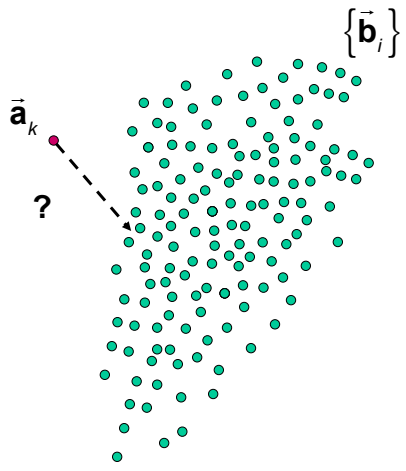
## Finding point-pairs

- Given an **a**, find a corresponding **b** on the surface.
- 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

Copyright Russell Taylor, 2010-2022


1

## Suppose surface is represented by dense cloud of points

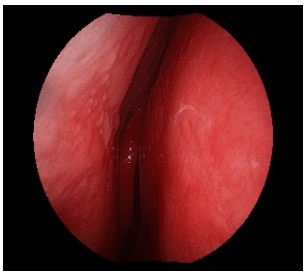



Copyright Russell Taylor, 2010-2022

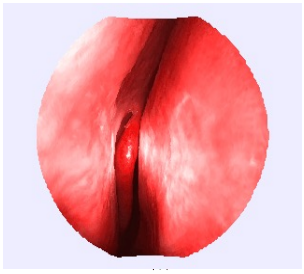

2

## 3D airway reconstruction during nasal endoscopic procedures without external tracking devices

Xingtong Liu



Monoscopic Endoscope Video



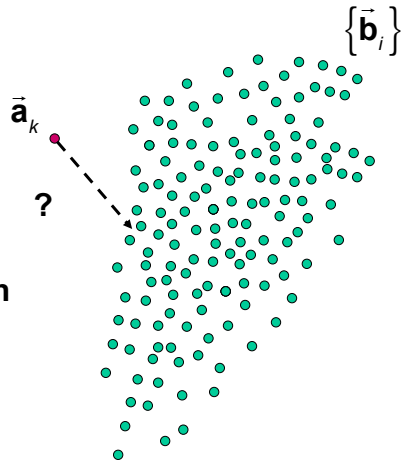
Dense Point Cloud Reconstruction

Copyright Russell Taylor, 2010-2022



3

## Suppose surface is represented by dense cloud of points




**How do we deal with the large number of possible matches?**

Copyright Russell Taylor, 2010-2022



4

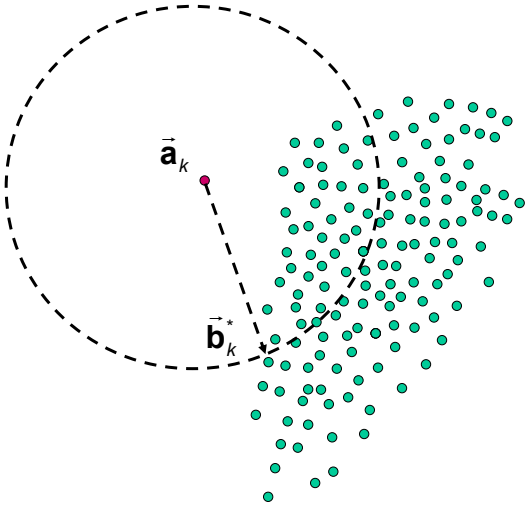
## Find Closest Point from Dense Cloud

- 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 $\Delta^* = \| \mathbf{b_k}^* - \mathbf{a_k} \|$ obviously acts as an upper 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 lower bound $\Delta_L$ on the distance from **a** to any point in **R,** then we need only consider points inside **R** if $\Delta_L < \Delta^*$.

Copyright Russell Taylor, 2010-2022


5

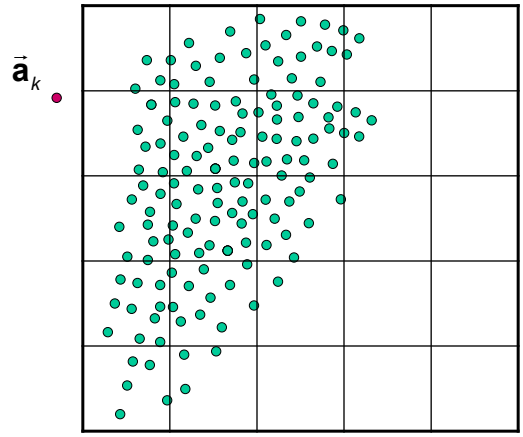
## Given a match, is there anything closer?




Copyright Russell Taylor, 2010-2022


6

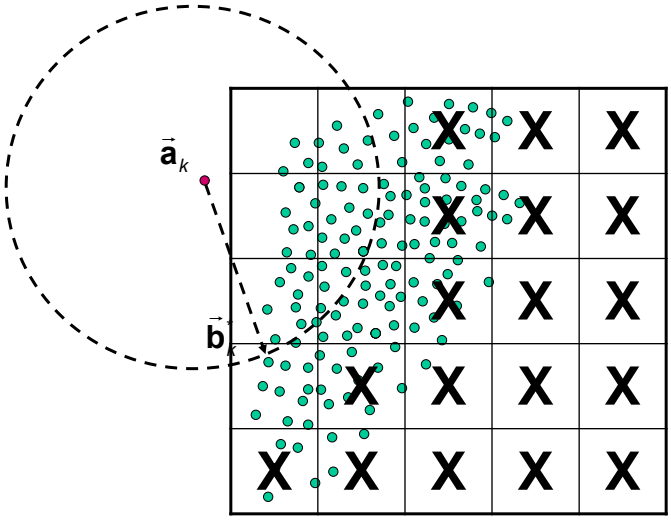
# Divide cloud into cells




Copyright Russell Taylor, 2010-2022


7

# Can exclude everything outside circle




Copyright Russell Taylor, 2010-2022


8

## 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.
- Basic idea also works with surface meshes, but need to find closest point on a triangle.

Copyright Russell Taylor, 2010-2022

9

## 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 or bounding boxes 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.

Copyright Russell Taylor, 2010-2022

10

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

Many approaches. One is to solve the system

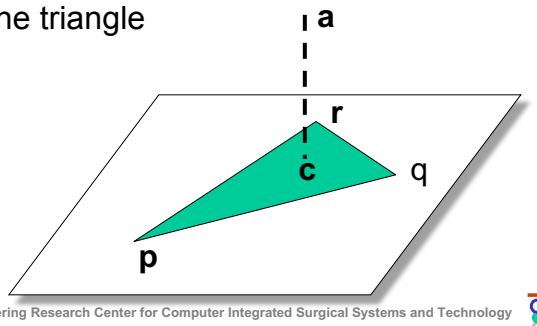
$$\mathbf{a}-\mathbf{p} \approx \lambda(\mathbf{q}-\mathbf{p})+\mu(\mathbf{r}-\mathbf{p})$$

in a least squares sense for $\lambda$ and $\mu$. 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

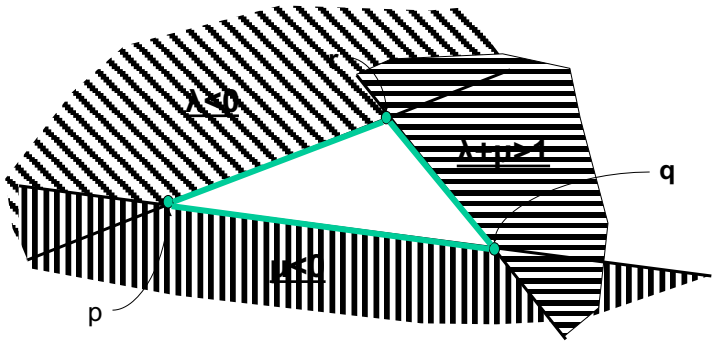
**Hint:** For efficiency, work out the least squares problem explicitly for λ, μ




Copyright Russell Taylor, 2010-2022


11

# Finding closest point on triangle

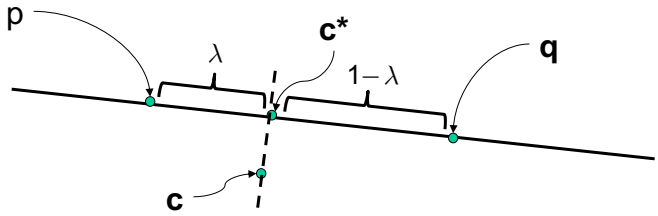



| Region | Closest point |
|---|---|
| **λ<0** | *ProjectOnSegment(***c,r,p***)* |
| **μ<0** | *ProjectOnSegment(***c,p,q***)* |
| **λ+μ>1** | *ProjectOnSegment(***c,q,r***)* |

Copyright Russell Taylor, 2010-2022


12

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




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

$$\lambda^{(seg)} = Max(0, Min(\lambda, 1))$$

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

Copyright Russell Taylor, 2010-2022


13

## FindClosestPoint(a,[p,q,r]) – Barycentric Form

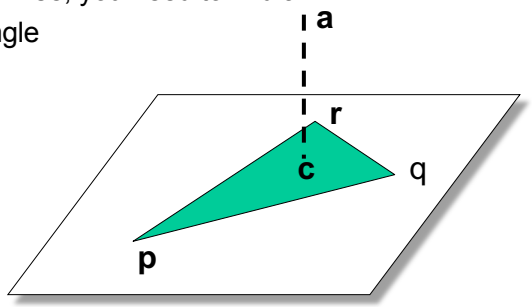
For a barycentric formulation, you can solve the system

$\mathbf{a} \approx \lambda\mathbf{q} + \mu\mathbf{r} + \nu\mathbf{p}$ such that $\lambda+\mu+\nu=1$

in a least squares sense for $\lambda$, $\mu$, and $\nu$. Then compute

$\mathbf{c} = \lambda\mathbf{q} + \mu\mathbf{r} + \nu\mathbf{p}$

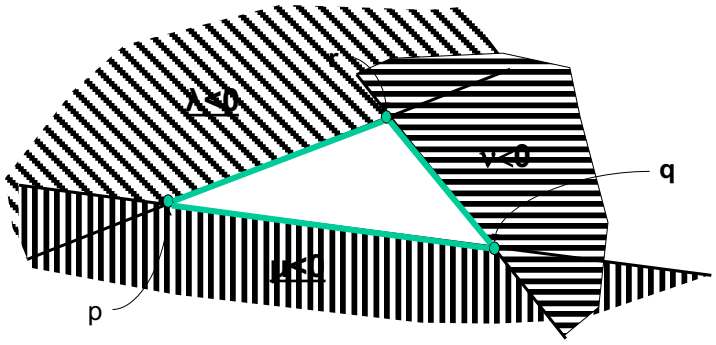
If $\lambda \geq 0, \mu \geq 0, \nu \geq 0$, then **c** lies within the triangle and is the closest point. Otherwise, you need to find a point on the border of the triangle




Copyright Russell Taylor, 2010-2022


14

## Finding closest point on triangle




| Region | Closest point |
|---|---|
| λ<0 | *ProjectOnSegment*(**c,r,p**) |
| μ<0 | *ProjectOnSegment*(**c,p,q**) |
| ν<0 | *ProjectOnSegment*(**c,q,r**) |

Copyright Russell Taylor, 2010-2022


15

## Simple Search with Bounding Boxes

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

// Bounding box lower = $\vec{L}_i = [L_{xi}, L_{yi}, L_{zi}]^T$; upper = $\vec{U}_i = [U_{xi}, U_{yi}, U_{zi}]^T$

$bound = \infty$

for $i = 1$ to $N$ do

{ if $\left(L_{xi} - bound \le a_x \le U_{xi} + bound\right)$ and $\left(L_{yi} - bound \le a_y \le U_{yi} + bound\right)$

and $\left(L_{zi} - bound \le a_z \le U_{zi} + bound\right)$ then

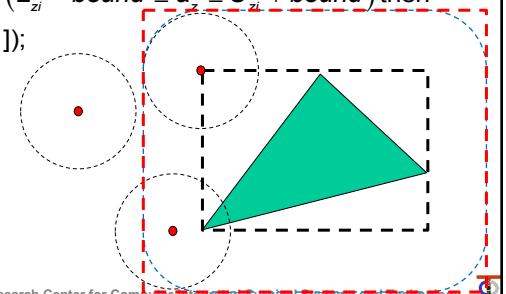
{ $\vec{\mathbf{h}} = \text{FindClosestPoint}(\vec{\mathbf{a}}, [\vec{\mathbf{p}}_i, \vec{\mathbf{q}}_i, \vec{\mathbf{r}}_i])$;

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

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

};

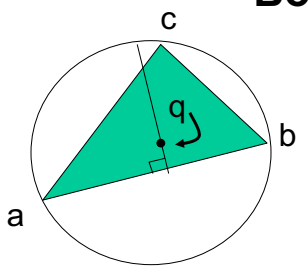
};



Copyright Russell Taylor, 2010-2022


16

# Bounding Sphere



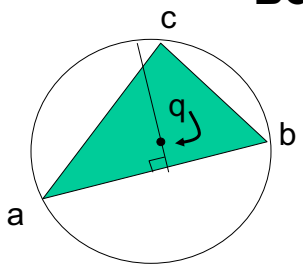

Suppose you have a point $\vec{\mathbf{p}}$ and are trying to find the closest triangle $(\vec{\mathbf{a}}_k, \vec{\mathbf{b}}_k, \vec{\mathbf{c}}_k)$ to $\vec{\mathbf{p}}$. If you have already found a triangle $(\vec{\mathbf{a}}_j, \vec{\mathbf{b}}_j, \vec{\mathbf{c}}_j)$ with a point $\vec{\mathbf{r}}_j$ 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 $\rho_k$ enclosing $(\vec{\mathbf{a}}_k, \vec{\mathbf{b}}_k, \vec{\mathbf{c}}_k)$, then you only need to check carefully if $\left\|\vec{\mathbf{p}} - \vec{\mathbf{q}}_k\right\| - \rho_k < \left\|\vec{\mathbf{p}} - r_j\right\|$.

Copyright Russell Taylor, 2010-2022

17

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

Copyright Russell Taylor, 2010-2022

18

## 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

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

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

Copyright Russell Taylor, 2010-2022

## Simple Search with Bounding Spheres

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

// Surrounding sphere i has radius $\rho_i$ and center $\vec{\mathbf{d}}_i$

*bound* $= \infty$;

for i=1 to N do

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

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

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

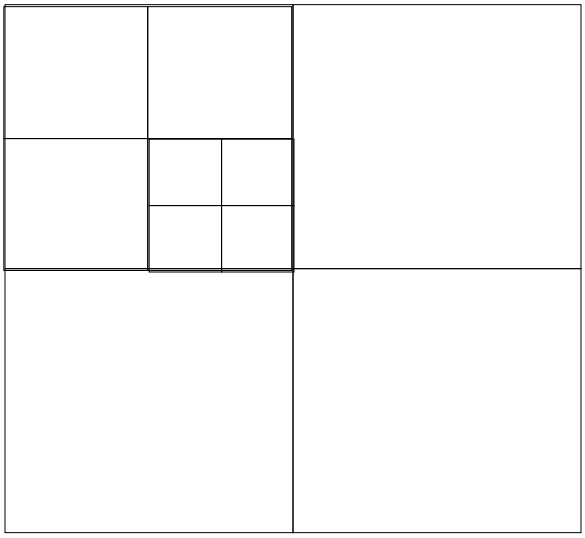
{ $\vec{\mathbf{c}} = \vec{\mathbf{h}}$; *bound* $= \left|\left|\vec{\mathbf{h}} - \vec{\mathbf{a}}\right|\right|$;};

};

};

Copyright Russell Taylor, 2010-2022

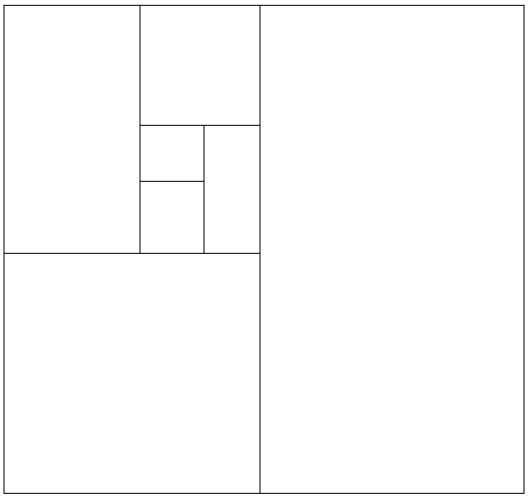
# Hierarchical cellular decompositions



Copyright Russell Taylor, 2010-2022


21

# Hierarchical cellular decompositions



Copyright Russell Taylor, 2010-2022


22

# 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
    };
```

Copyright Russell Taylor, 2010-2022

23

# 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);
   };
```

Copyright Russell Taylor, 2010-2022

24

# Constructing octree of bounding spheres

```
BoundingBoxTreeNode(BoundingSphere** BS, int nS)
{   Spheres = BS; nSpheres = nS;
    Center = Centroid(Spheres, nSpheres);
            // This will be the splitting point
            //  Centroid is efficient to compute
            //   But other choices are possible
    MaxRadius = FindMaxRadius(Spheres,nSpheres);
    UB = FindMaxCoordinates(Spheres,nSpheres);
    LB = FindMinCoordinates(Spheres,nSpheres);
    ConstructSubtrees();
    };
```

Copyright Russell Taylor, 2010-2022

25

# 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);
                     :
                     :
   }
```

Copyright Russell Taylor, 2010-2022

26

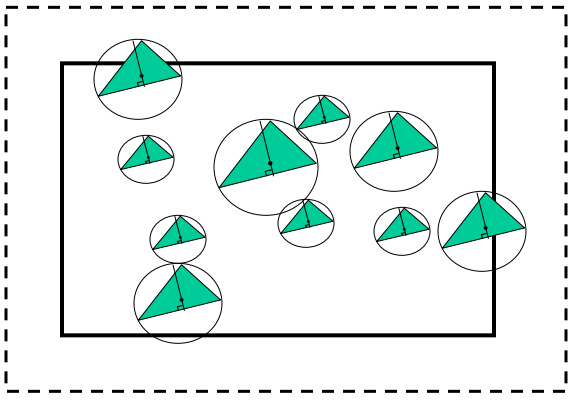
# Constructing octree of bounding spheres

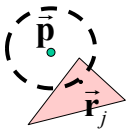
```
SplitSort(Vec3 SplittingPoint, BoundingSphere** Spheres,
          int& nnn, int& npn, … ,int& pnp)
{  // reorder Spheres(…) into eight buckets according to
   // comparison of coordinates of Sphere(k)->Center
   // with coordinates of splitting point.  E.g., first bucket has
   //       Sphere(k)->Center.x < 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.
}
```

Copyright Russell Taylor, 2010-2022


27

# Searching an octree of bounding spheres

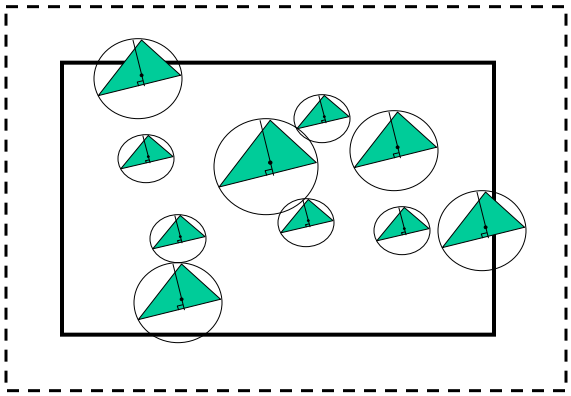


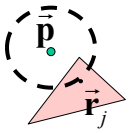



Copyright Russell Taylor, 2010-2022


28

# Searching an octree of bounding spheres






Copyright Russell Taylor, 2010-2022


29

# 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);

};
```

Copyright Russell Taylor, 2010-2022

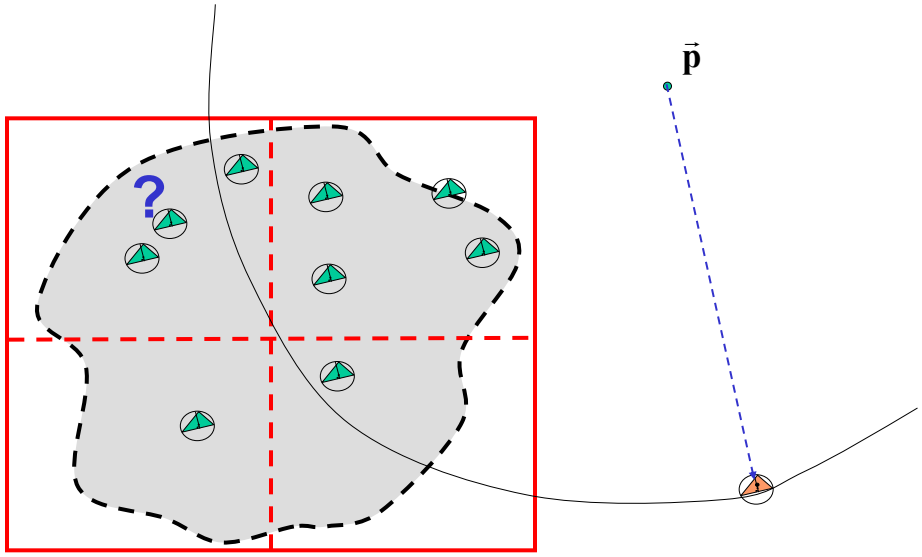

30

# 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;};
};
```

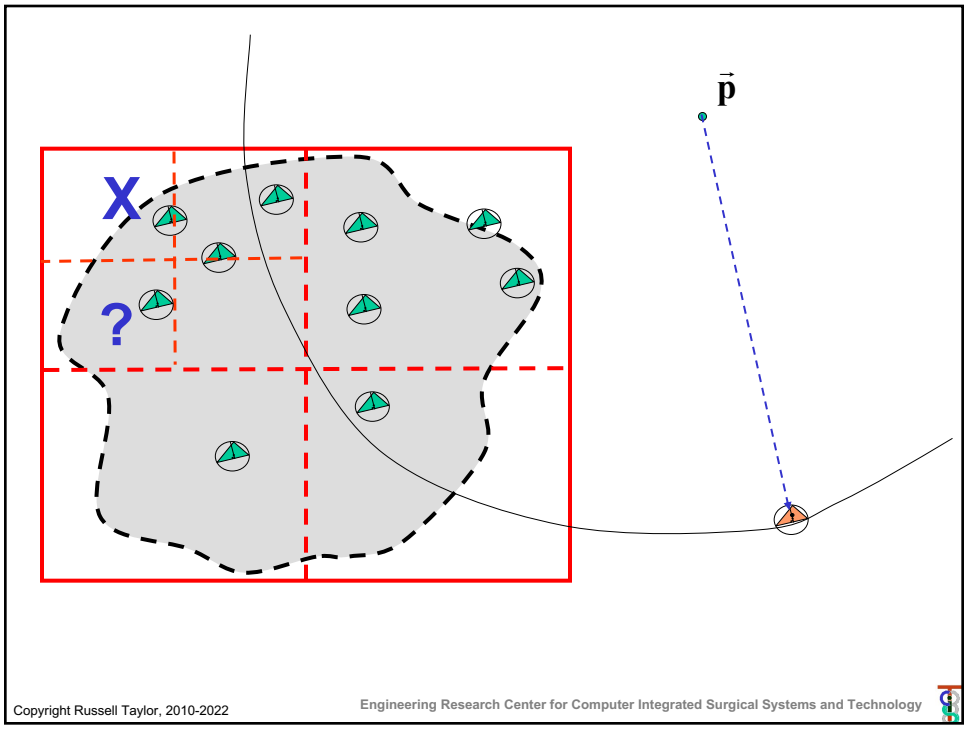
Copyright Russell Taylor, 2010-2022


31



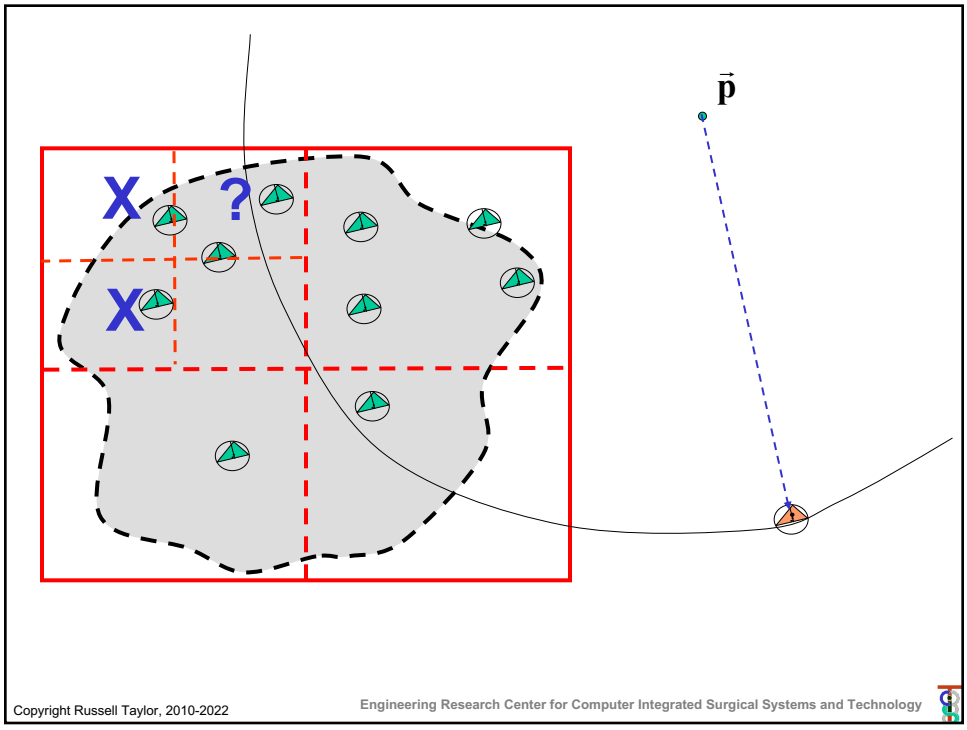

Copyright Russell Taylor, 2010-2022


32



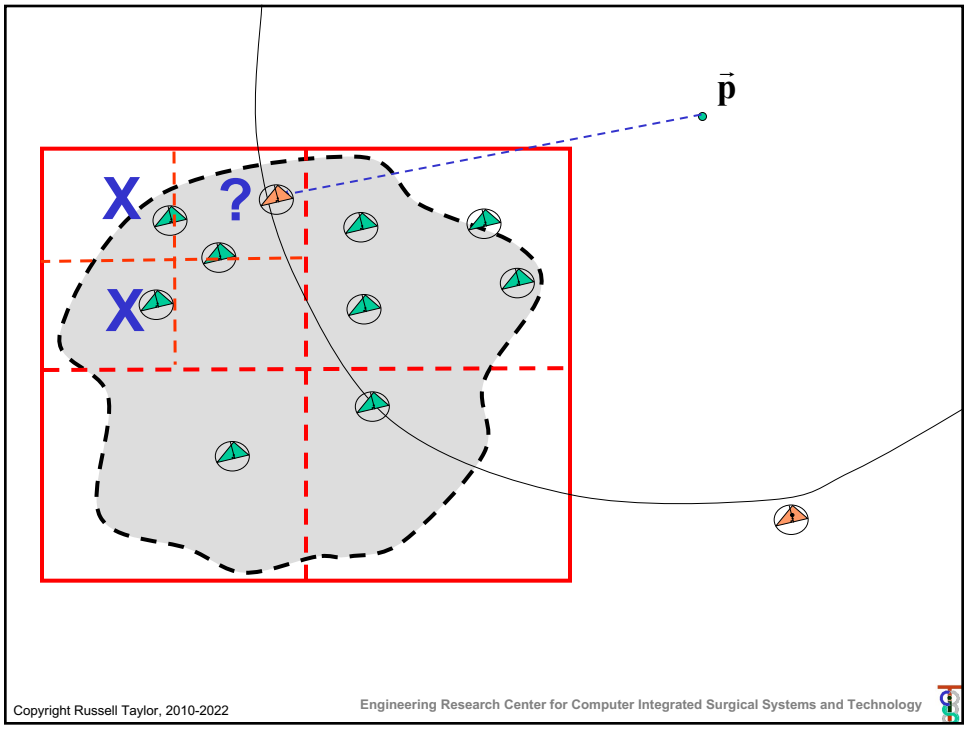

Copyright Russell Taylor, 2010-2022


33



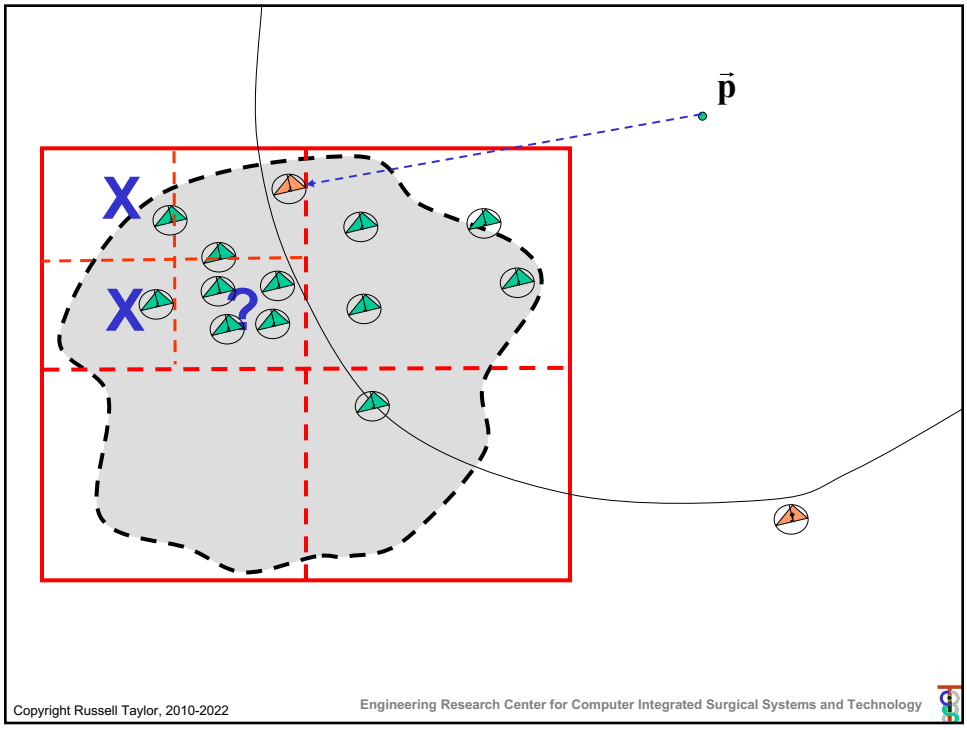

Copyright Russell Taylor, 2010-2022


34



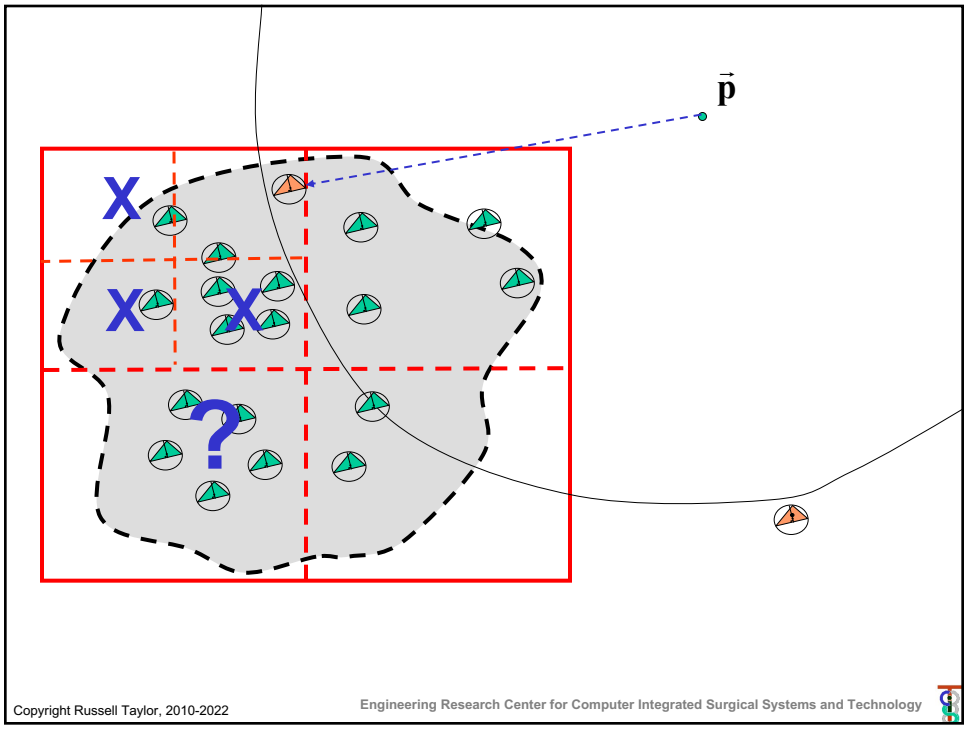

Copyright Russell Taylor, 2010-2022


35



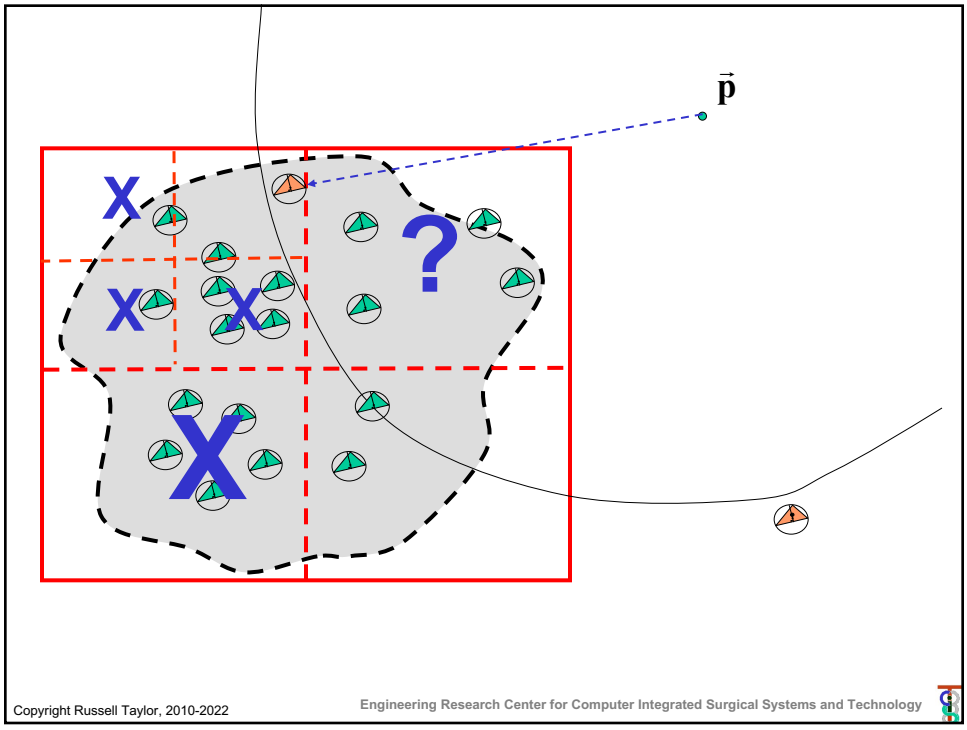

Copyright Russell Taylor, 2010-2022


36



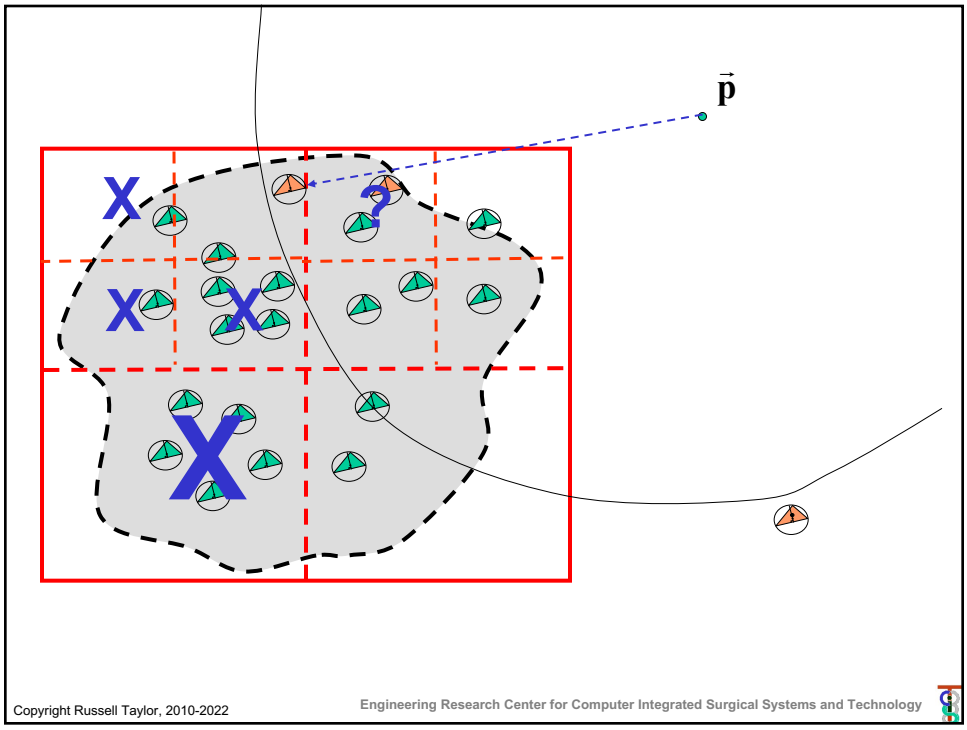
Copyright Russell Taylor, 2010-2022


37



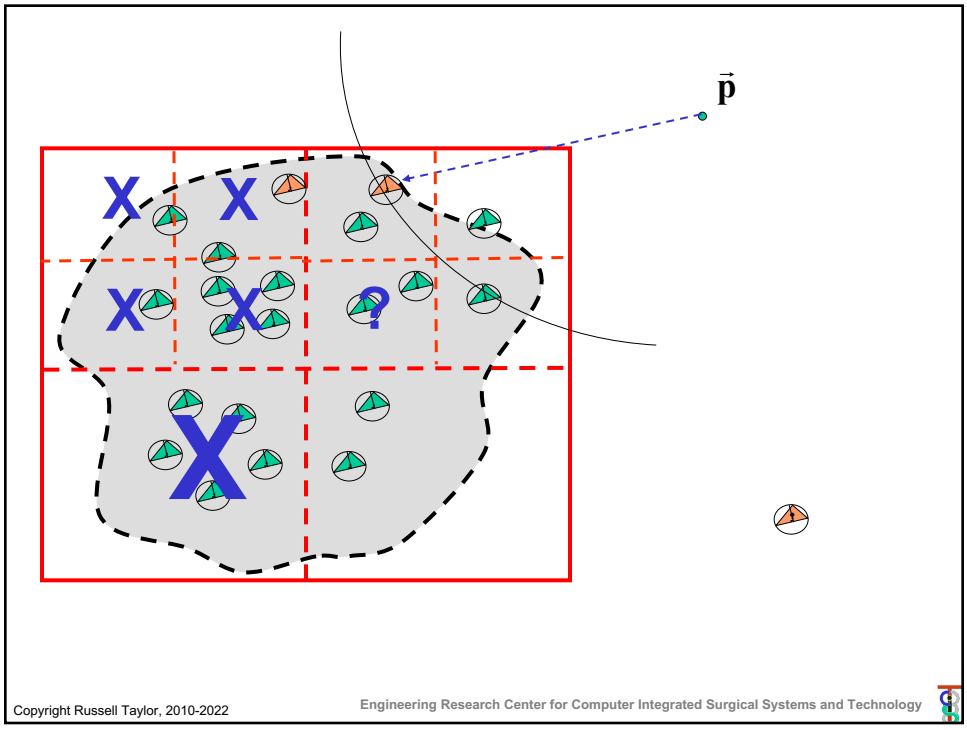
Copyright Russell Taylor, 2010-2022


38



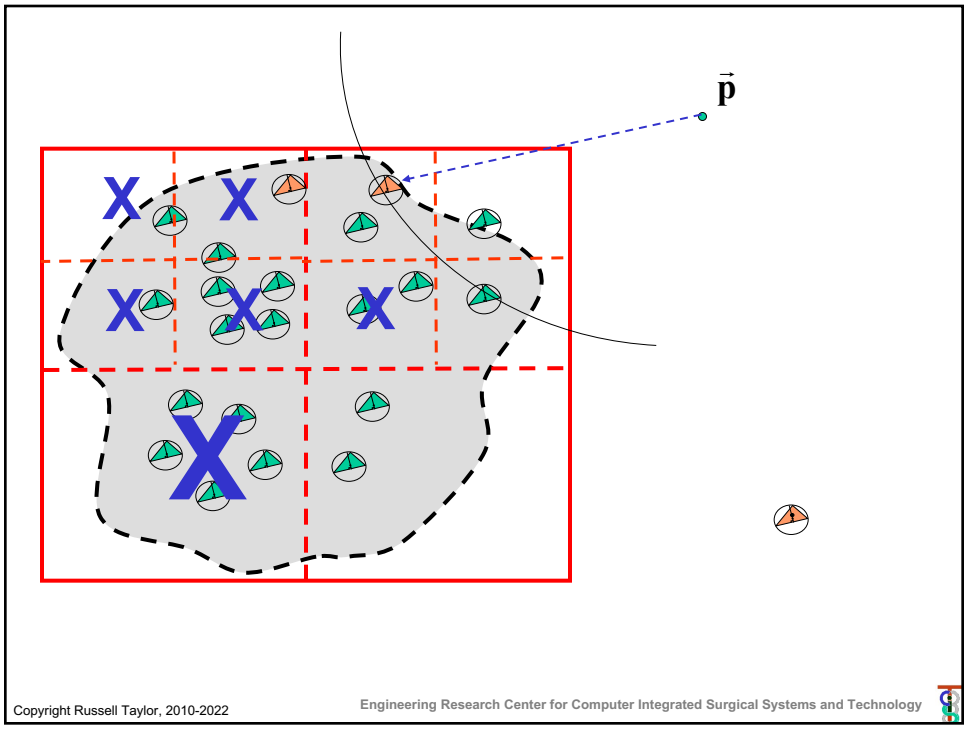
Copyright Russell Taylor, 2010-2022


39

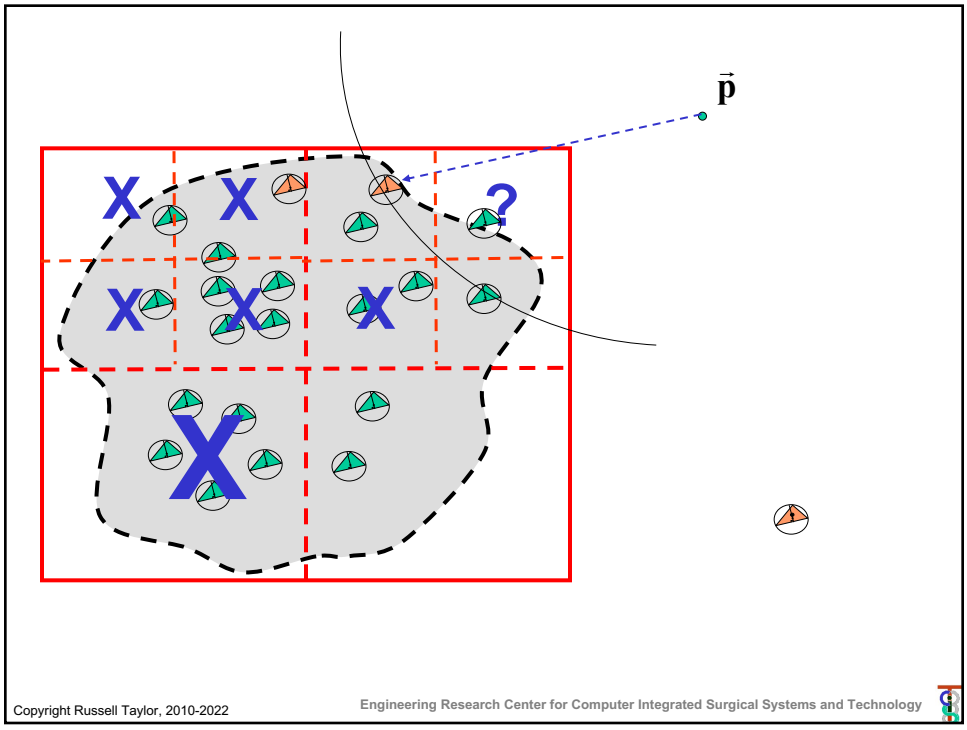


Copyright Russell Taylor, 2010-2022

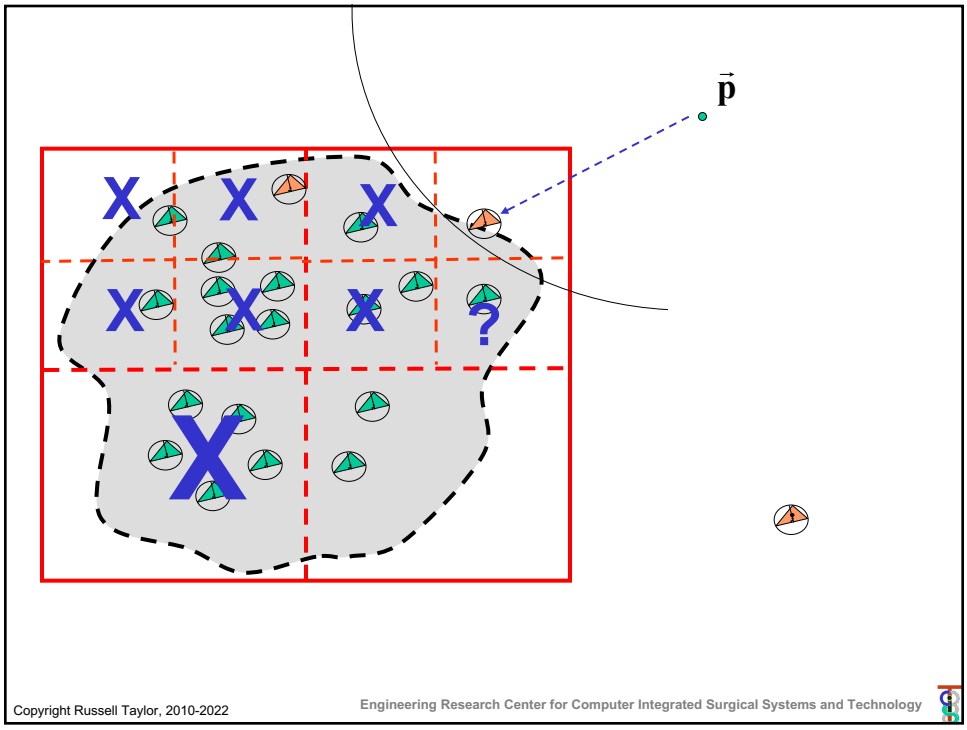

40


$\vec{p}$
Copyright Russell Taylor, 2010-2022
Engineering Research Center for Computer Integrated Surgical Systems and Technology

41

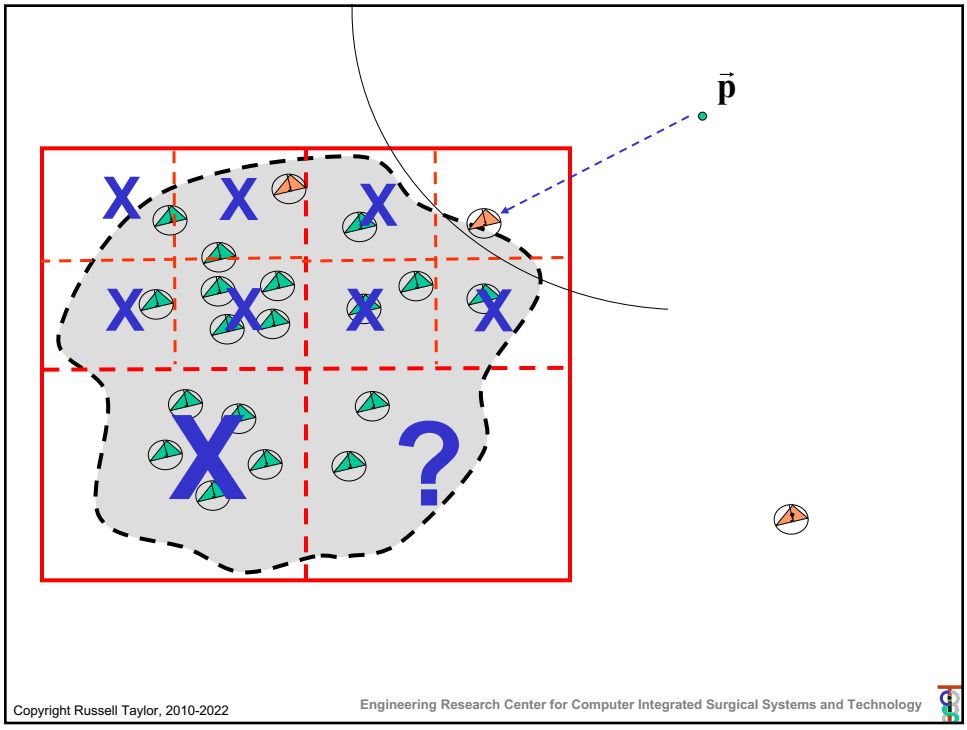

$\vec{p}$
?
Copyright Russell Taylor, 2010-2022
Engineering Research Center for Computer Integrated Surgical Systems and Technology

42



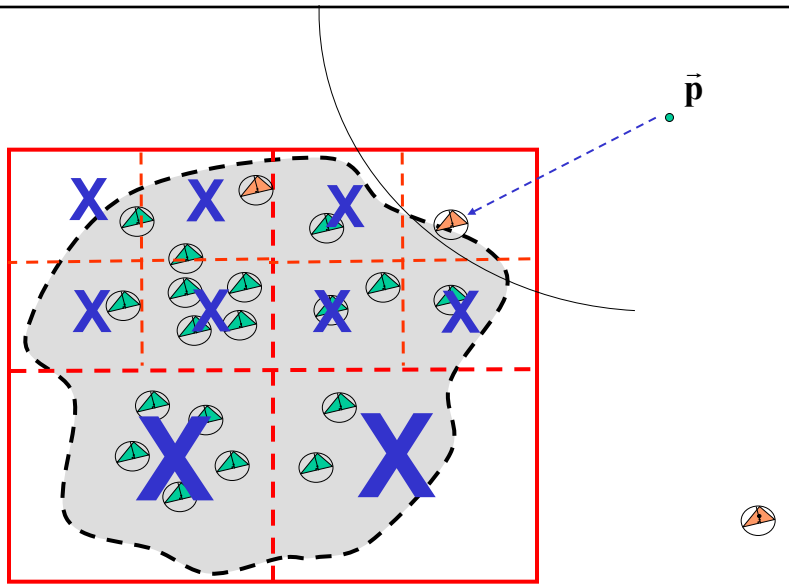
Copyright Russell Taylor, 2010-2022


43



Copyright Russell Taylor, 2010-2022

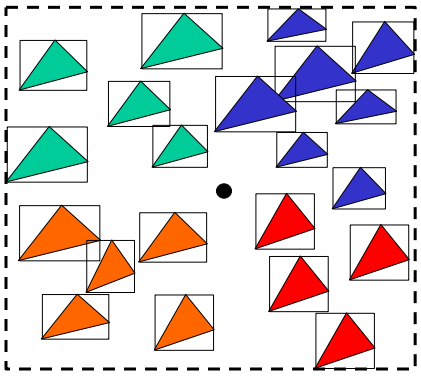

44




Copyright Russell Taylor, 2010-2022


45

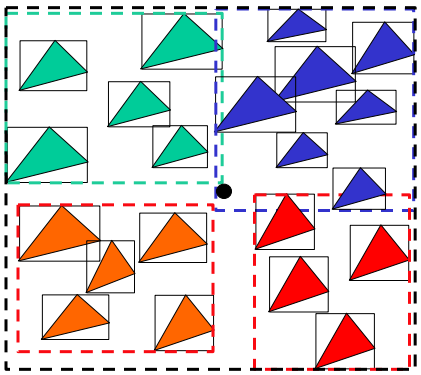
## Constructing octree of bounded things



Copyright Russell Taylor, 2010-2022


46

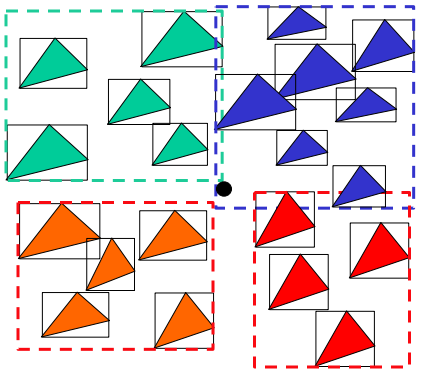
# Constructing octree of bounded things



Copyright Russell Taylor, 2010-2022


47

# Constructing octree of bounded things



Copyright Russell Taylor, 2010-2022


48

# Constructing octree of bounded things

```
class BoundingBoxTreeNode {
    Vec3 Center;          // splitting point
    Vec3 UB;              //  corners of box
    Vec3 LB;
    int HaveSubtrees;
    int nThings;
    BoundingBoxTreeNode* SubTrees[2][2][2];
    Thing** Things;
          :
          :
    BoundingBoxTreeNode(Thing** BS, int nS);
    ConstructSubtrees();
    void FindClosestPoint(Vec3 v, double& bound, Vec3& closest);
    };
```

Copyright Russell Taylor, 2010-2022

# Properties of "Things"

```
Class Thing
{ public:
          :
  vec3 SortPoint();
          // returns a point that can be used to sort the object
  vec3 ClosestPointTo(vec3 p);
          // returns point in this thing closest to p
  [vec3,vec3] EnlargeBounds(frame F,vec3 LB, vec3 UB);
          // Given frame F, and corners LB and UB of bounding box
          // around some other things, returns a the corners of a bounding
          // box that includes this Thing2 as well,
          // where Thing2=F.Inverse()*this thing
  [vec3,vec3] BoundingBox(F);
          { return EnlargeBounds(F,[∞, ∞, ∞],[-∞,-∞,-∞]);};

   int MayBeInBounds(Frame F, vec3 LB, vec3 UB);
          // returns 1 if any part of this F.Inverse()*this thing could be
          // in the bounding box with corners LB and UB
}
```

Copyright Russell Taylor, 2010-2022

# Triangle Things

```
Class Triangle : public Thing
{vec3 Corners[3];  // vertices of triangle
          :
 vec3 SortPoint() { return Mean(Corners);}; // or use Corner[0]
 [vec3,vec3] EnlargeBounds(frame F,vec3 LB, vec3 UB)
          { vec3 FiC[3]=F.inverse()*Corners;
            for (int I=0;I<3;I++)
                    { LB.x = min(LB.x,FiC[i].x); UB.x = max(UB.x,FiC[i].x);
                      LB.y = min(LB.y,FiC[i].y); UB.y = max(UB.y,FiC[i].y);
                      LB.z = min(LB.y,FiC[i].z); UB.z = max(UB.y,FiC[i].z);
                    };
          return [LB, UB];
          };
  [vec3,vec3] BoundingBox(F)
          { return EnlargeBounds(F,[∞, ∞, ∞],[-∞,-∞,-∞]);};
  int MayBeInBounds(Frame F, vec3 LB, vec3 UB)
          { vec3 FiC[3]=F.inverse()*Corners;
            for (int k=0;k<3; k++) if (InBounds(FiC[k],LB,UB)) return 1;
            return 0;}
}
```

Copyright Russell Taylor, 2010-2022

# Constructing octree of bounded things

```
BoundingBoxTreeNode(Thing** BS, int nS)
{   Things = BS; nThings = nS;
    UB = FindMaxCoordinates(Things,nThings);
    LB = FindMinCoordinates(Things,nThings);
   Center = LB+(UB-LB)/2.0;    // Splitting point
                               // Not necessarily the best
                               // Alternatives would be centroid or
                               // the median of the SortPoint()'s.
    ConstructSubtrees();
    };
```

Copyright Russell Taylor, 2010-2022

## Constructing octree of bounded things

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

Copyright Russell Taylor, 2010-2022

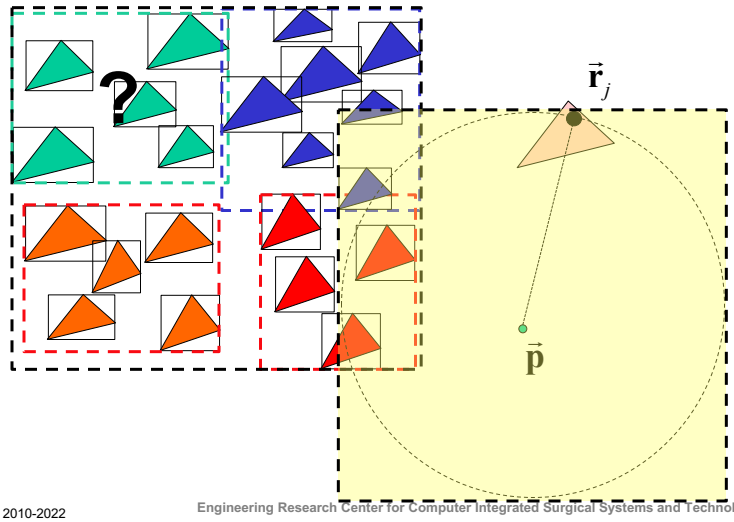
## Constructing octree of bounded things

```
SplitSort(Vec3& SplittingPoint, BoundingThing** Things,
              int& nnn, int& npn, … ,int& pnp)
{  // reorder Spheres(…) into eight buckets according to
   // comparison of coordinates of Thing(k)->SortPoint()
   // with coordinates of splitting point.  E.g., first bucket has
   //       Thing(k)->Center.x < SplittingPoint.x
   //       Thing(k)->Center.y < SplittingPoint.y
   //       Thing(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.

   // If desired, may be modified to simultaneously find a good
   // value for SplittingPoint (e.g., median)
}
```

Copyright Russell Taylor, 2010-2022

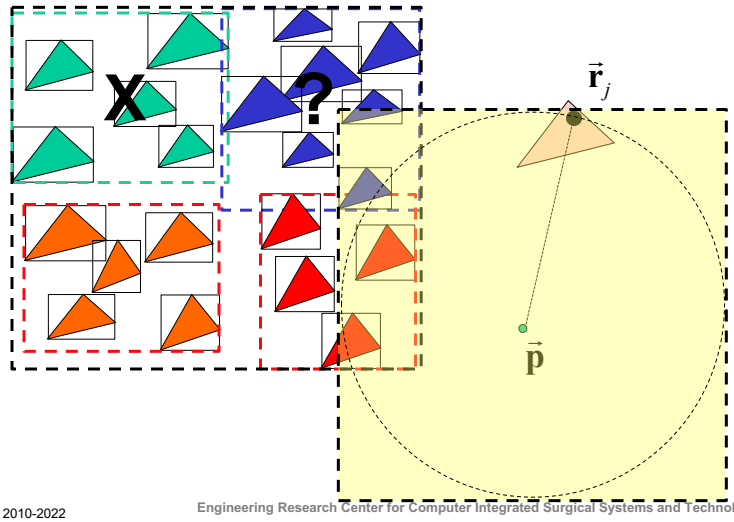
## Searching octree of bounded things




Copyright Russell Taylor, 2010-2022


55

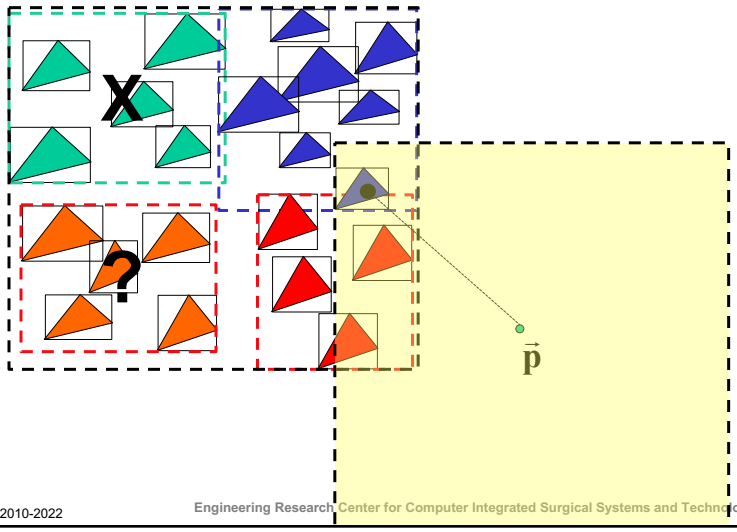
## Searching octree of bounded things




Copyright Russell Taylor, 2010-2022


56

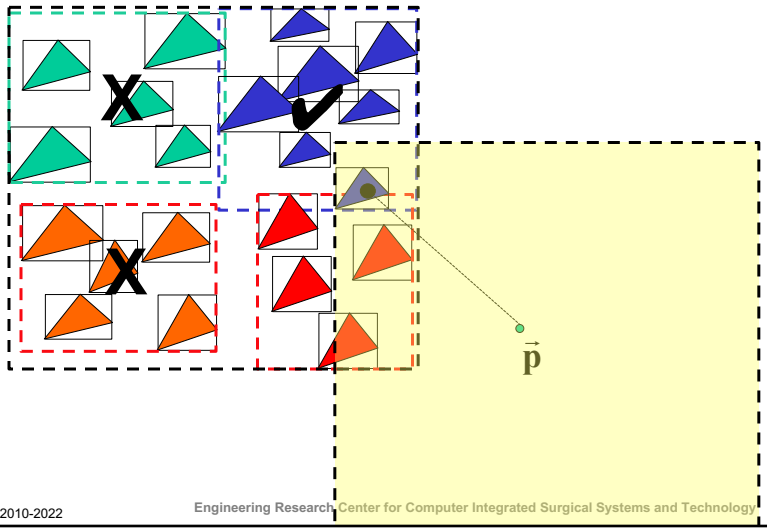
## Searching octree of bounded things




Copyright Russell Taylor, 2010-2022


57

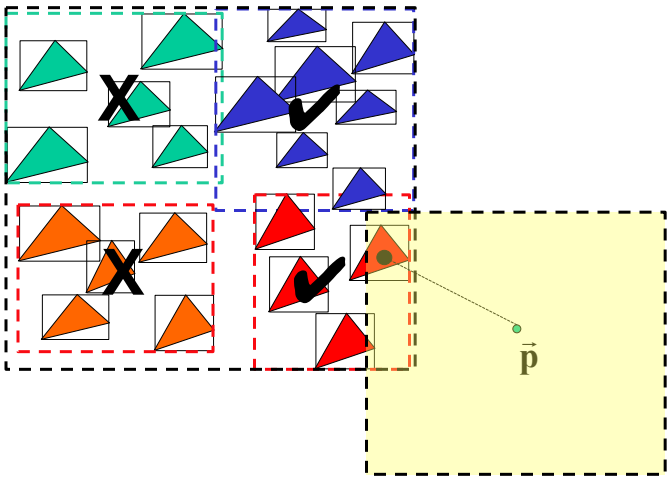
## Searching octree of bounded things




Copyright Russell Taylor, 2010-2022


58

# Searching octree of bounded things




Copyright Russell Taylor, 2010-2022


59

# Searching octree of bounded things

```
void BoundingBoxTreeNode::FindClosestPoint
                          (Vec3 v, double& bound, Vec3& closest)
{   if ((v.x > UB.x+bound) || (v.x<LB.x-bound)) return;
    if ((v.y > UB.y+bound) || (v.y<LB.y-bound)) return;
    if ((v.z > UB.z+bound) || (v.z<LB.z-bound)) 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<nThings;I++)
         UpdateClosest(Things[i],v,bound,closest);

};
```

Copyright Russell Taylor, 2010-2022


60

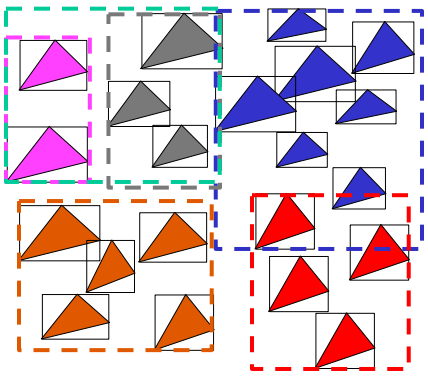
# Searching octree of bounded things

```
void UpdateClosest(Thing* Thing,
                   Vec3 v, double& bound, Vec3& closest)
{ Vec3 cp  = Thing->ClosestPointTo(v);
  dist = LengthOf(cp-v);
  if (dist<bound) { bound = dist; closest=cp;};
};
```

Copyright Russell Taylor, 2010-2022


61

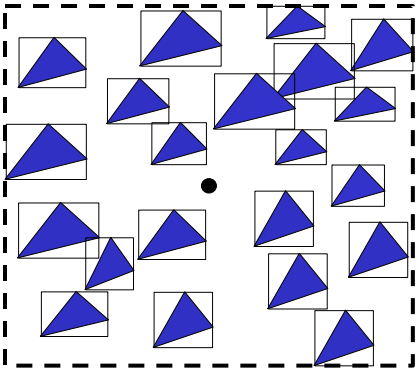
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


62

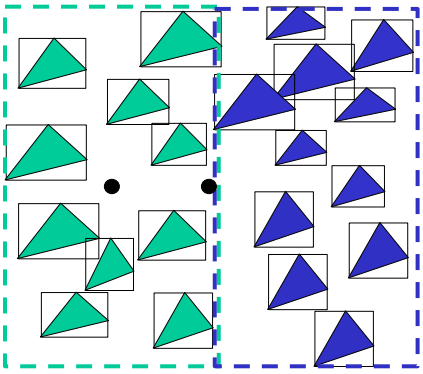
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


63

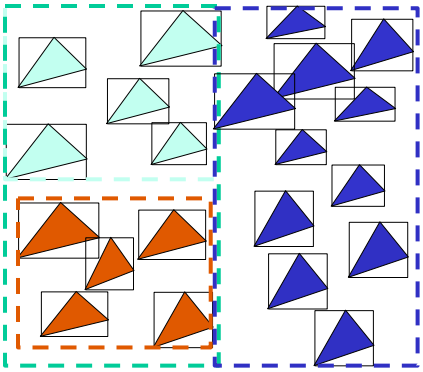
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


64

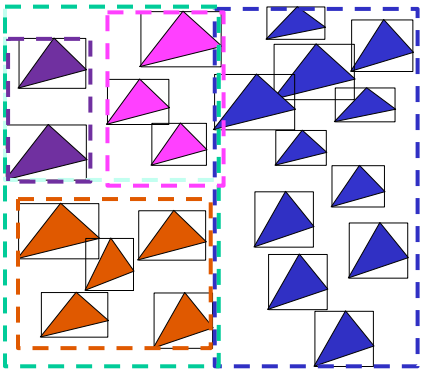
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


65

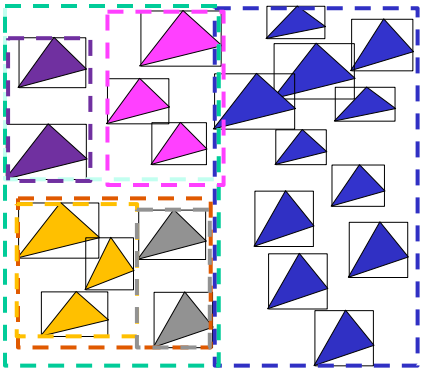
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


66

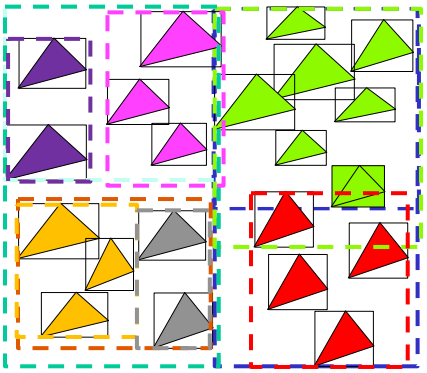
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


67

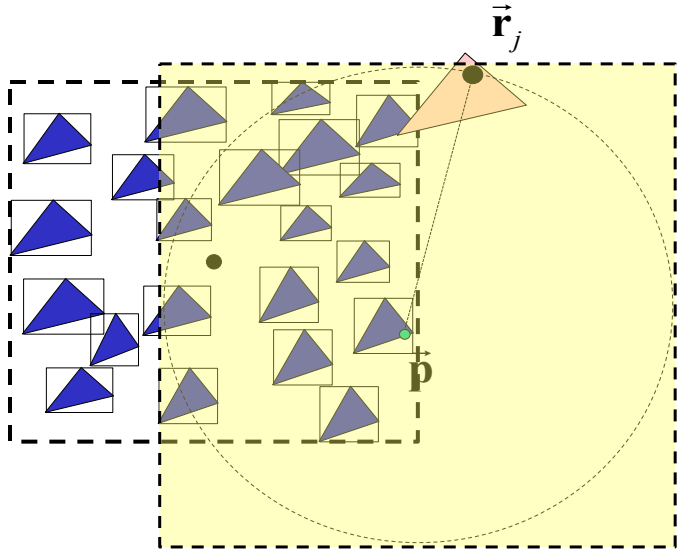
# Constructing KD tree of bounded things



Copyright Russell Taylor, 2010-2022


68

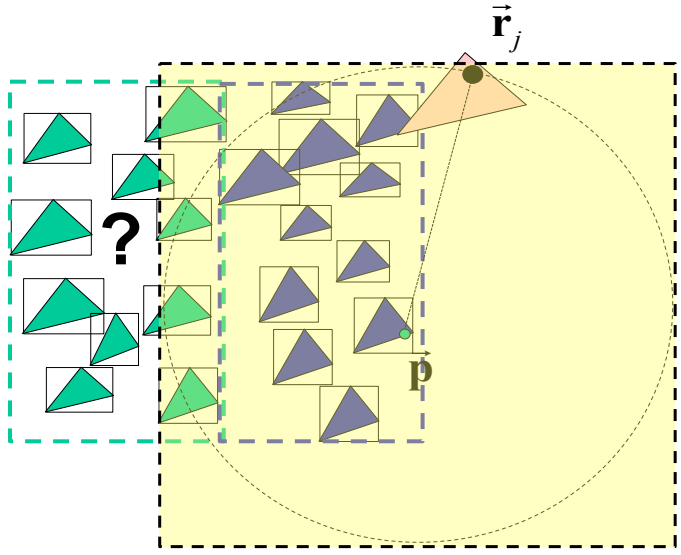
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


69

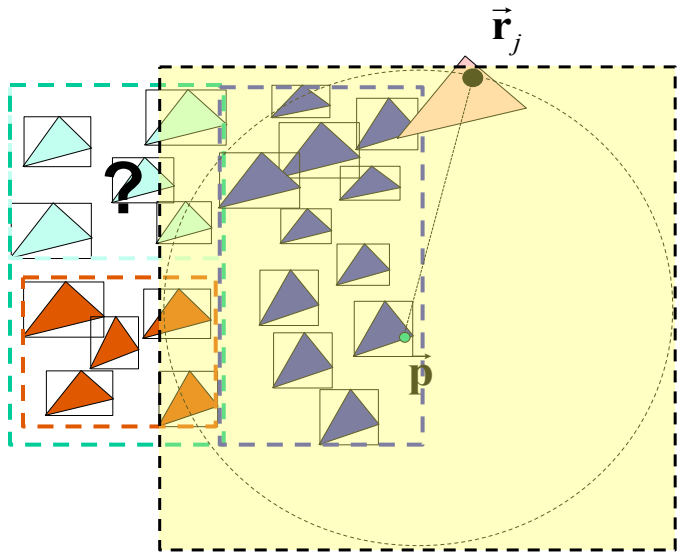
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


70

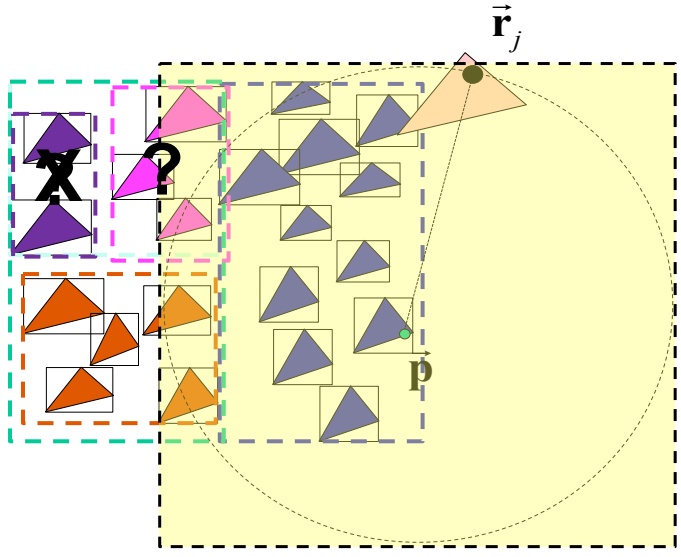
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


71

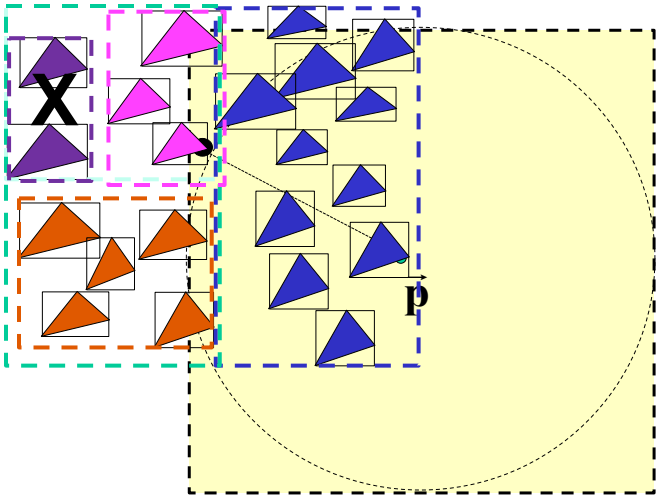
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


72

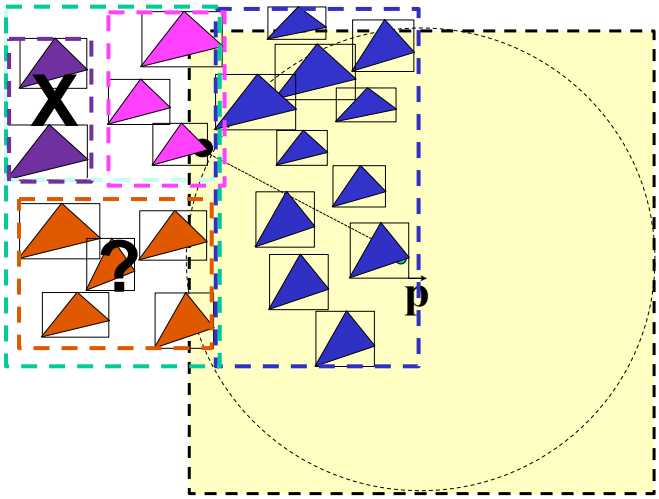
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


73

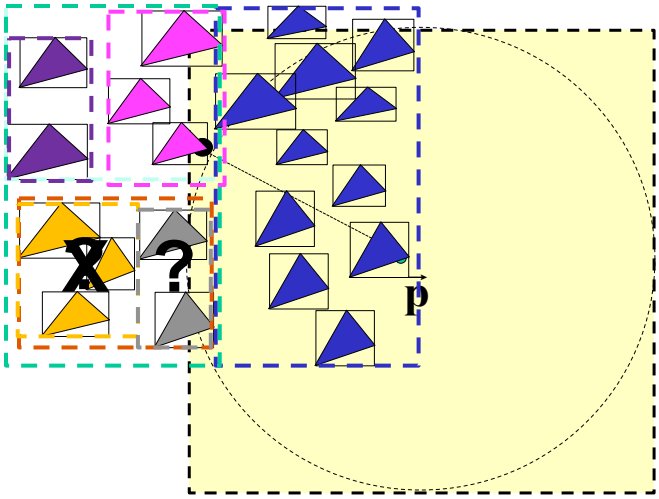
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


74

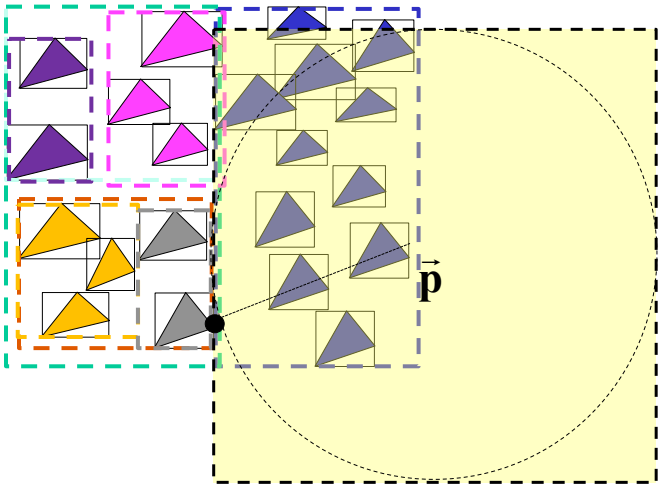
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


75

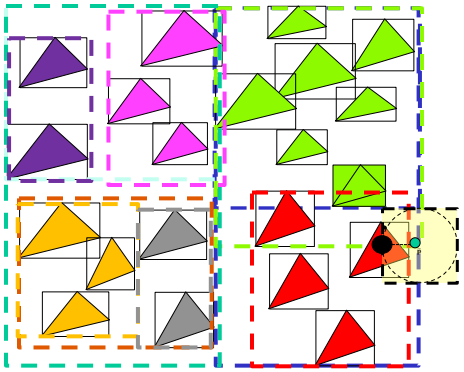
## Searching KD Tree of Bounded Things




Copyright Russell Taylor, 2010-2022


76

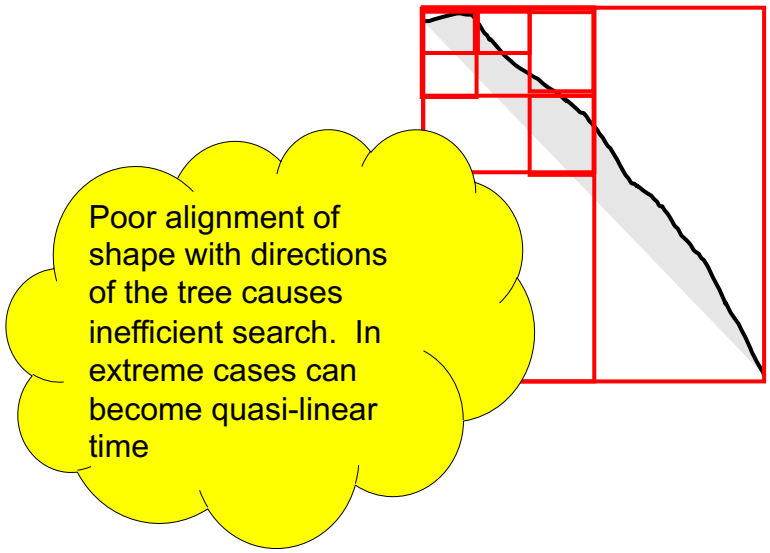
## Searching KD Tree of Bounded Things



Copyright Russell Taylor, 2010-2022


77

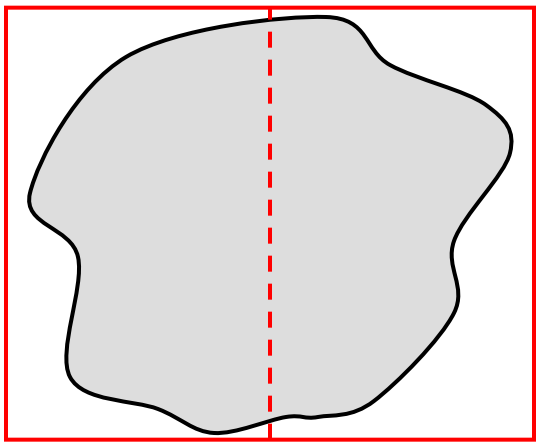
## Possible pathology with KD trees and Octrees




Copyright Russell Taylor, 2010-2022


78

## Solution: Covariance Trees*

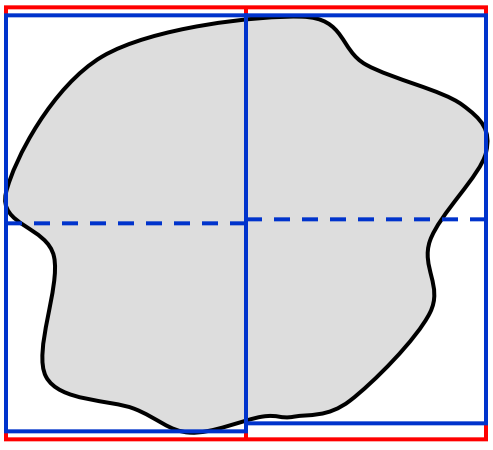


* Referred to by my former student Seth Billings as Principal Direction Trees

Copyright Russell Taylor, 2010-2022


79

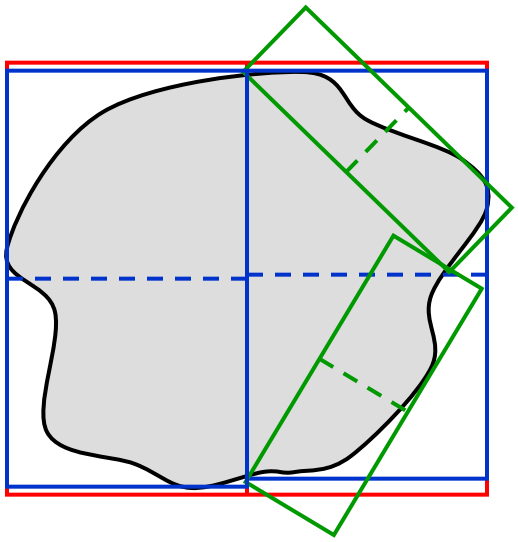
## Covariance Trees



Copyright Russell Taylor, 2010-2022


80

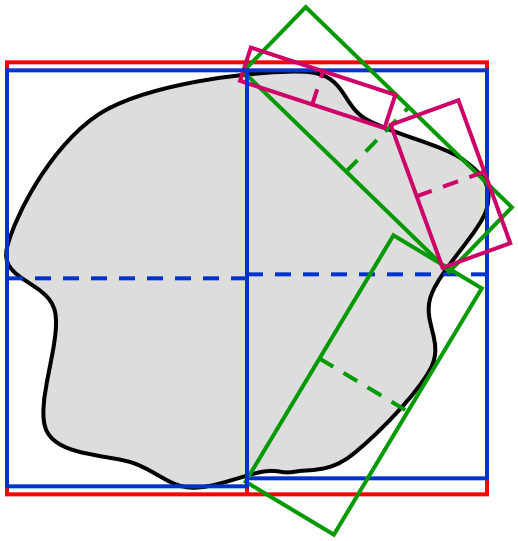
## Covariance Trees



Copyright Russell Taylor, 2010-2022


81

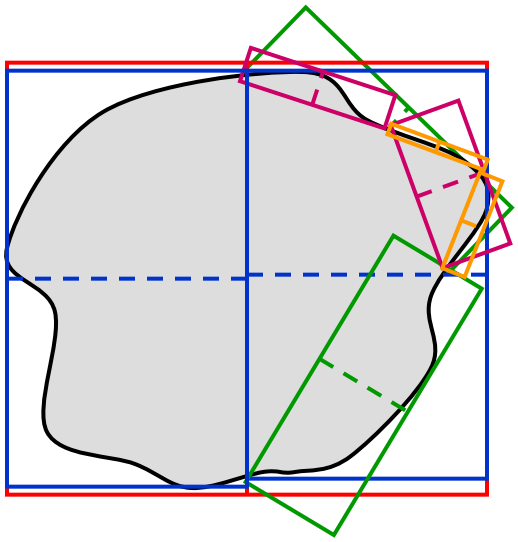
## Covariance Trees



Copyright Russell Taylor, 2010-2022


82

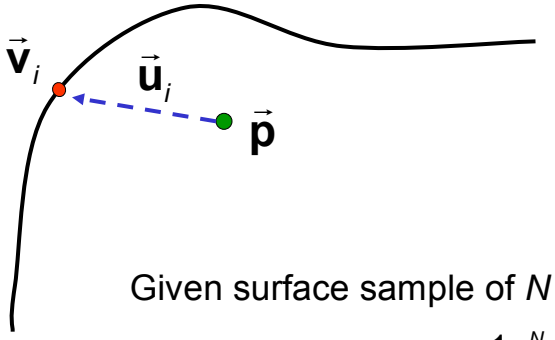
## Covariance Trees



Copyright Russell Taylor, 2010-2022


83

## Covariance Tree Construction




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

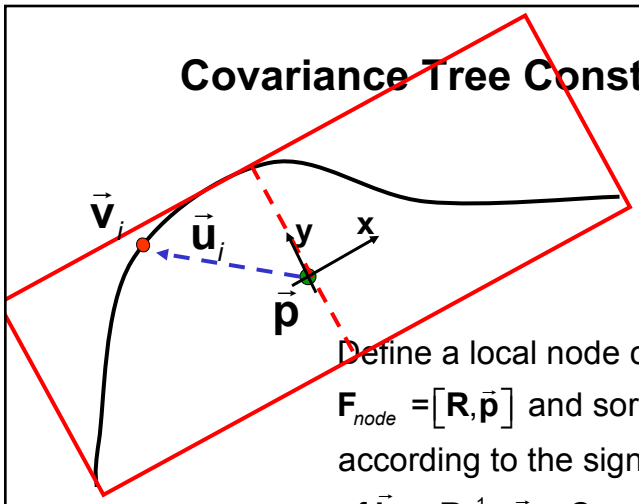
Compute centroid $\vec{\mathbf{p}} = \frac{1}{N}\sum_{i=1}^{N}\vec{\mathbf{v}}_i$

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

Copyright Russell Taylor, 2010-2022


84

## Covariance Tree Construction




Define a local node coordinate system $\mathbf{F}_{node} = [\mathbf{R}, \vec{\mathbf{p}}]$ 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

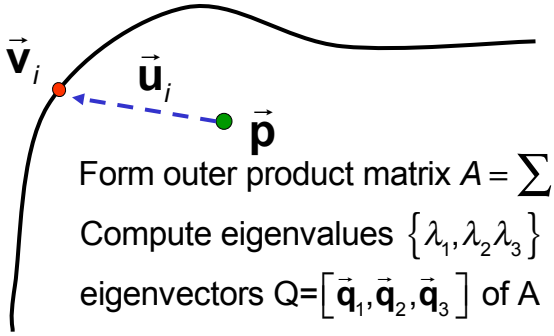
$$\vec{\mathbf{b}}^{\min} \le \mathbf{R}^{-1} \bullet \vec{\mathbf{u}}_i \le \vec{\mathbf{b}}^{\max}$$

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

Copyright Russell Taylor, 2010-2022


85

## 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 $Q = [\vec{\mathbf{q}}_1, \vec{\mathbf{q}}_2, \vec{\mathbf{q}}_3]$ 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)

Copyright Russell Taylor, 2010-2022


86

# Constructing Cov Tree of Objects

```
class CovTreeNode {
    Frame F;          // splitting point
    Vec3 UB;              //  corners of box
    Vec3 LB;
    int HaveSubtrees;
    int nThings;
    CovTreeNode* SubTrees[2];
    Thing** Things;
          :
          :
    CovTreeNode(Thing** Ts, int nT);
    ConstructSubtrees();
    void FindClosestPoint(Vec3 v, double& bound, Vec3& closest);
    };
```

Copyright Russell Taylor, 2010-2022

87

# Constructing Cov Tree of Things

```
CovTreeNode(Thing** Ts, int nT)
{   Things = Ts; nThings = nT;
    F = ComputeCovFrame(Things,nThings);
    [UB,LB] = FindBoundingBox(F,Things,nThings);
    ConstructSubtrees();
    };

[vec3 UB,vec3 LB]=FindBoundingBox(F,Things,nThings)
{ UB = LB = F.inverse()*(Things[0]->SortPoint());
  for (int k=0;k<nThings;k++)
    { [LB,UB] = Things[k]->EnlargeBounds(F,LB,UB);
    };
  return [UB,LB];
};
```

Copyright Russell Taylor, 2010-2022

88

# Constructing Cov Tree of Things

```
Frame F = FindCovFrame(Thing** Ts, int nT)
{   [vec3 Points, int nP] = ExtractPoints(Ts,nT);
          // may extract nT sort points or perhaps
          // all corner points if things are triangles
    return FindCovFrame(Points,nP);
    };

Frame F = FindCovFrame(vec3* Ps, int nP)
{   vec3 C = Centroid(Ps,nP);
    Matrix A = 0;
    for (i=0;i<nP;i++) A+=OuterProduct(Ps[i],Ps[i]);
    R = CorrespondingRotationMatrix(A);  // see notes
    return Frame(R,C);
    };
```

Copyright Russell Taylor, 2010-2022

# Constructing Cov Tree of Things

```
ConstructSubtrees()
{  if (nThings<= minCount || length(UB-LB)<=minDiag)
       { HaveSubtrees=0; return; };
   HaveSubtrees = 1;
   int nSplit;
   nSplit = SplitSort(F,things);
   Subtrees[0] = CovarianceTreeNode(Things[0],nSplit);
   Subtrees[1] = CovarianceTreeNode(Things[nSplit],nThings-nSplit);
}
```

Copyright Russell Taylor, 2010-2022

## Constructing Cov Tree of Things

```
Int nSplit = SplitSort(Frame F, Thing** Ts,int nT)
{  // find an integer nSplit and reorder Things(…)  so that
   //      F.inverse()*(Thing[k]->SortPoint()).x <0 if and only if k<nSplit
   // This can be done "in place" by suitable exchanges.
   return nSplit;
}
```

Copyright Russell Taylor, 2010-2022

## 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 $\vec{\mathbf{c}}$, $dist = \left\| \vec{\mathbf{a}} - \vec{\mathbf{c}} \right\|$

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.

Copyright Russell Taylor, 2010-2022

## Searching a Covariance Tree of Things

```
void CovarianceTreeNode::FindClosestPoint
                          (Vec3 v, double& bound, Vec3& closest)
{   vLocal=F.Inverse()*v; // transform v to local coordinate system
    if (vLocal.x > UB.x+bound) return;
    if (vLocal.y > UB.y+bound) return;
       // similar checks on remaining bounds go here …. ;
    if (vLocal.z < LB.z-bound) return;
   if (HaveSubtrees)
       { Subtrees[0].FindClosestPoint(v,bound,closest);
         Subtrees[1].FindClosestPoint(v,bound,closest);
       }
  else
    for (int i=0;i<nThings;I++)
         UpdateClosest(Things[i],v,bound,closest);
};
```

Copyright Russell Taylor, 2010-2022

93

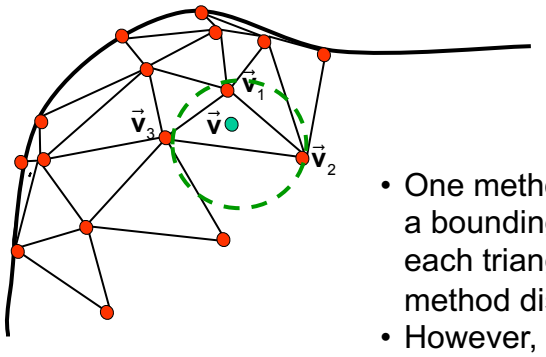
## Searching a Covariance Tree of Things

```
void UpdateClosest(Thing* T, Vec3 v, double& bound, Vec3& closest)
{  // here can include filter if have a bounding sphere to check
   Vec3 cp  = T->ClosestPointTo(v);
   dist = LengthOf(cp-v);
   if (dist<bound) { bound = dist; closest=cp;};
};
```

Copyright Russell Taylor, 2010-2022

94

# Covariance Trees for Triangle Meshes

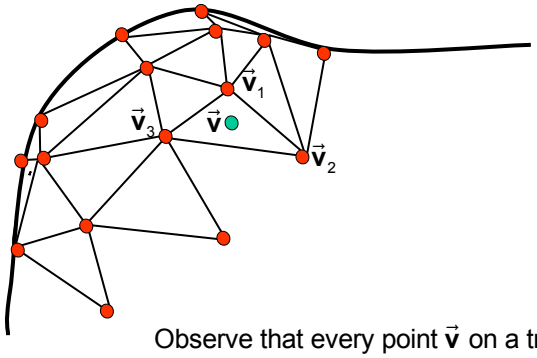



- One method is simply to place a bounding sphere around each triangle, and then use the method discussed previously
- However, this may be inconvenient if the mesh is deforming

Copyright Russell Taylor, 2010-2022


95

# Covariance Trees for Triangle Meshes

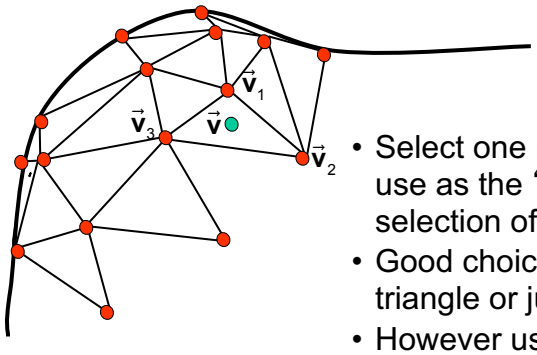



Observe that every point $\vec{\mathbf{v}}$ on a triangle $\left[\vec{\mathbf{v}}_1, \vec{\mathbf{v}}_2, \vec{\mathbf{v}}_3\right]$ can be expressed as a convex linear combination $\vec{\mathbf{v}} = \lambda_1 \vec{\mathbf{v}}_1 + \lambda_2 \vec{\mathbf{v}}_2 + \lambda_3 \vec{\mathbf{v}}_3$ with $\lambda_1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

Copyright Russell Taylor, 2010-2022


96

## Covariance Trees for Triangle Meshes

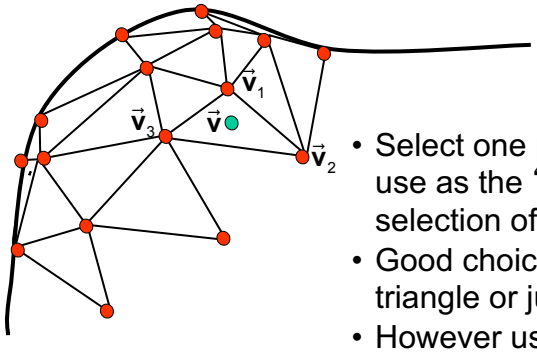



- Select one point on the triangle to use as the "sort" point for selection of left/right subtrees.
- Good choices are centroid of triangle or just one of the vertices.
- However use all vertices of each triangle in determining the size of bounding boxes.
- Note this would work equally well for octrees.

Copyright Russell Taylor, 2010-2022


97

## Covariance Trees for Triangle Meshes

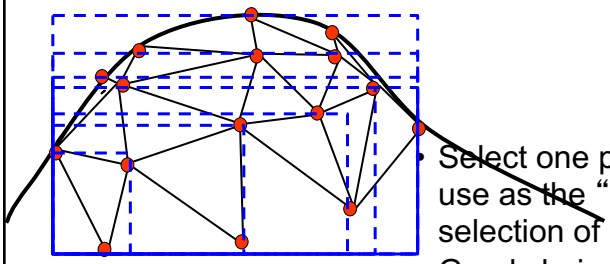



- Select one point on the triangle to use as the "sort" point for selection of left/right subtrees.
- Good choices are centroid of triangle or just one of the vertices.
- However use all vertices of each triangle in determining the size of bounding boxes.
- Note this would work equally well for octrees.

Copyright Russell Taylor, 2010-2022


98

## Covariance Trees for Triangle Meshes

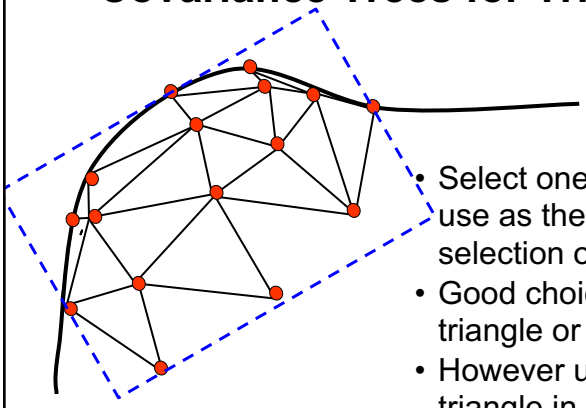


- Select one point on the triangle to use as the "sort" point for selection of left/right subtrees.
- Good choices are centroid of triangle or just one of the vertices.
- However use all vertices of each triangle in determining the size of bounding boxes.
- Note this would work equally well for octrees.

Copyright Russell Taylor, 2010-2022


99

## Covariance Trees for Triangle Meshes

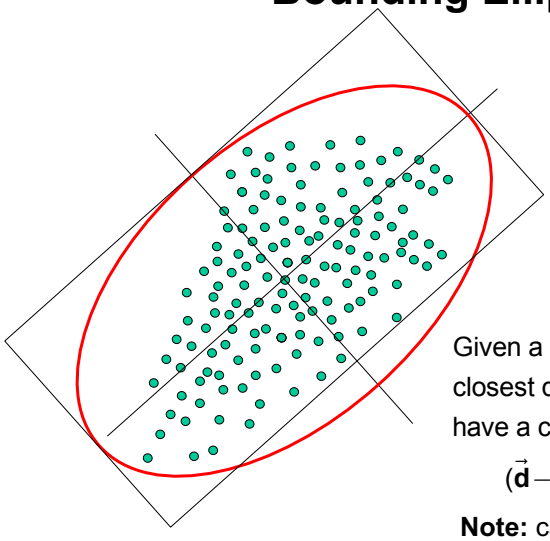


- Select one point on the triangle to use as the "sort" point for selection of left/right subtrees.
- Good choices are centroid of triangle or just one of the vertices.
- However use all vertices of each triangle in determining the size of bounding boxes.
- Note this would work equally well for octrees.

Copyright Russell Taylor, 2010-2022


100

# An Alternative to Bounding Boxes: Bounding Ellipsoids



Compute

$$\vec{\mathbf{p}}_C = \frac{1}{N}\sum_N \vec{\mathbf{v}}_i$$

$$\vec{\mathbf{u}}_i = \vec{\mathbf{v}}_i - \vec{\mathbf{p}}_C$$

$$\mathbf{A} = \sum_i \vec{\mathbf{u}}_i \vec{\mathbf{u}}_i^T = \mathbf{Q}\Lambda\mathbf{Q}^T$$

$$\Lambda = diag(\vec{\lambda})$$

$$\rho^2 = \max_i \vec{\mathbf{u}}_i^T \mathbf{A} \vec{\mathbf{u}}_i$$

Given a search point $\vec{\mathbf{d}}$ and previous closest distance $\delta$, the ellipsoid may have a closer point if

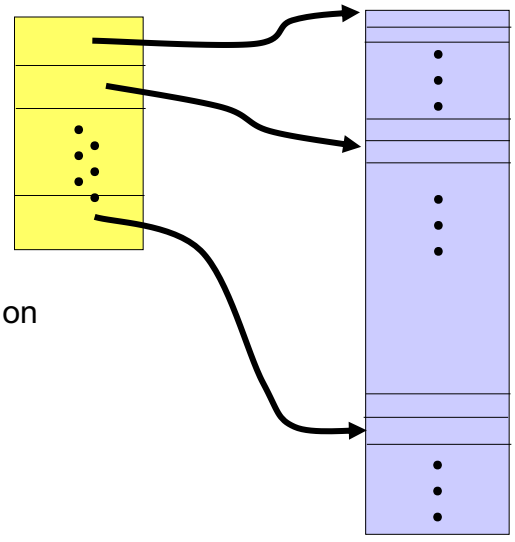
$$(\vec{\mathbf{d}} - \vec{\mathbf{p}}_C)^T \mathbf{A} (\vec{\mathbf{d}} - \vec{\mathbf{p}}_C) < \rho^2 + (\delta \max_k \lambda_k)^2$$

**Note:** can probably get a tighter bound, but this will work

Copyright Russell Taylor, 2010-2022


101

# Simple spatial sort



Point or triangle list

**Note:** must ensure that every element is in at least one bucket

Index based on coordinates

Copyright Russell Taylor, 2010-2022


102