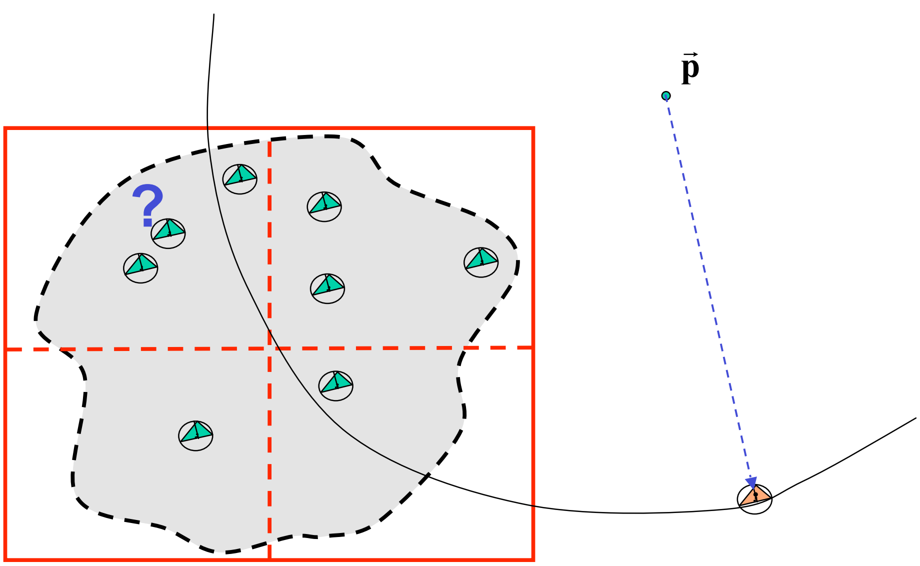
Searching an octree of bounding spheres

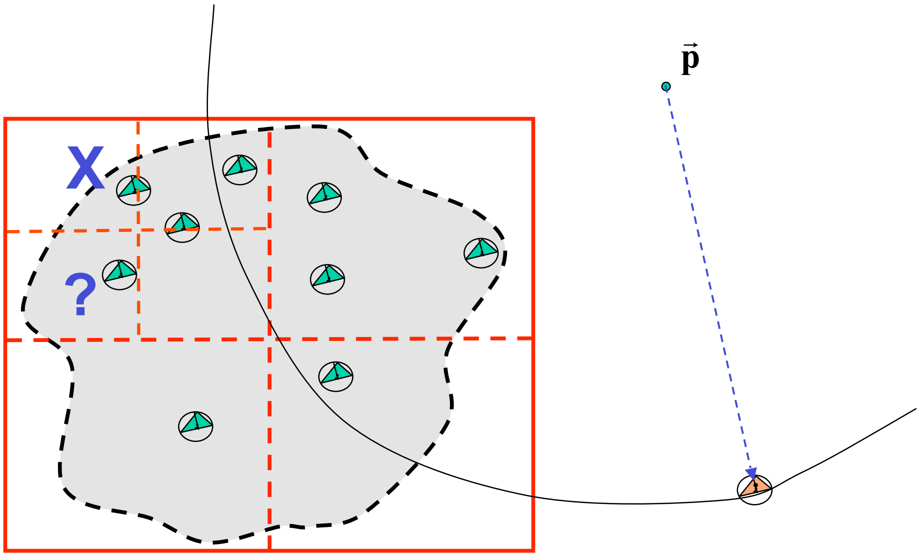


Searching an octree of bounding spheres

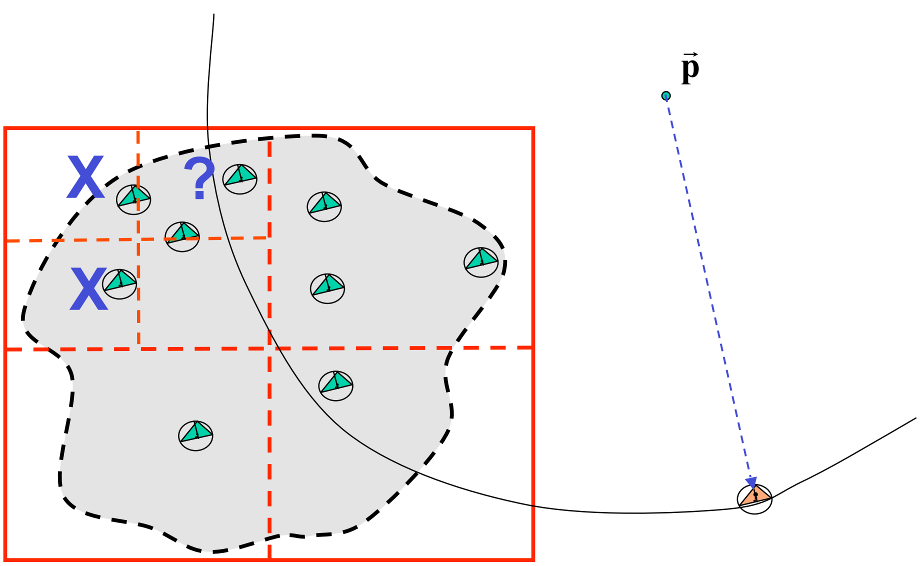
```
void BoundingBoxTreeNode::FindClosestPoint
                          (Vec3 v, double& bound, Vec3& closest)
{  double dist = bound + MaxRadius;
   if (v.x > UB.x+dist) return;  if (v.y > UB.y+dist) return;
      …. ;  if (v.z < LB.z-dist) return;
   if (HaveSubtrees)
      { Subtrees[0][0][0].FindClosestPoint(v,bound,closest);
           :
        Subtrees[1][1][1].FindClosestPoint(v,bound,closest);
      }
  else
    for (int i=0;i<nSpheres;I++)
        UpdateClosest(Spheres[i],v,bound,closest);

};
```

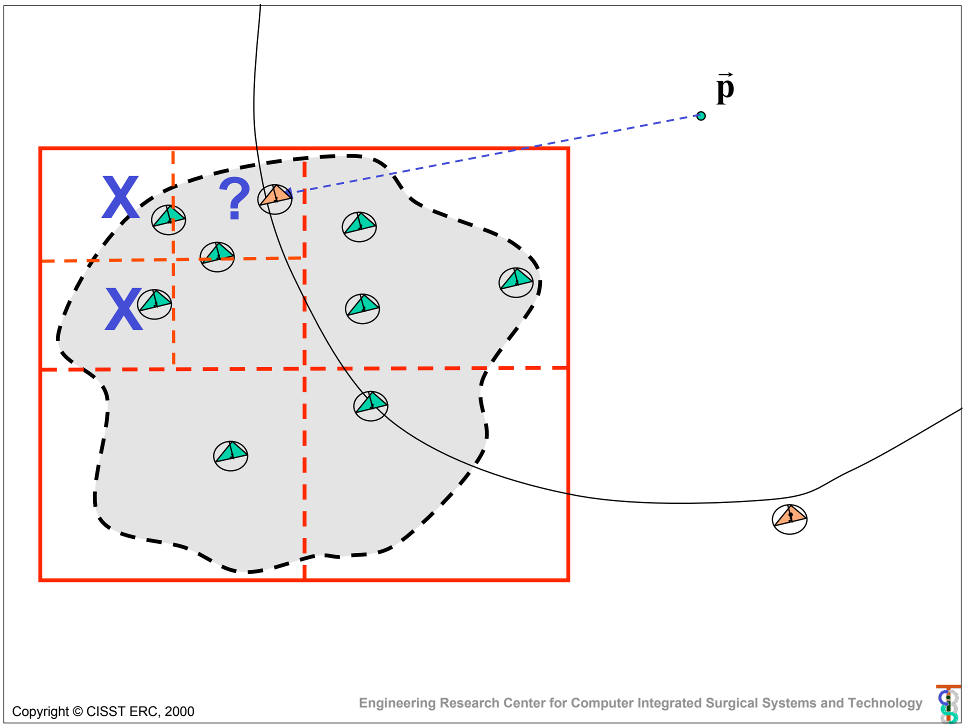



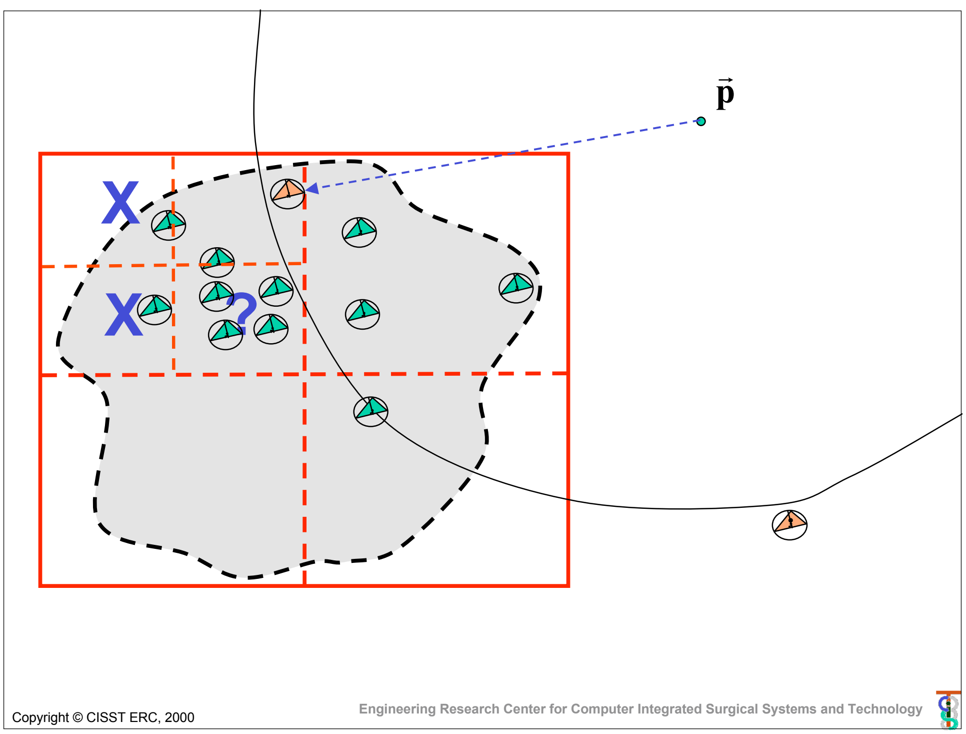


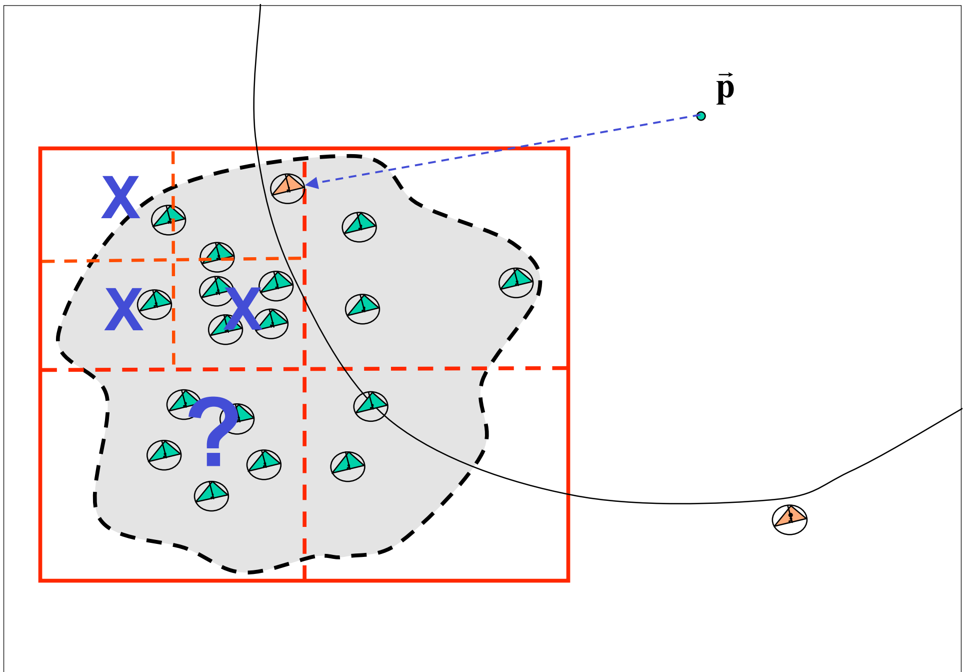
$\vec{p}$
X
?



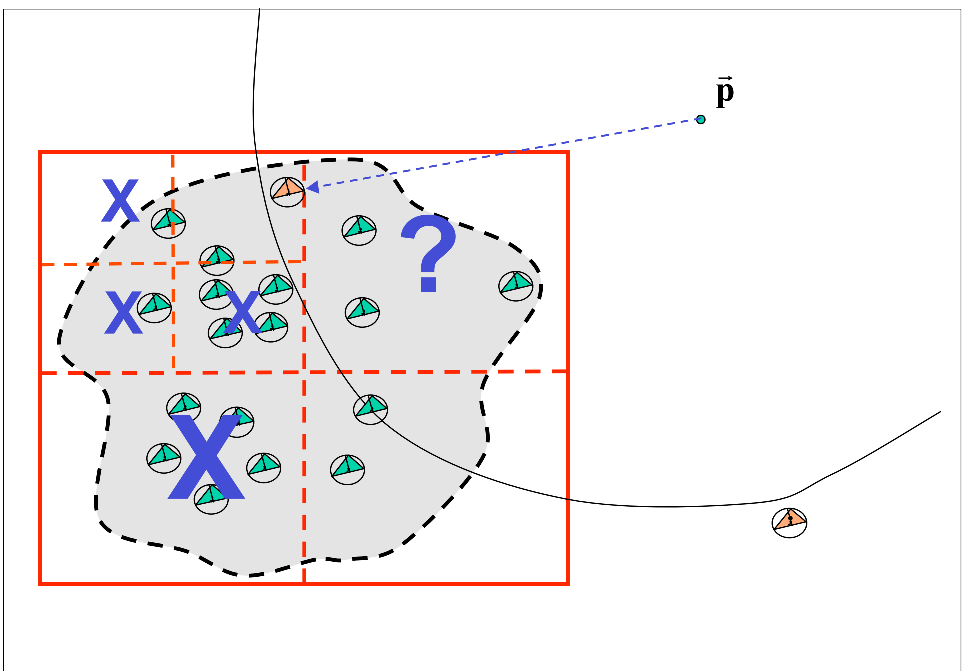
$\vec{p}$
X
?
X


$\vec{p}$
X
?
X
Copyright © CISST ERC, 2000
Engineering Research Center for Computer Integrated Surgical Systems and Technology

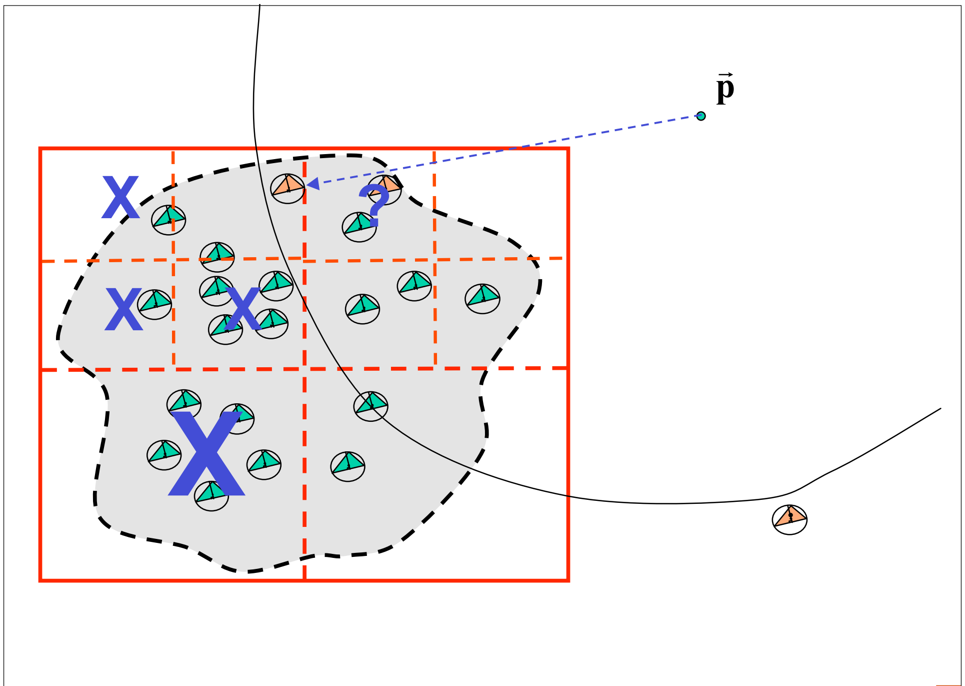

$\vec{p}$
X
X
?
Copyright © CISST ERC, 2000
Engineering Research Center for Computer Integrated Surgical Systems and Technology



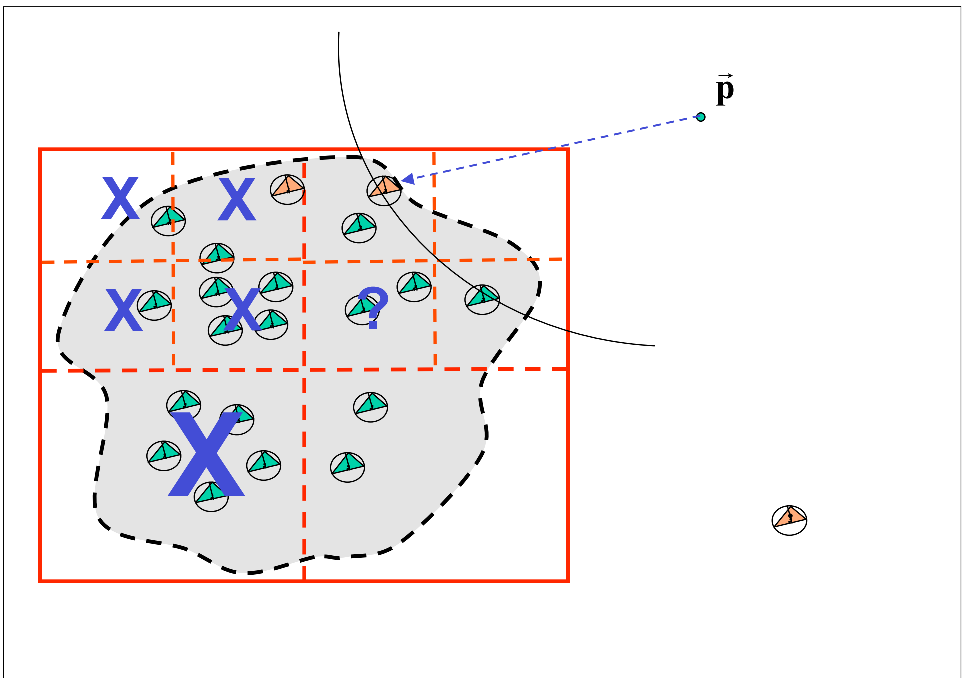
$\vec{p}$
X
X
X
?



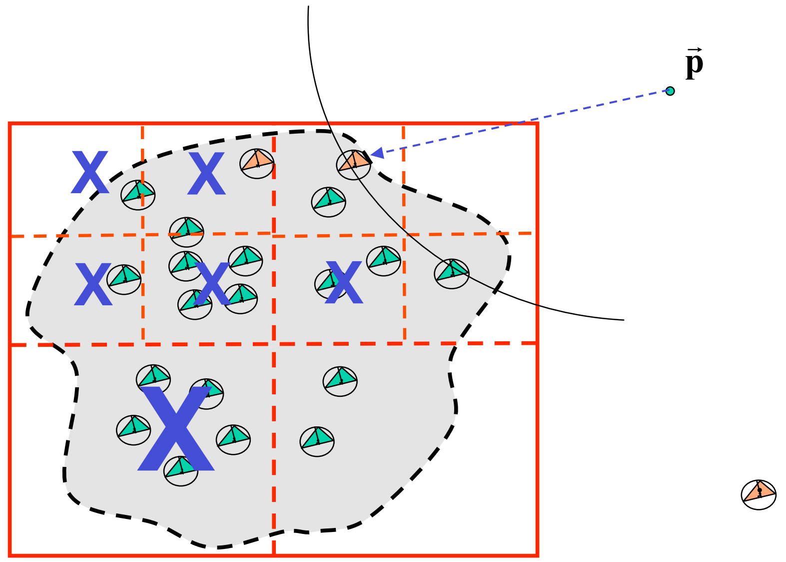
$\vec{p}$
X
?
X
X
X



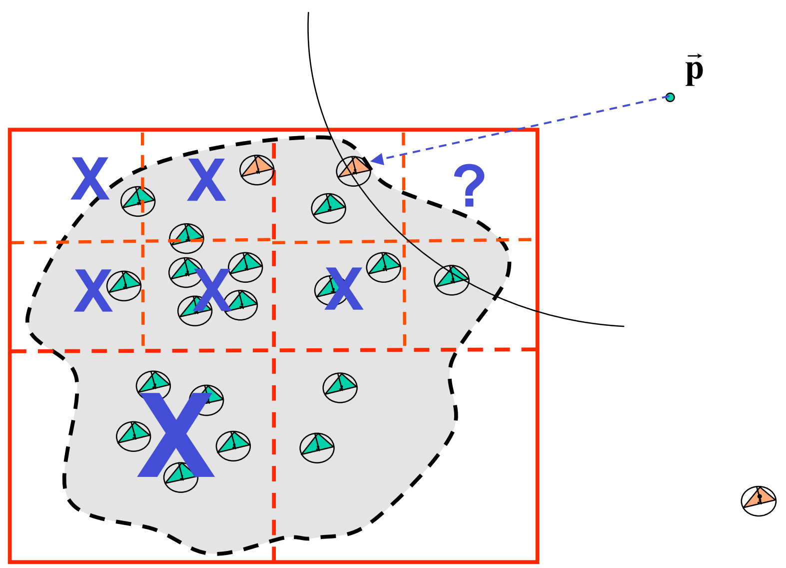
$\vec{p}$
X
X
X
?
X



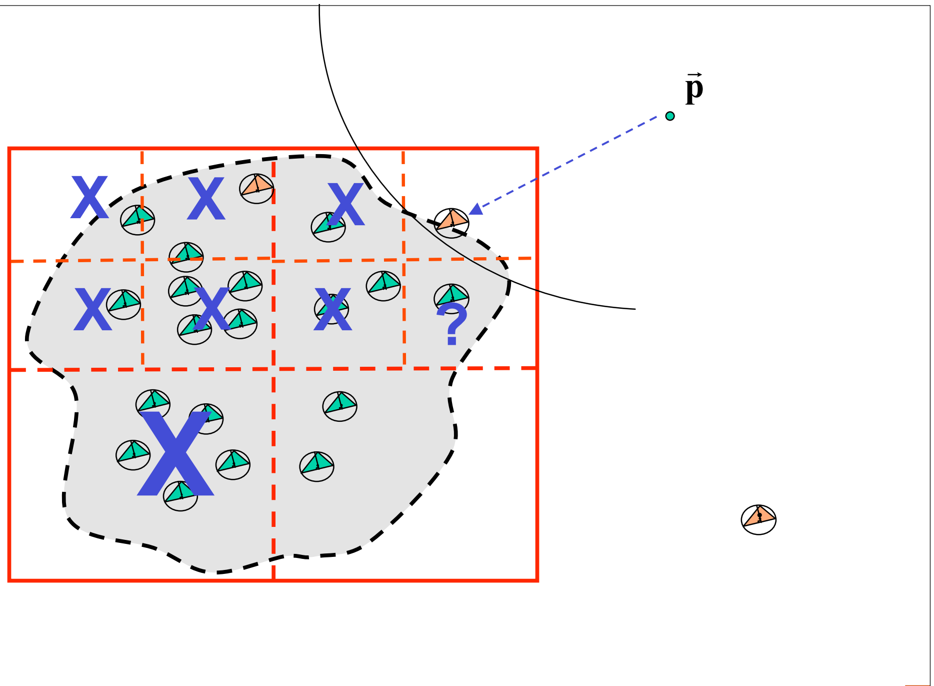
$\vec{p}$
X
X
X
X
?
X



$\vec{p}$
X
X
X
X
X
X

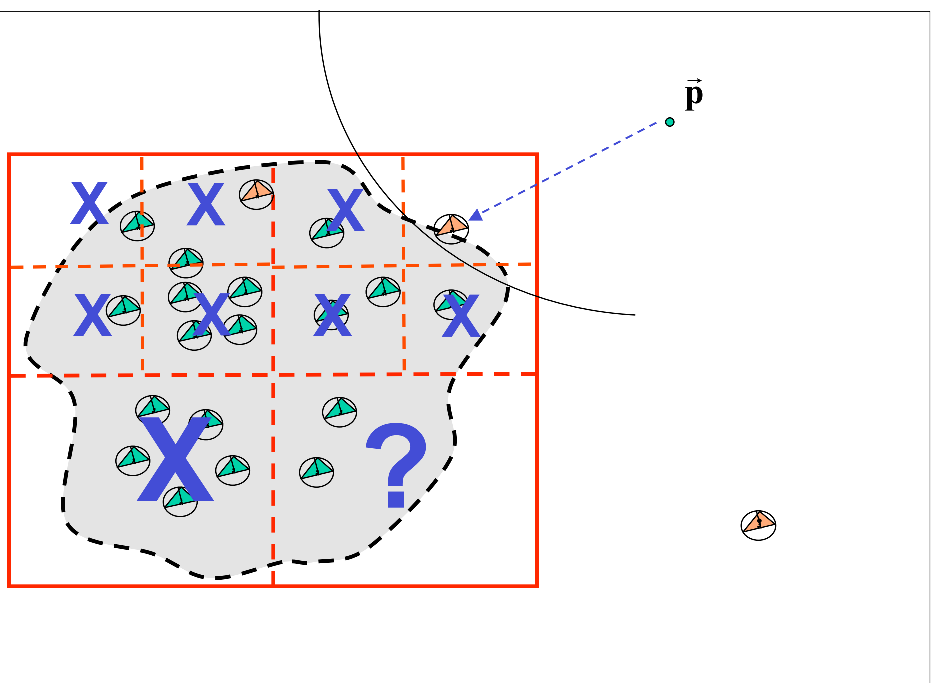


$\vec{p}$
X
X
?
X
X
X
X



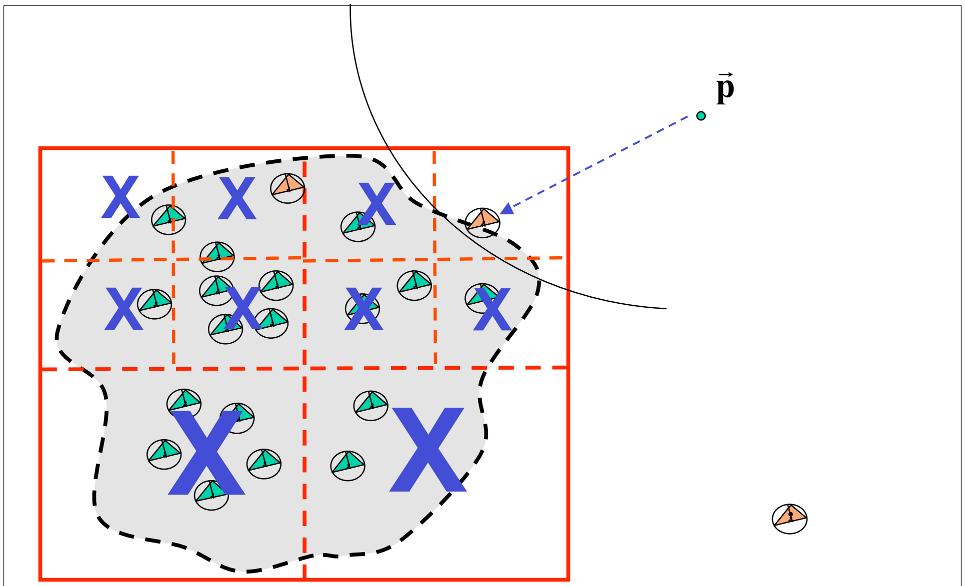
Copyright © CISST ERC, 2000

Engineering Research Center for Computer Integrated Surgical Systems and Technology



Copyright © CISST ERC, 2000

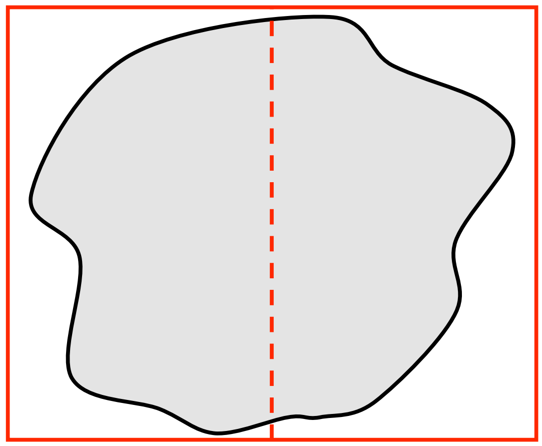
Engineering Research Center for Computer Integrated Surgical Systems and Technology



Searching an octree of bounding spheres

```
void UpdateClosest(BoundingSphere* S,
                   Vec3 v, double& bound, Vec3& closest)
{  double dist = v-S->Center;;
   if (dist - S->Radius > bound) return;
  Vec3 cp  = ClosestPointTo(*S->Object,v);
  dist = LengthOf(cp-v);
  if (dist<bound) { bound = dist; closest=cp;};
};
```

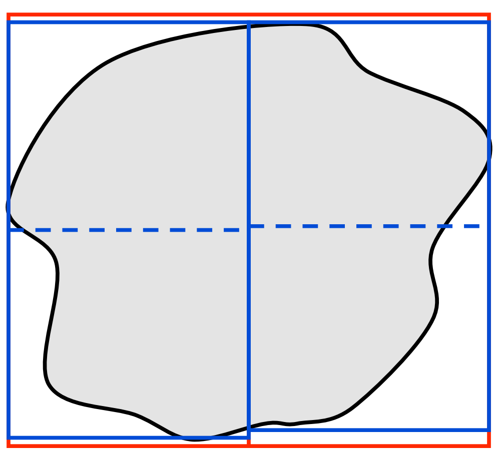
Covariance Trees



Copyright © CISST ERC, 2000

Engineering Research Center for Computer Integrated Surgical Systems and Technology

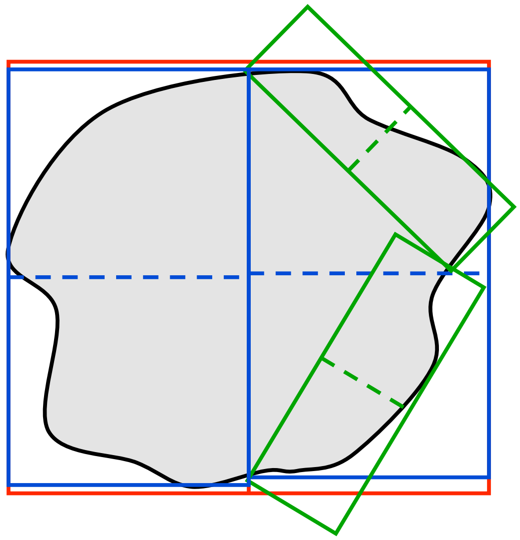
Covariance Trees



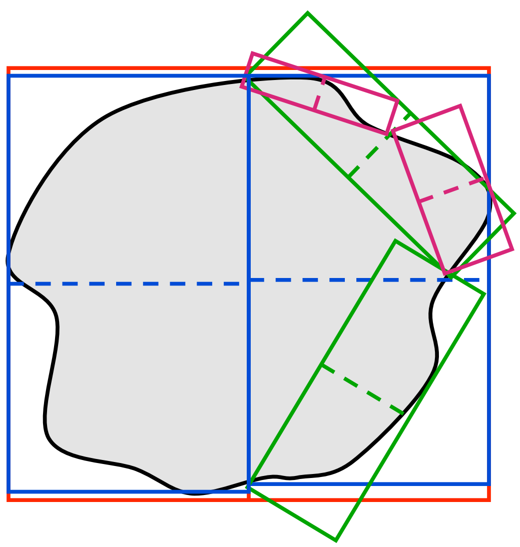
Copyright © CISST ERC, 2000

Engineering Research Center for Computer Integrated Surgical Systems and Technology

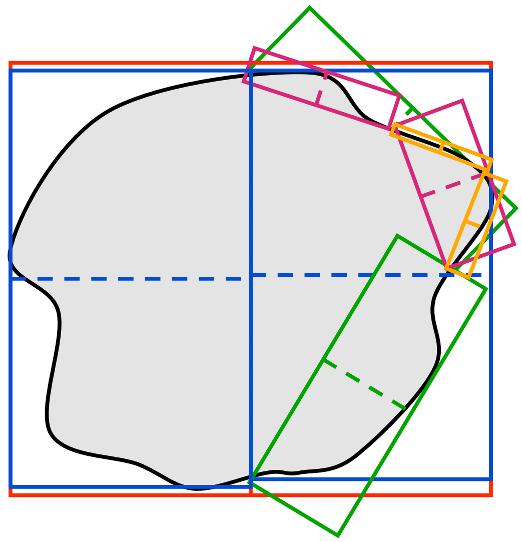
Covariance Trees



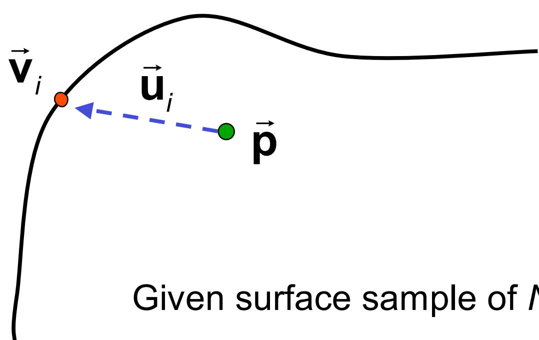
Covariance Trees



Covariance Trees



Covariance Tree Construction

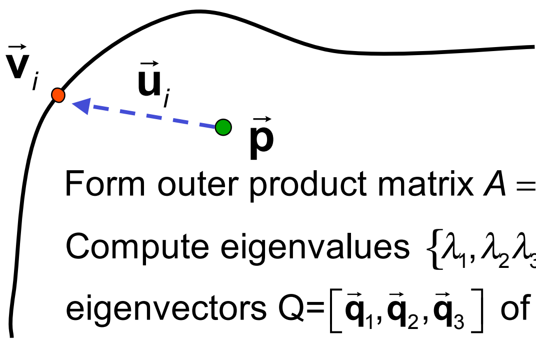


Given surface sample of N points $\{\vec{\mathbf{v}}_i\}$

Compute centroid $\vec{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N}\vec{\mathbf{v}}_i$

Compute residual vectors $\vec{\mathbf{u}}_i = \vec{\mathbf{v}}_i - \vec{\mathbf{p}}$

Covariance Tree Construction

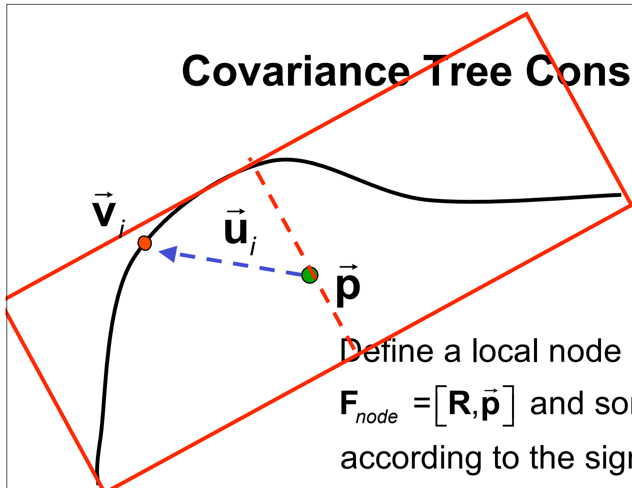


Form outer product matrix $A = \sum_i \vec{\mathbf{u}}_i \vec{\mathbf{u}}_i^T$

Compute eigenvalues $\{\lambda_1, \lambda_2 \lambda_3\}$ and eigenvectors Q=$\left[\vec{\mathbf{q}}_1, \vec{\mathbf{q}}_2, \vec{\mathbf{q}}_3\right]$ of A

Find a rotation **R** such that $\mathbf{R}_x$ is the eigenvector corresponding to the largest eigenvalue. (Depending on algorithm used, Q will be a rotation matrix, so all you may have to do is rotate Q)

Covariance Tree Construction



Define a local node coordinate system $\mathbf{F}_{node} = \left[\mathbf{R}, \vec{\mathbf{p}}\right]$ and sort the surface points according to the sign of the *x* component of $\vec{\mathbf{b}}_i = \mathbf{R}^{-1} \bullet \vec{\mathbf{u}}_i$. Compute bounding box

$$\vec{\mathbf{b}}^{\min} \leq \mathbf{R}^{-1} \bullet \vec{\mathbf{u}}_i \leq \vec{\mathbf{b}}^{\max}$$

Assign these points to "left" and "right" subtree nodes.

Covariance tree search

Given

- node with associated $\mathbf{F}_{node}$ and surface sample points $\vec{\mathbf{s}}_i$.
- sample point $\vec{\mathbf{a}}$, previous closest point $\vec{\mathbf{c}}$, $dist = \|\vec{\mathbf{a}} - \vec{\mathbf{c}}\|$

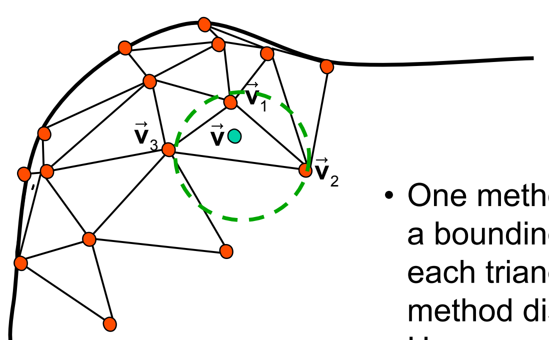
Transform $\vec{\mathbf{a}}$ into local coordinate system $\vec{\mathbf{b}} = \mathbf{F}_{node}^{-1}\vec{\mathbf{a}}$

Check to see if the point $\vec{\mathbf{b}}$ is inside an enlarged bounding box $\vec{\mathbf{b}}^{\min} - dist \le \vec{\mathbf{b}} \le \vec{\mathbf{b}}^{\max} + dist$. If not, then quit.

Otherwise, if no subnodes, do exhaustive search for closest.

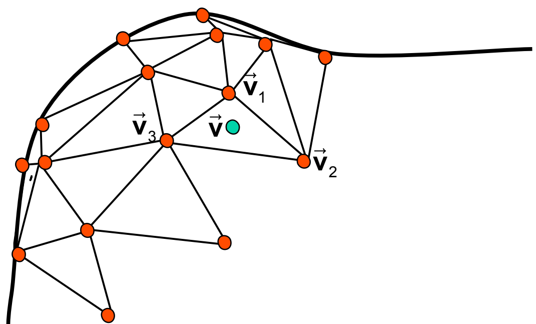
Otherwise, search left and right subtrees.

Covariance Trees for Triangle Meshes



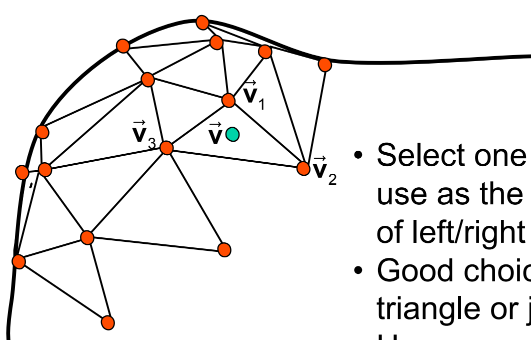
- One method is simply to place a bounding sphere around each triangle, and then use the method discussed previously
- However, this may be inconvenient if the mesh is deforming

Covariance Trees for Triangle Meshes



Observe that every point $\vec{\mathbf{v}}$ on a triangle $[\vec{\mathbf{v}}_1, \vec{\mathbf{v}}_2, \vec{\mathbf{v}}_3]$ can be expressed as a convex linear combination $\vec{\mathbf{v}} = \lambda_1\vec{\mathbf{v}}_1 + \lambda_2\vec{\mathbf{v}}_2 + \lambda_3\vec{\mathbf{v}}_3$ with $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Therefore, if $[\vec{\mathbf{v}}_1, \vec{\mathbf{v}}_2, \vec{\mathbf{v}}_3]$ are in some bounding box, then $\vec{\mathbf{v}}$ will also be in that bounding box

Covariance Trees for Triangle Meshes



- Select one point on the triangle to use as the “sort” point for selection of left/right subtrees.
- Good choices are centroid of triangle or just one of the vertices.
- However use <u>all</u> vertices of each triangle in determining the size of bounding boxes.
- Note this would work equally well for octrees.

Simple spatial sort