Contact Generation

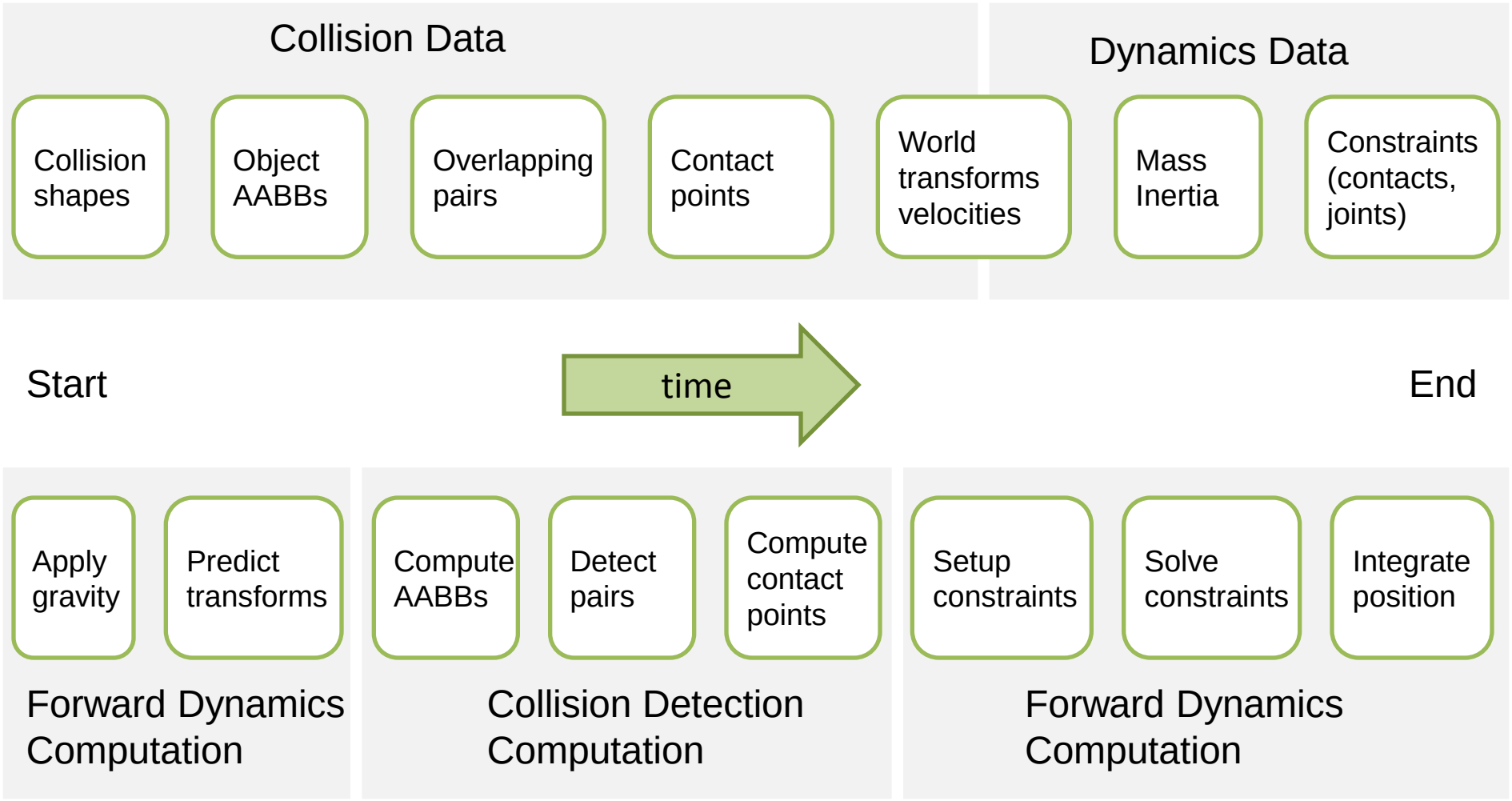
Erwin Coumans

Sony Computer Entertainment US R&D

Contact generation

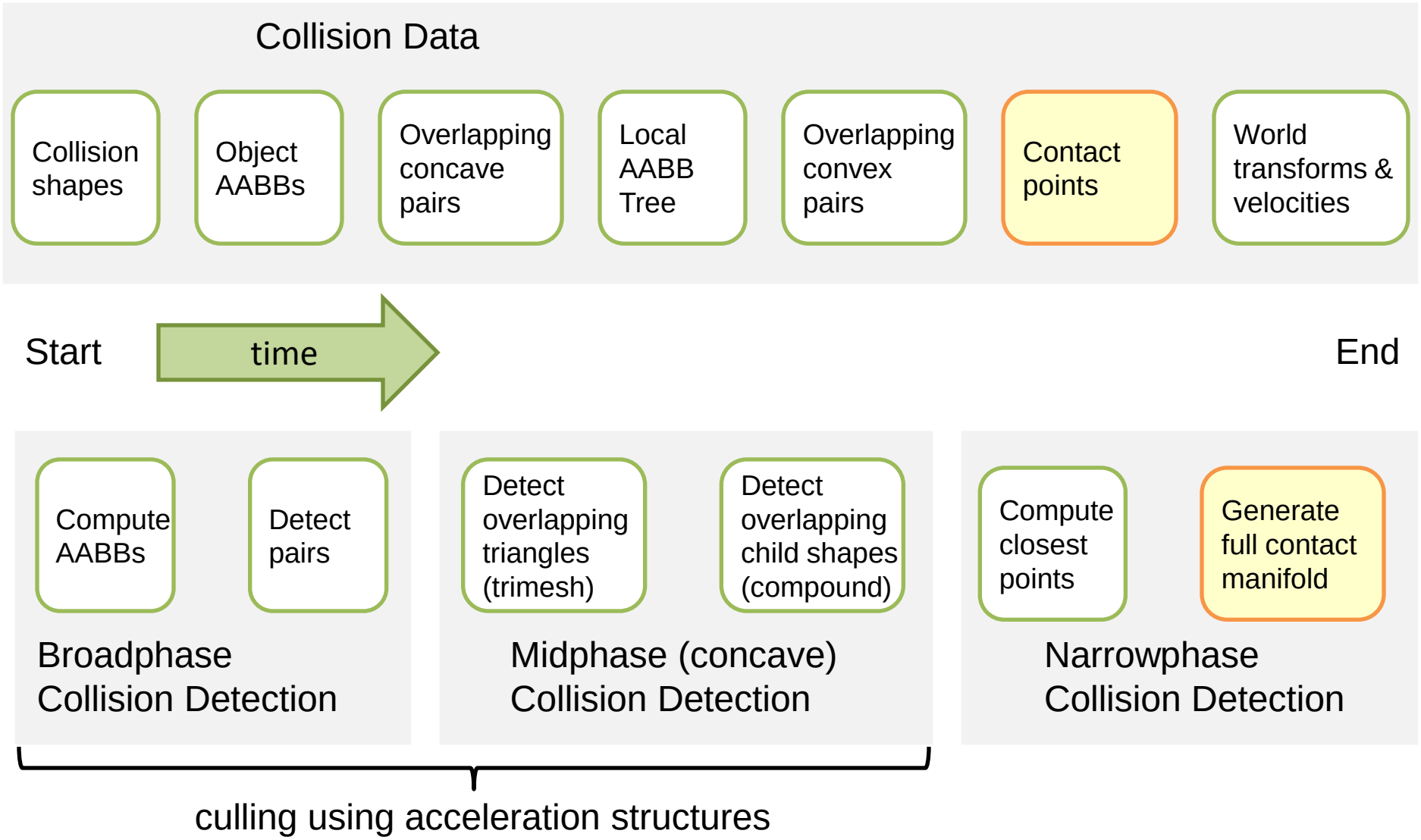
- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

Physics Pipeline



AABB = axis aligned bounding box

Collision Detection Pipeline



Contact generation

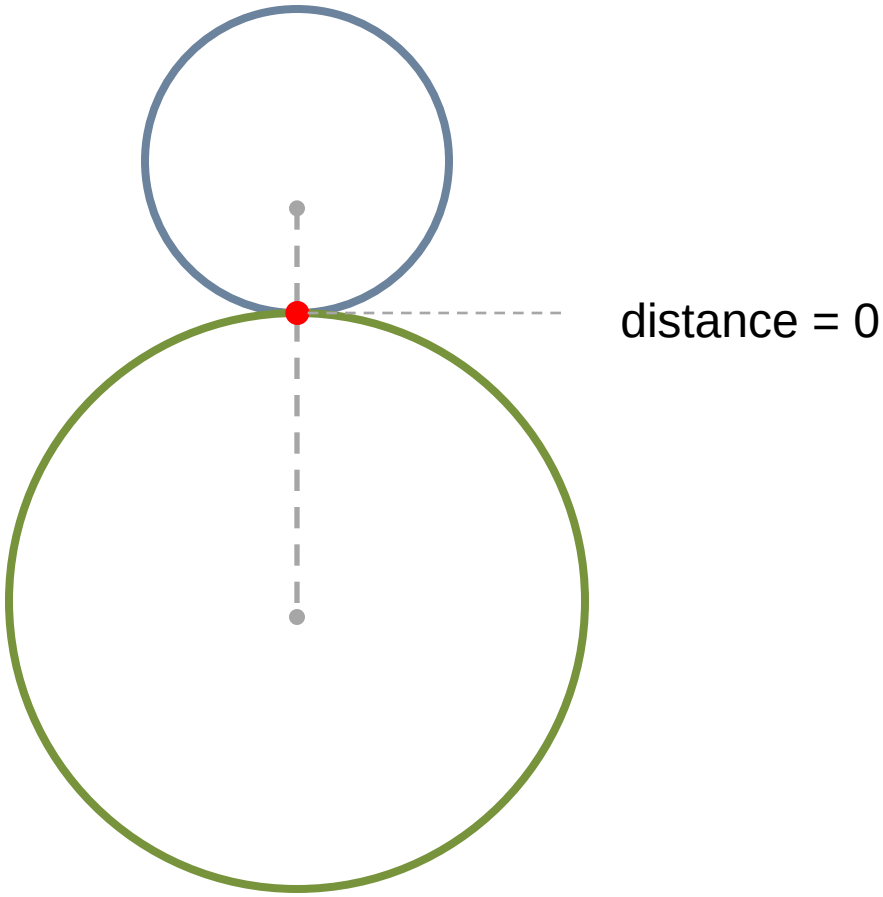
- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

Closest point computation

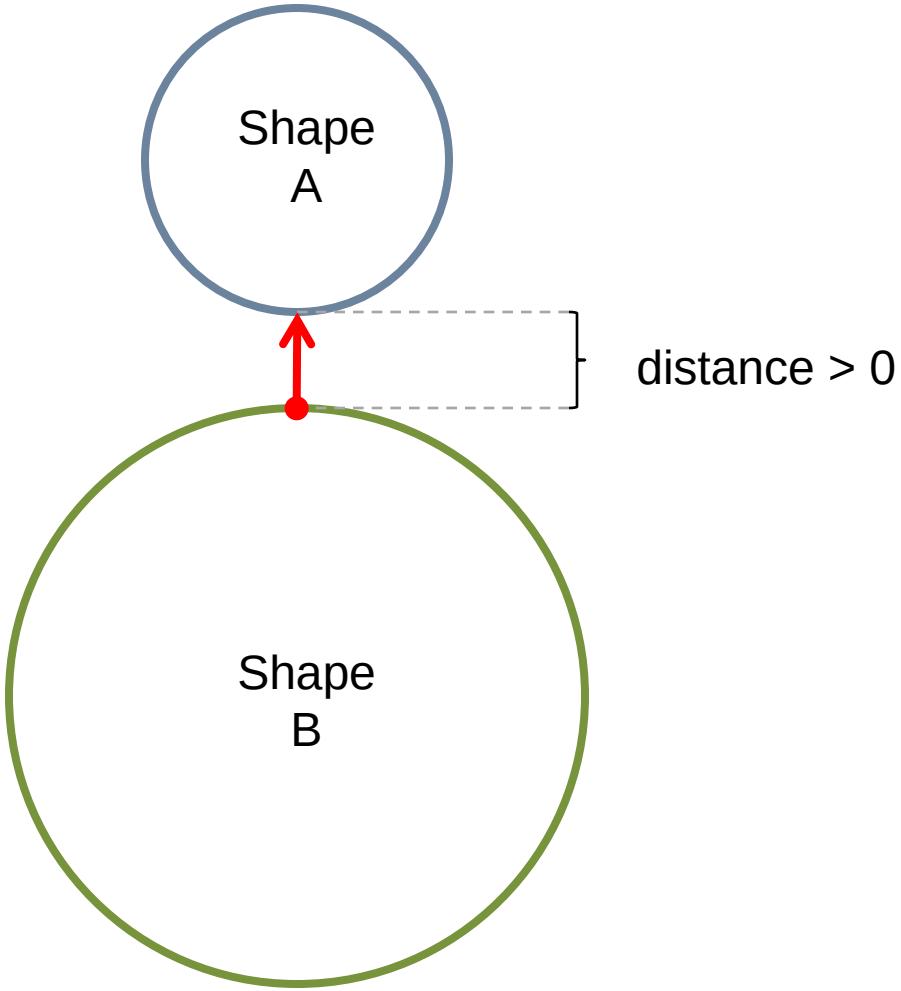
```
void computeClosestPoints(…);
```

- Black box is discussed in Erin Catto's GJK talk
- GJK needs companion algorithm for penetration

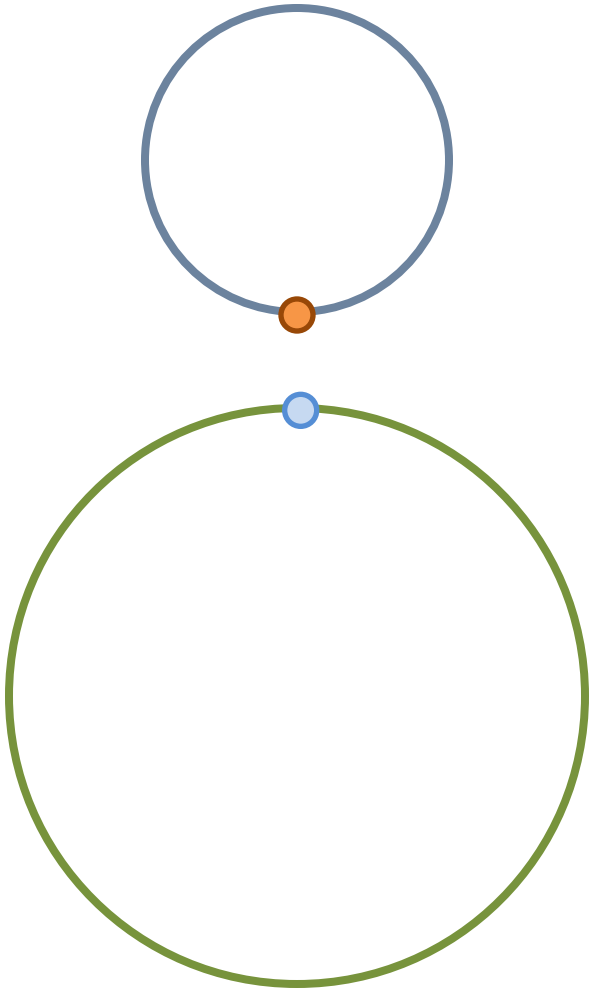
Touching contact



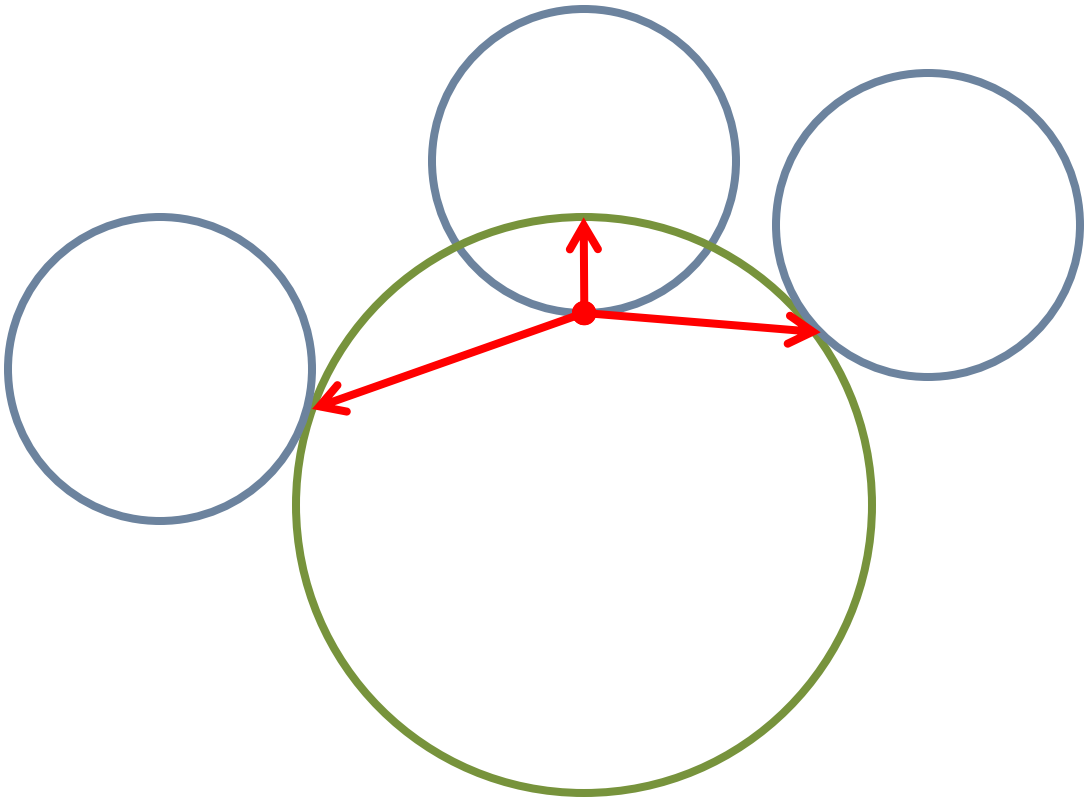
Shortest distance



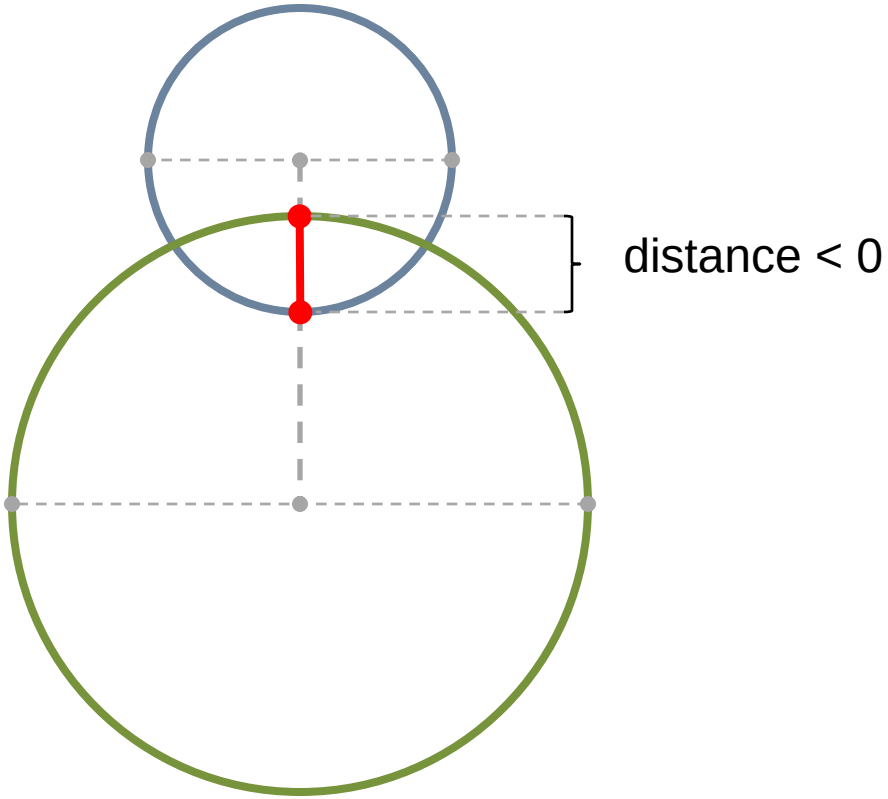
Closest points (witness)



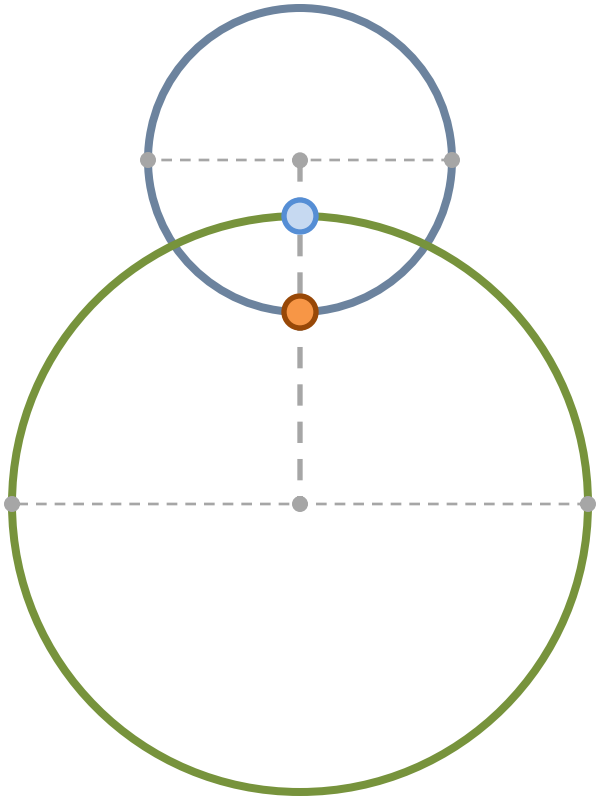
Separating vectors



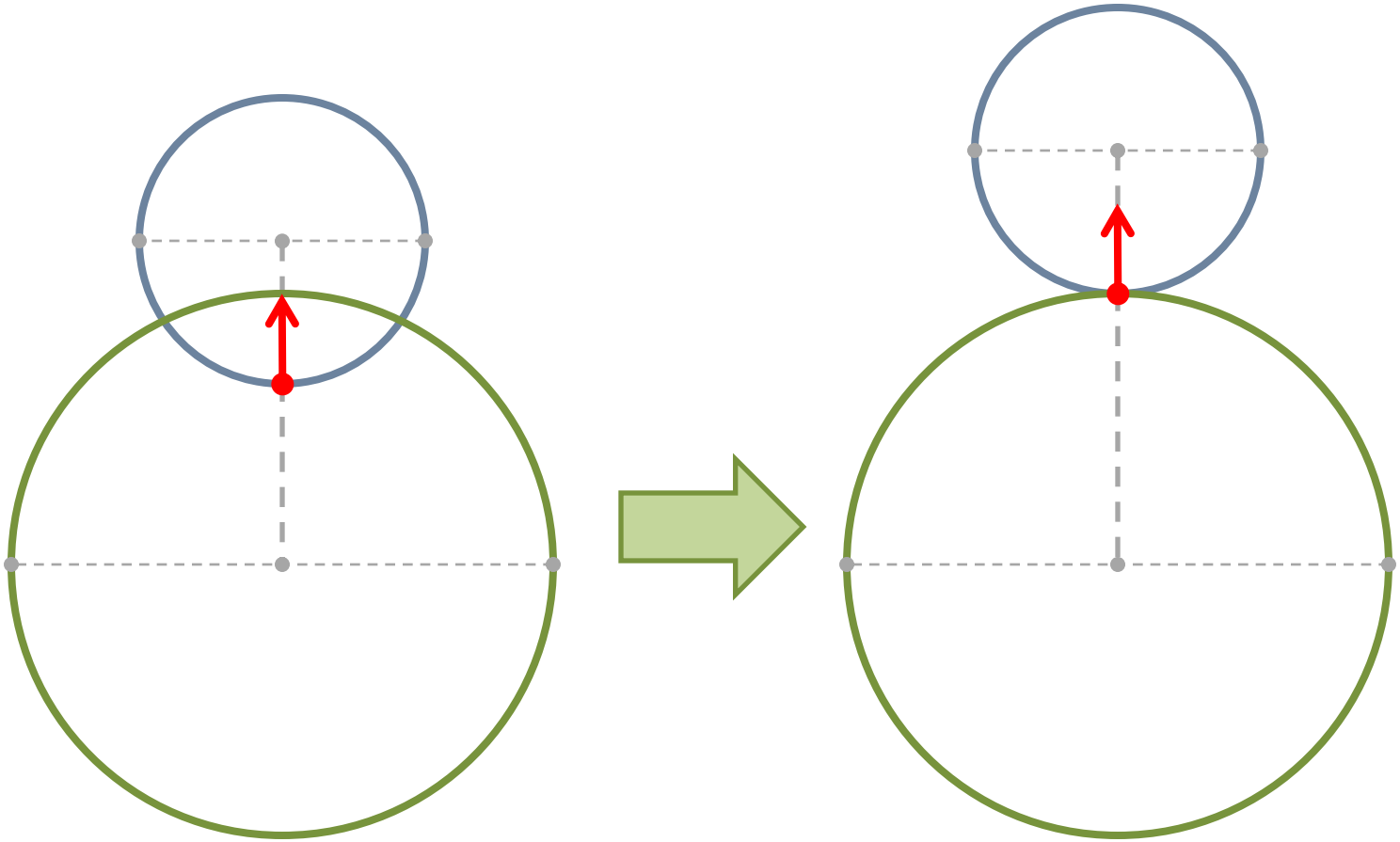
Minimum translational distance



Points of deepest penetration

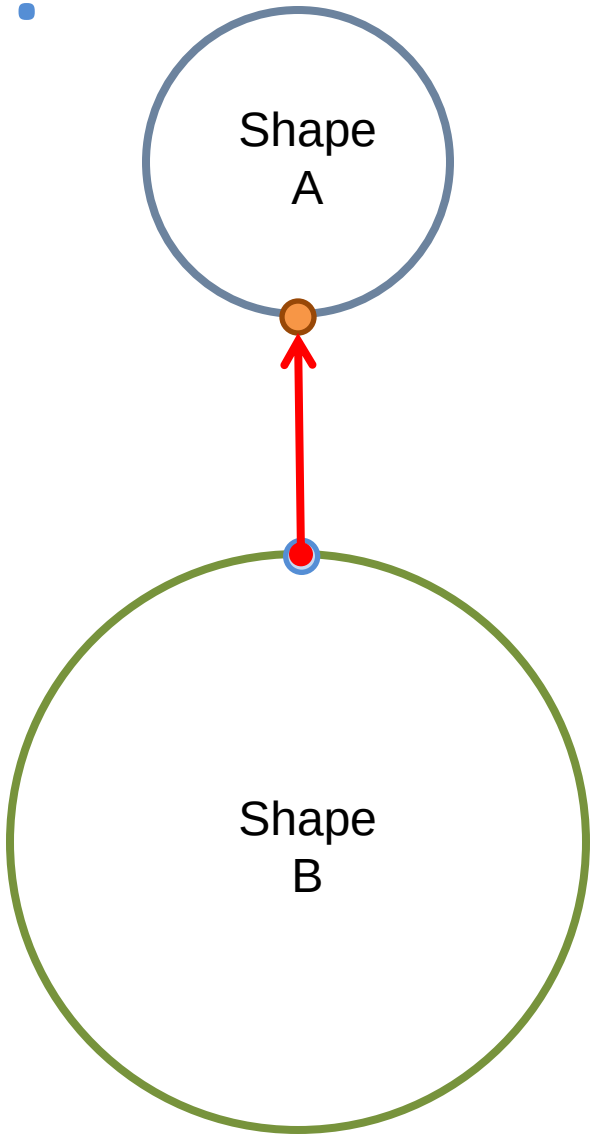


Minimum separating vector



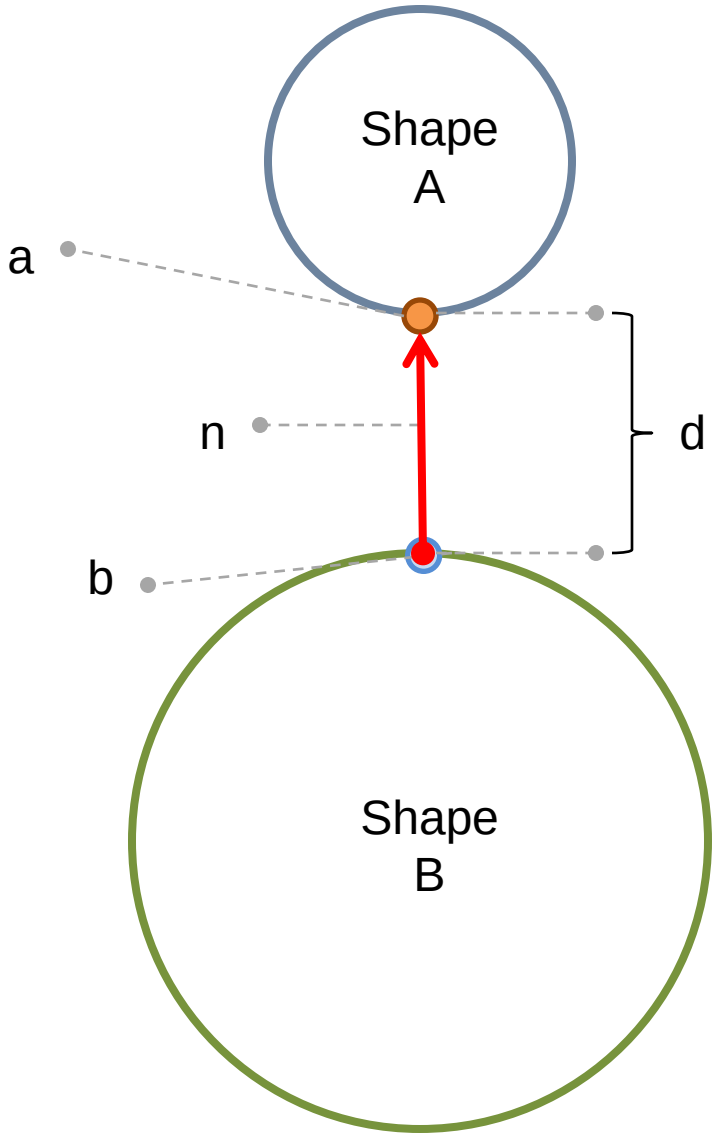
What is a contact point?

- Distance
- A separating normal
- Pair of closest points



Removing redundancy

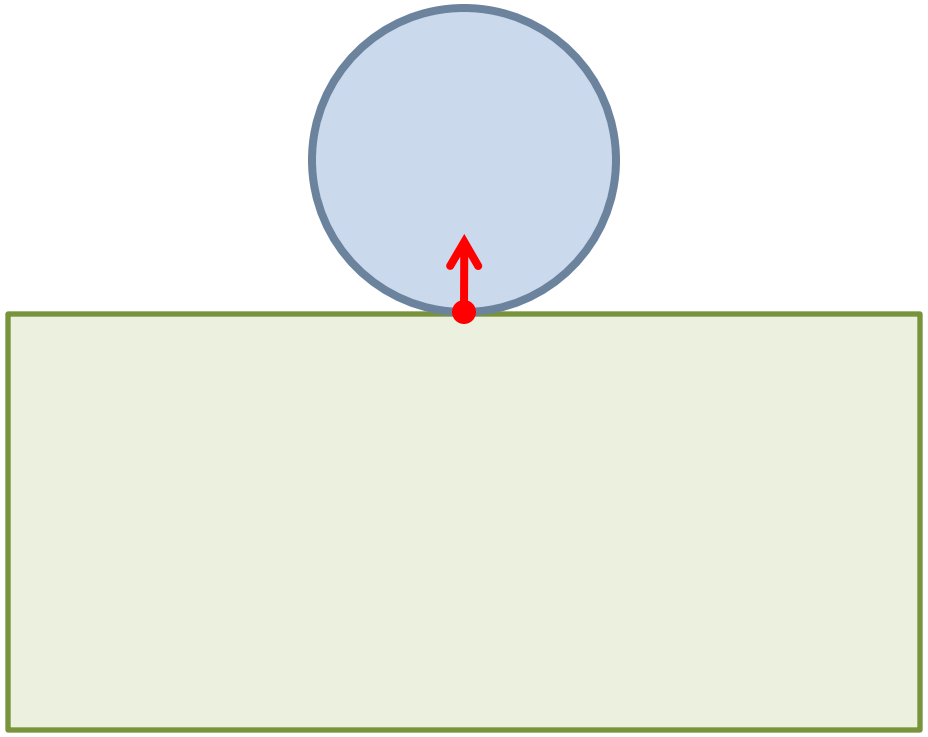
- a = b + n * d



Contact point structure

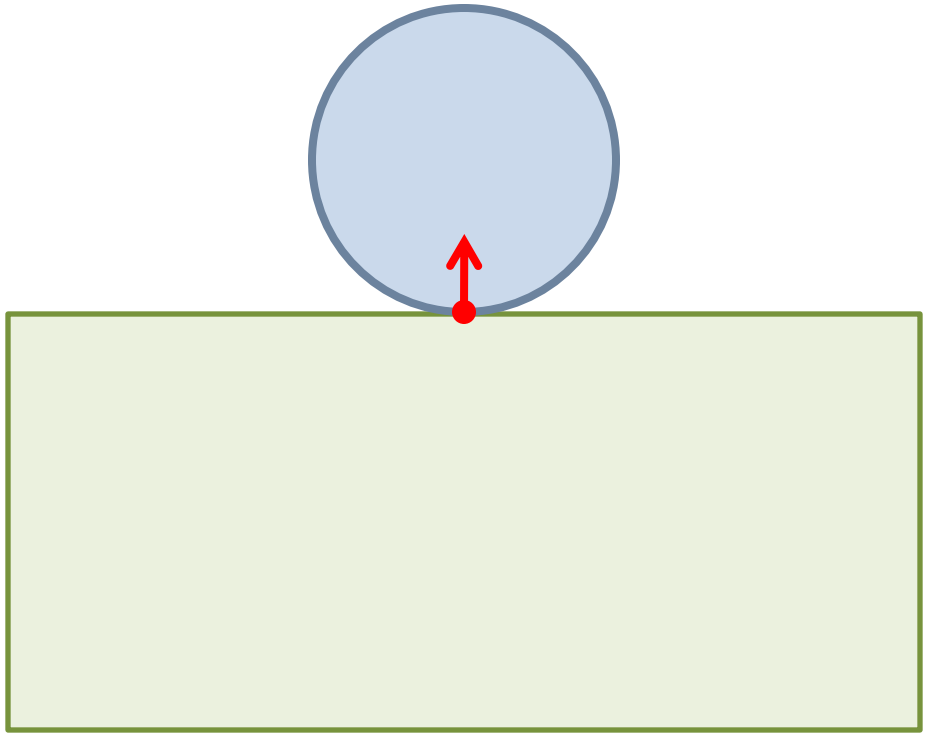
```
struct ContactPoint
{
   float     m_distance;
   Vector3   m_normalB;
   Vector3   m_pointOnB;
};
```

Single contact

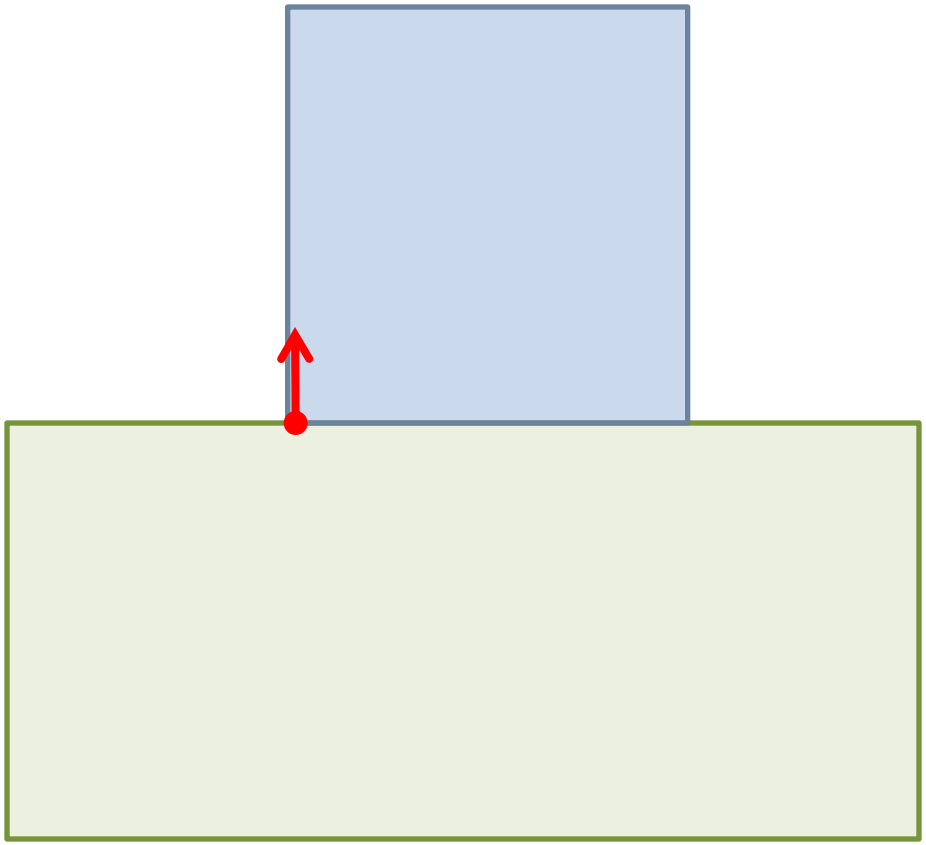


Single contact point

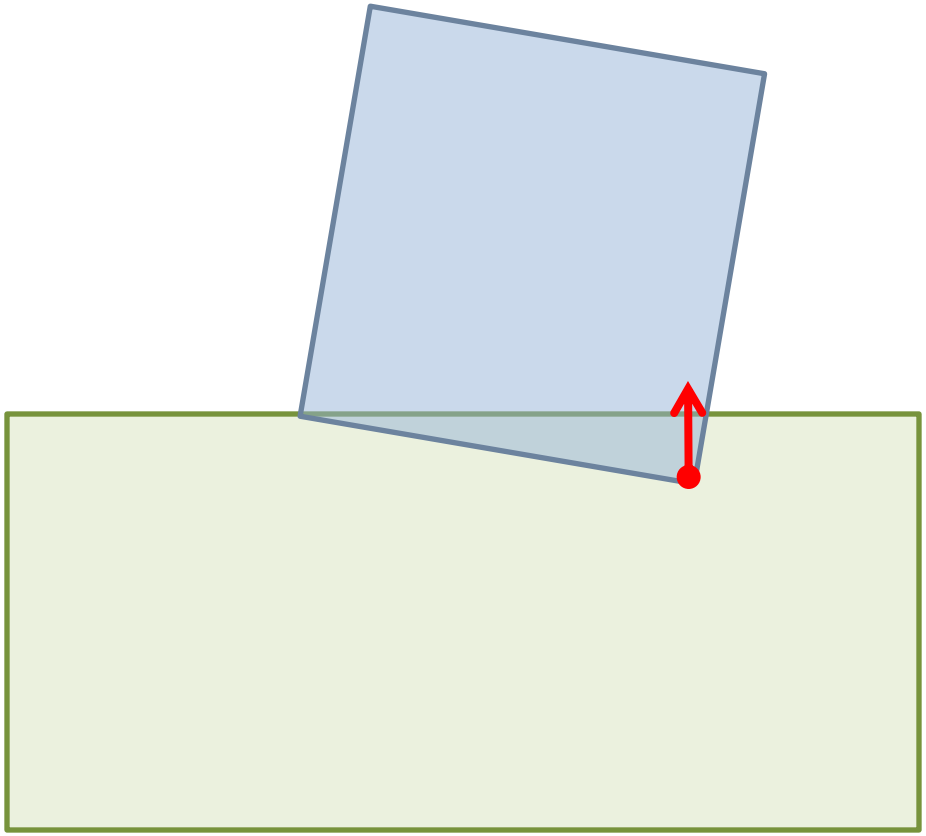
- Works fast and stable for simple cases



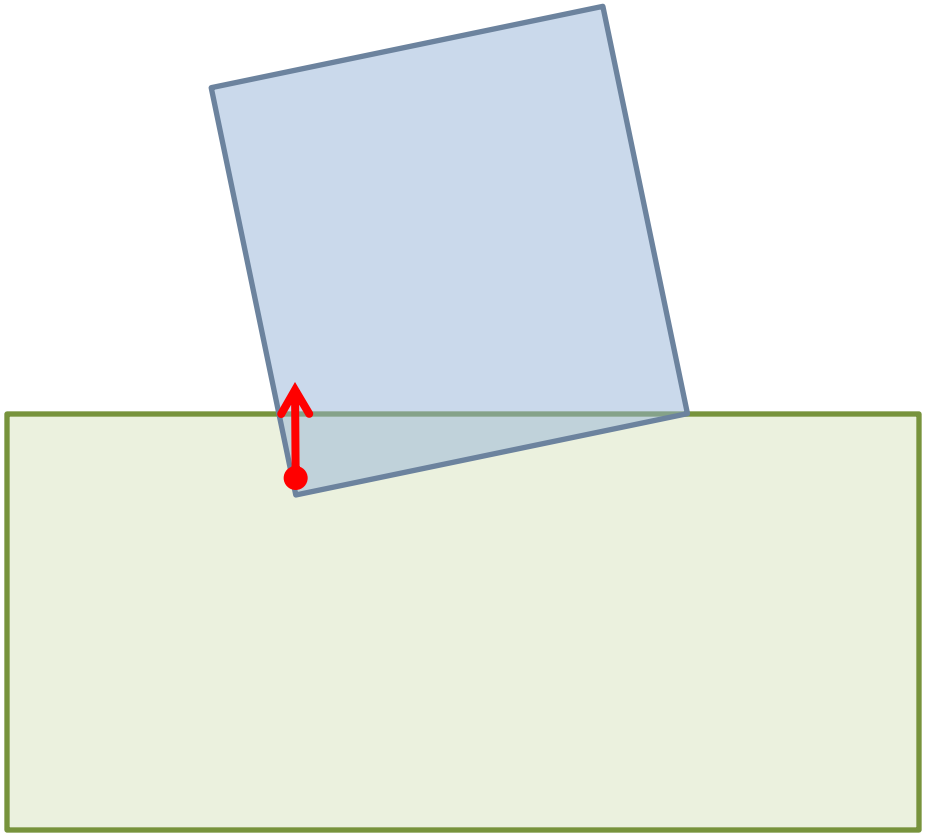
Single contact



Single contact



Single contact

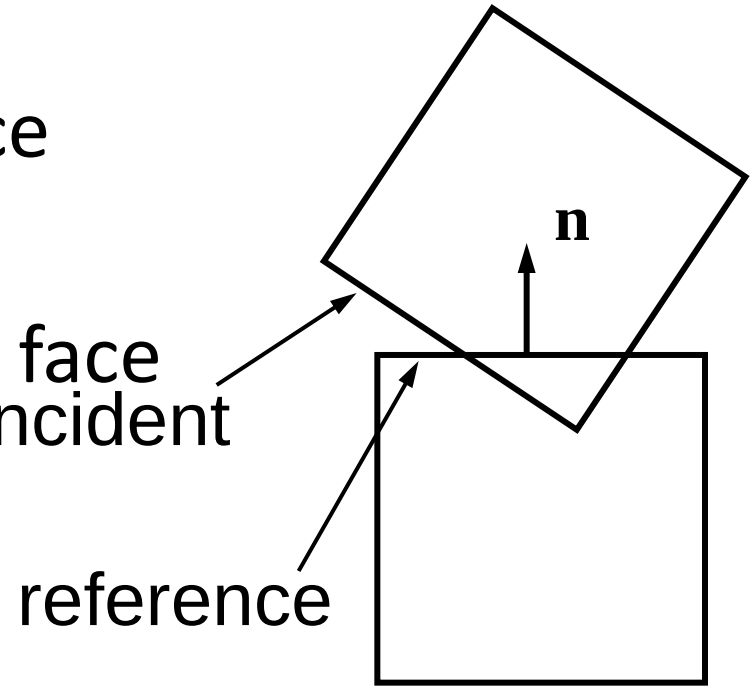


Contact generation

- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

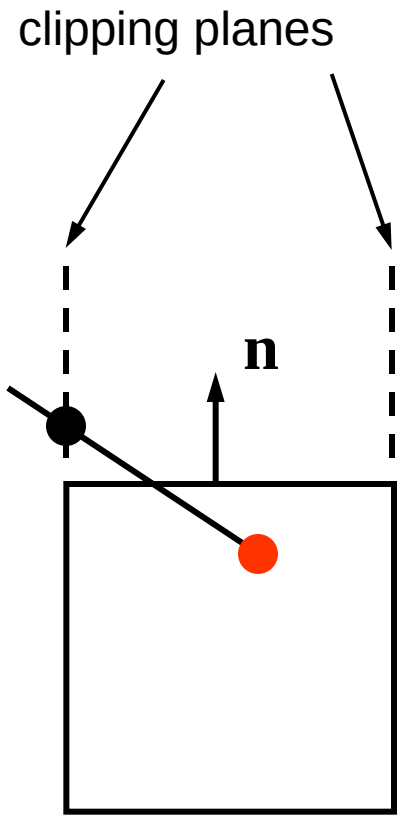
Box-Box Clipping Setup

- Identify reference face
- Identify incident face



Box-Box Clipping

- Clip incident face against reference face side planes (but not the reference face).
- Consider clip points with positive penetration.



Contact generation

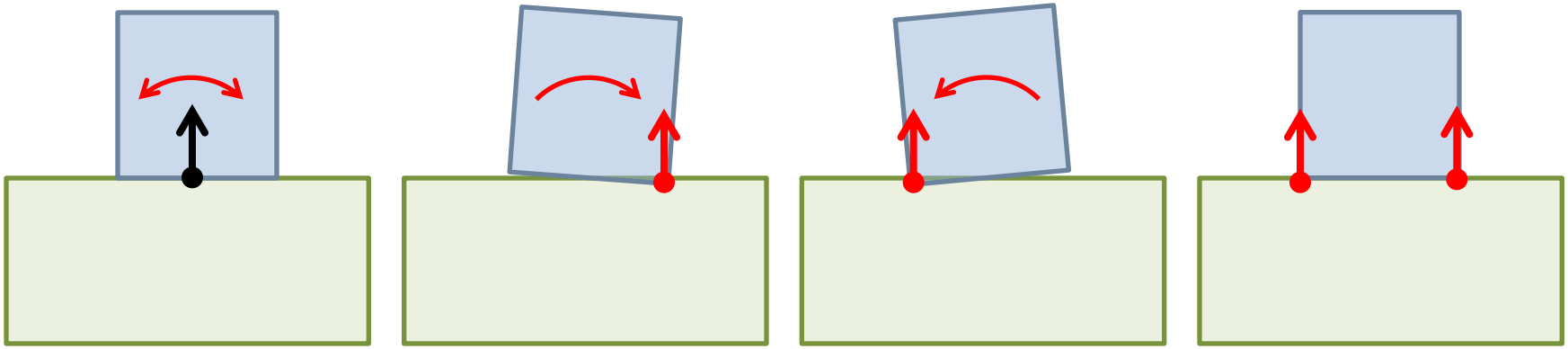
- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

Multiple points for general convex

- General convex might not have vertices/faces
- Compute multiple closest point samples
  - All “single shot” within this simulation step

Perturbing around normal

- Works well but costs additional GJK queries

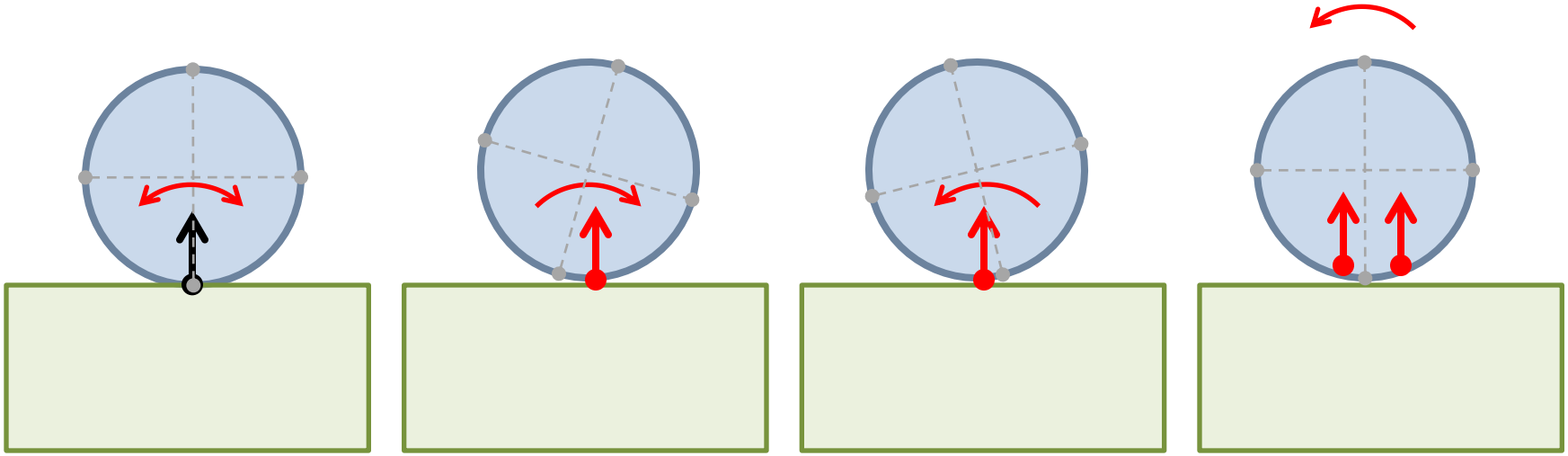


Perturbation pseudo code

```
contact = computeContact(shapeA,transA,shapeB,transB,…);
calcOrthogonalVectors(contact.normalB,ortho0,ortho1);
Matrix3x3 R_p = MatrixFromAxisAngle (ortho0, perturbationAngle);
float angle = 360 / number of iterations;
for (i = 0 ; i< number of iterations;i++)
{
   Matrix3x3 Rn = MatrixFromAxisAngle(contact.normalB,angle*i);
   perturbedA = Rn.inverse() * R_p * Rn * transA.R();
   computeContact(shapeA,perturbedA,shapeB,transB,...);
   updateContactDistance(transA,transB);
}
```

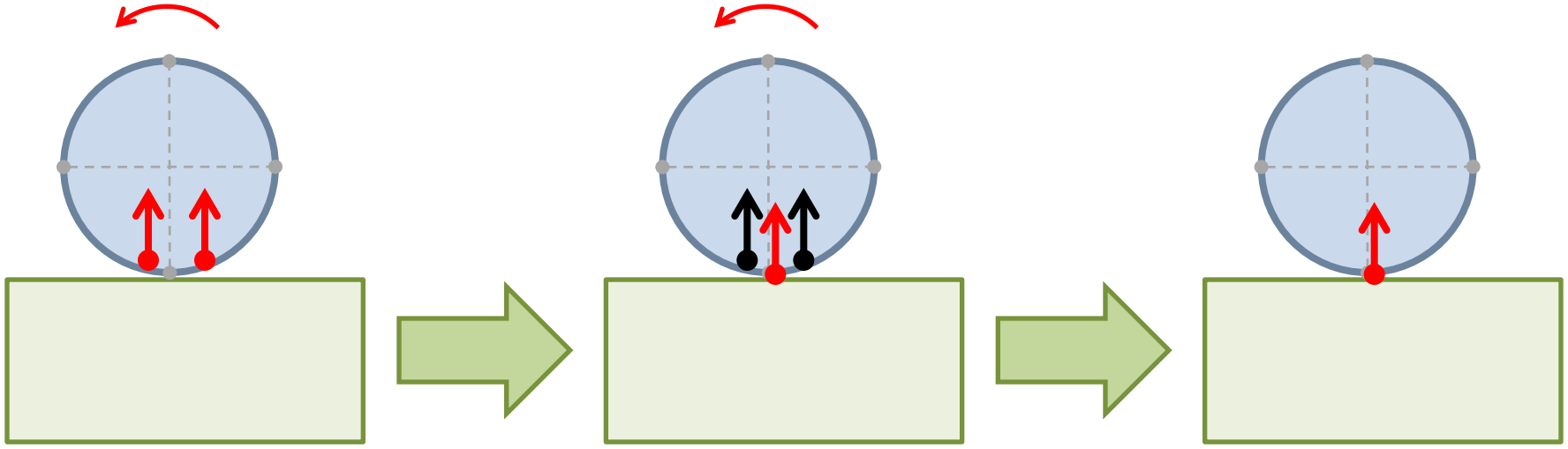
Perturbing around normal

- Issue with round shapes: they start rolling

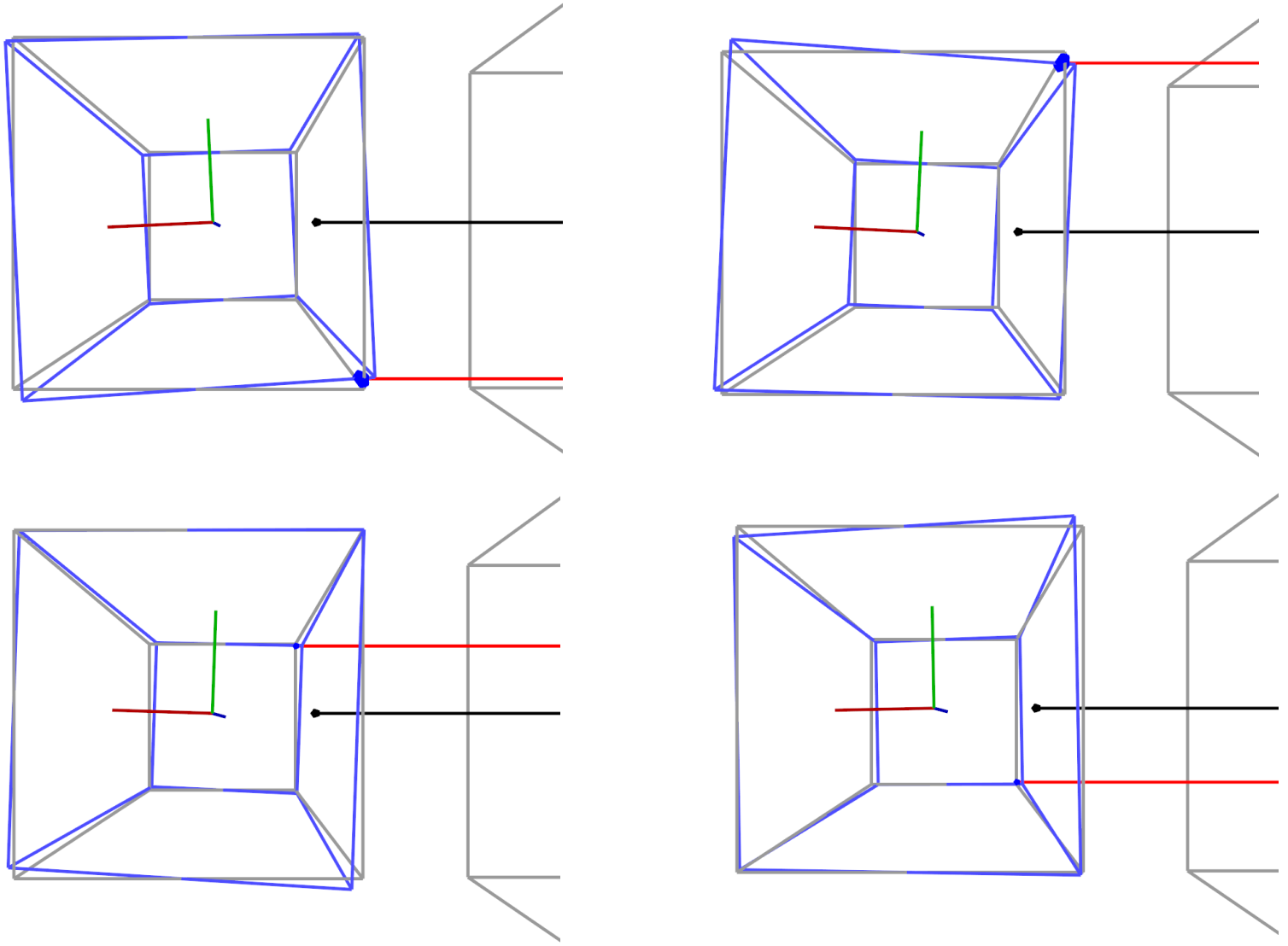


Perturbing around normal

- Replace contact at actual position
- Keep perturbation small
- Last resort: apply damping or skip round shapes



Perturbating in 3D



DEMO!!!

Contact generation

- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

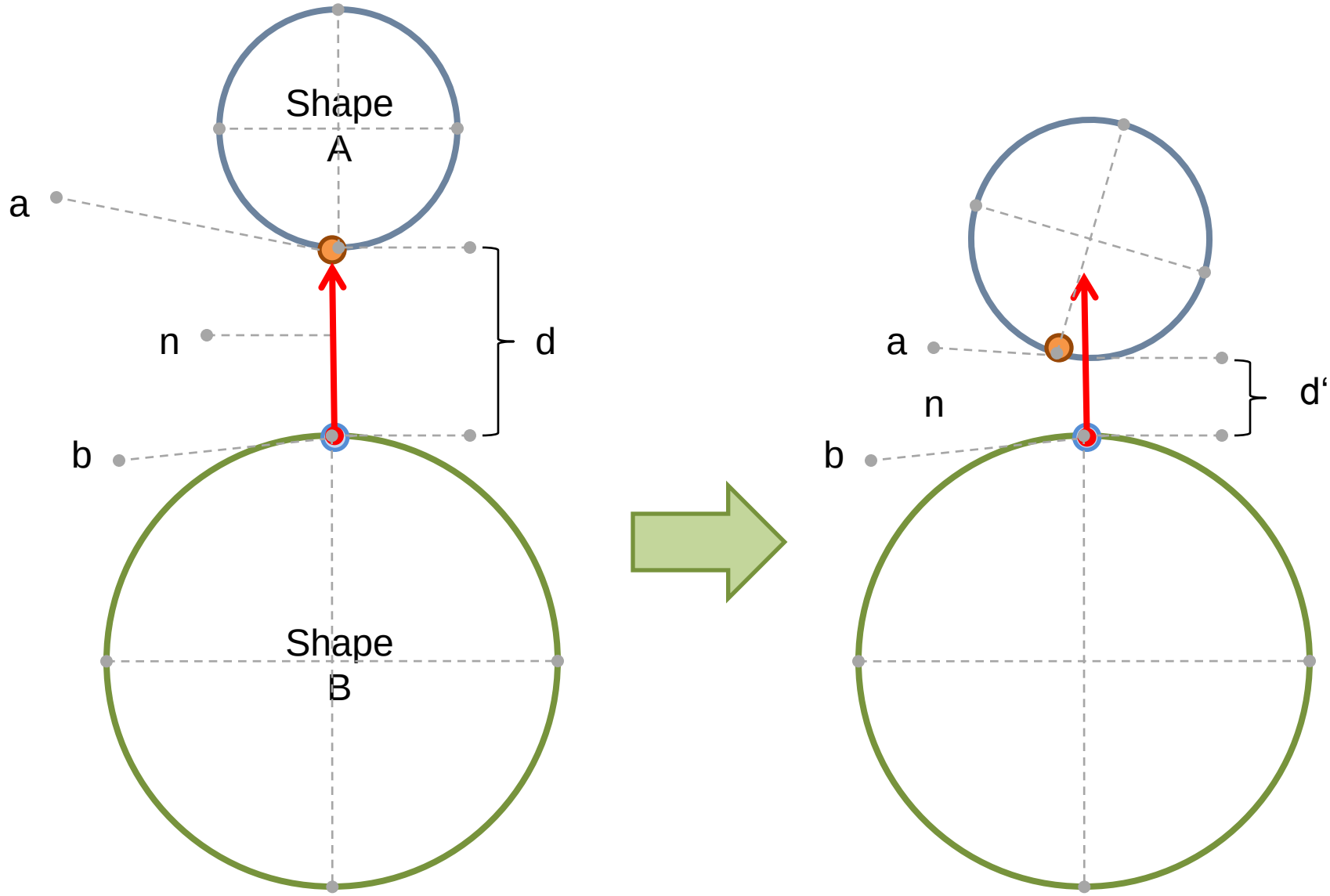
Incremental contact caching

- Add a single point at a time to small cache
- Refresh cache, update or remove points

Adding points to cache

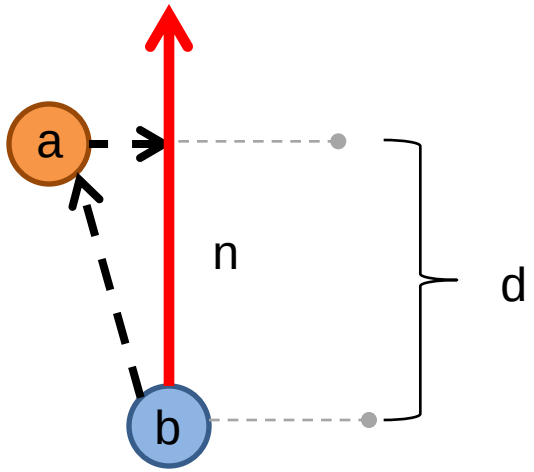
- Check for duplicate points
  - Use feature id or distance tolerance
- Reduce points when more than 4 points
  - Always keep point with deepest penetration
  - Try to maximize area

Cache local closest points

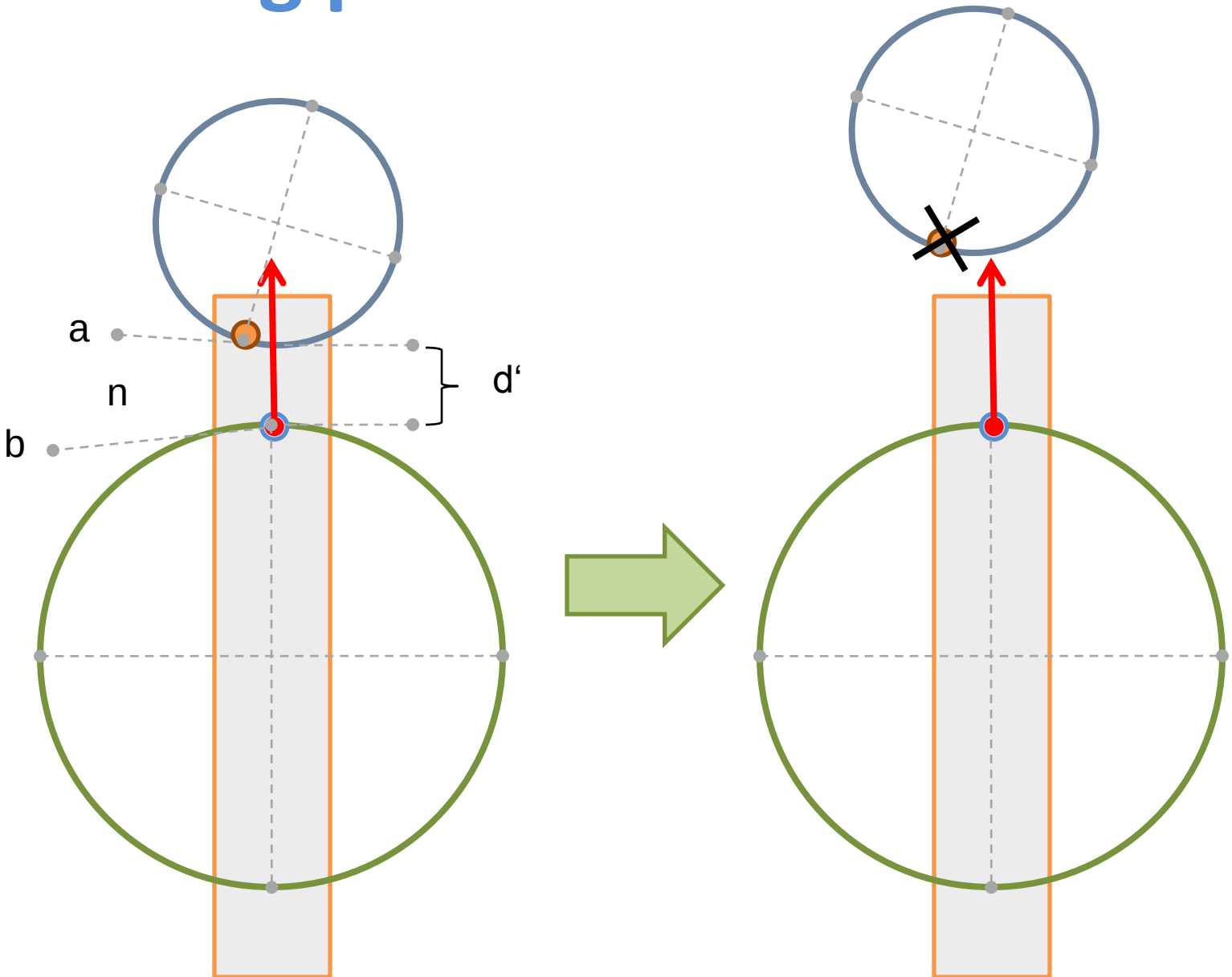


Update contact distance

```
worldPosA = transA(localPosA);
worldPosB = transB(localPosB);
distance = (worldPosA-worldPosB).dot(worldNormalB);
```



Removing points

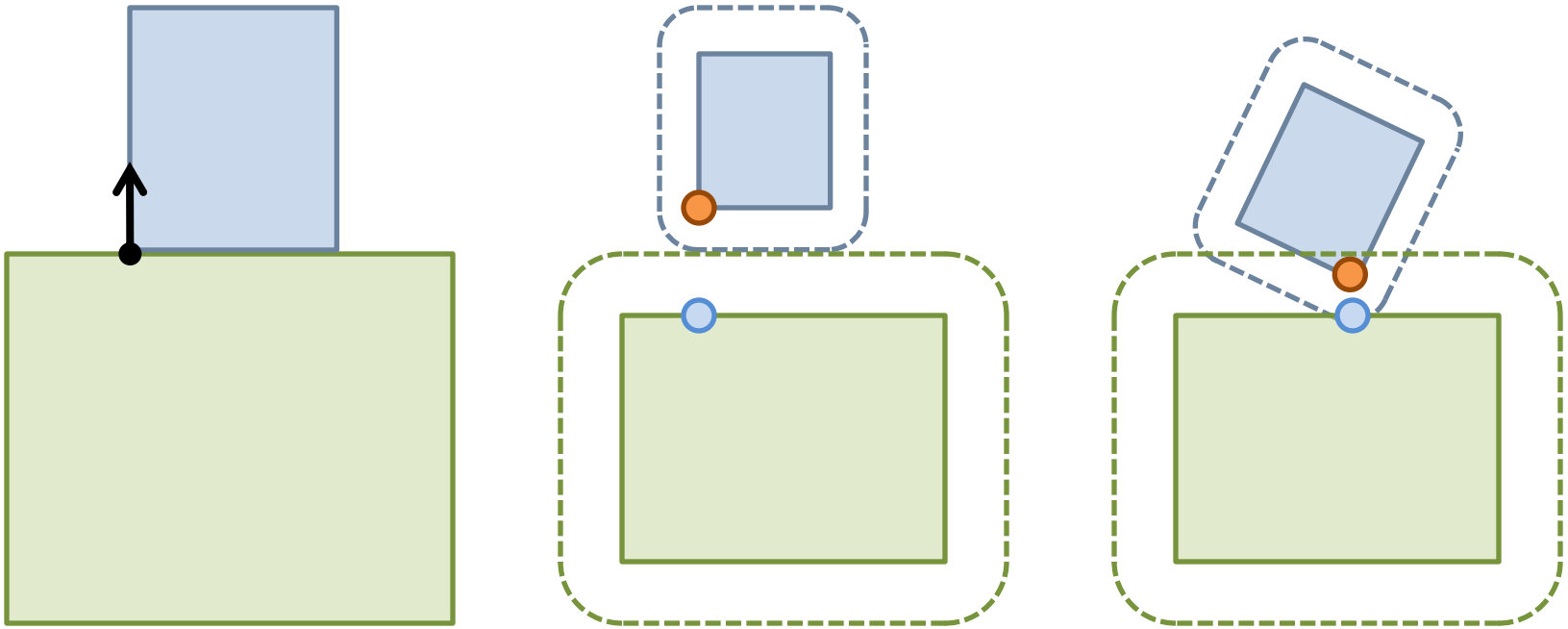


Hybrid method

- Single shot manifold to build a full cache
- Only add a single point to a full cache
- Google for btPersistentManifold

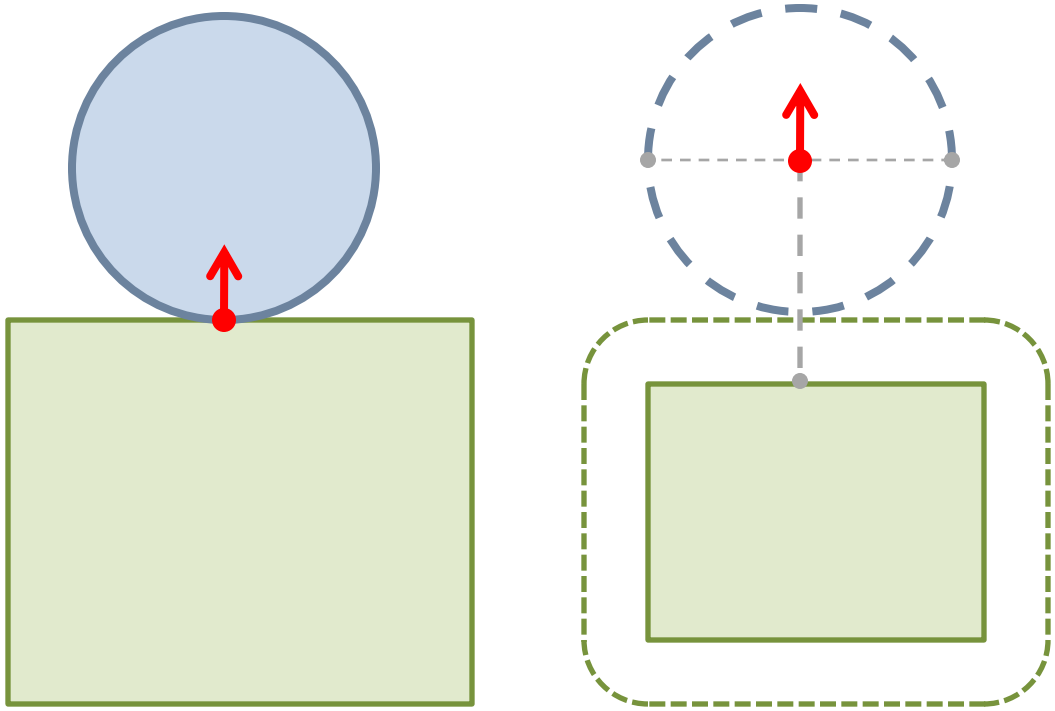
Collision margins

- GJK doesn’t work in penetrating cases
  - and penetration depth calculation is a bit slower

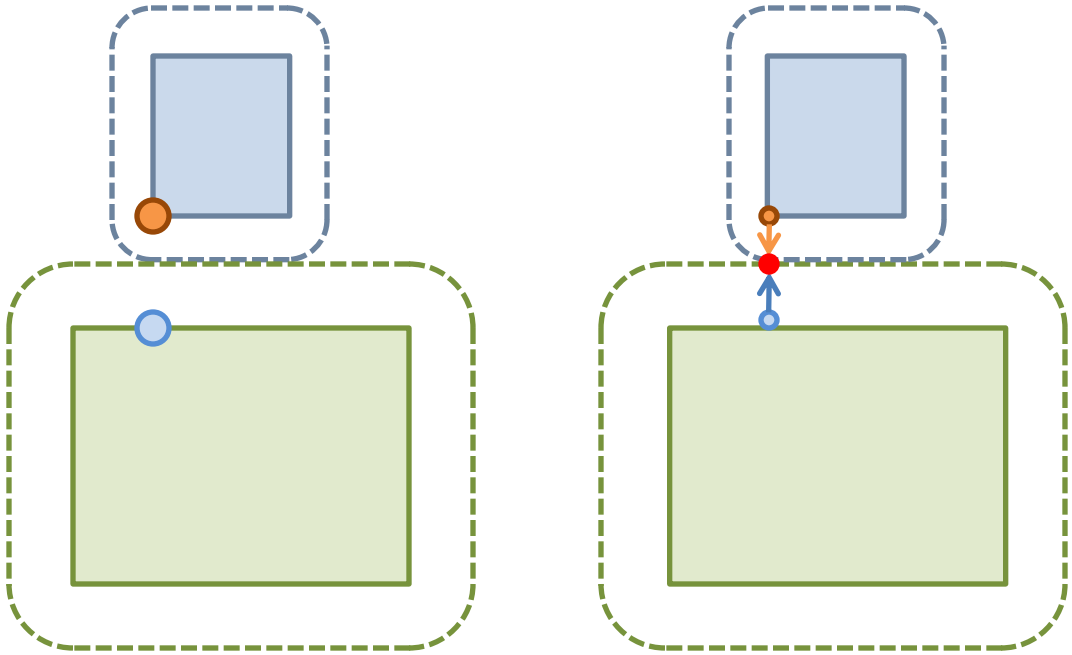


Collision margins

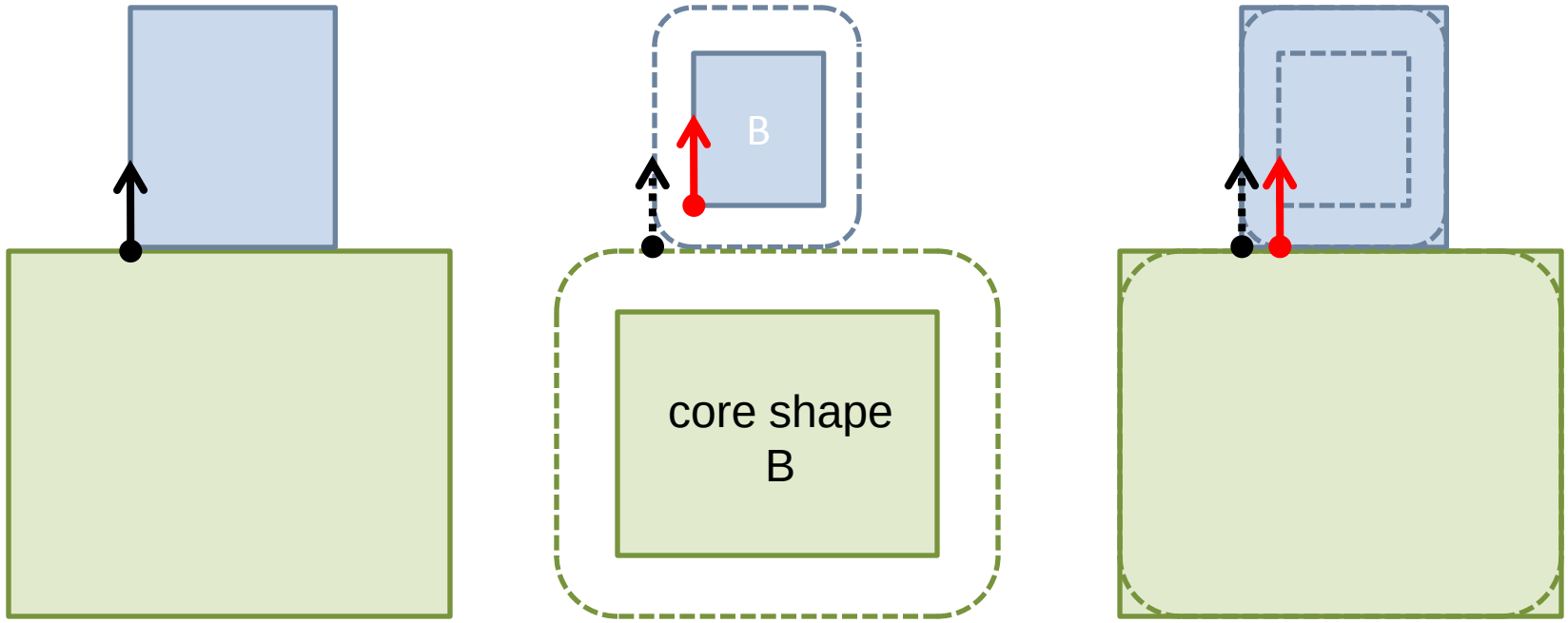
- Sphere can be a point with radius as margin



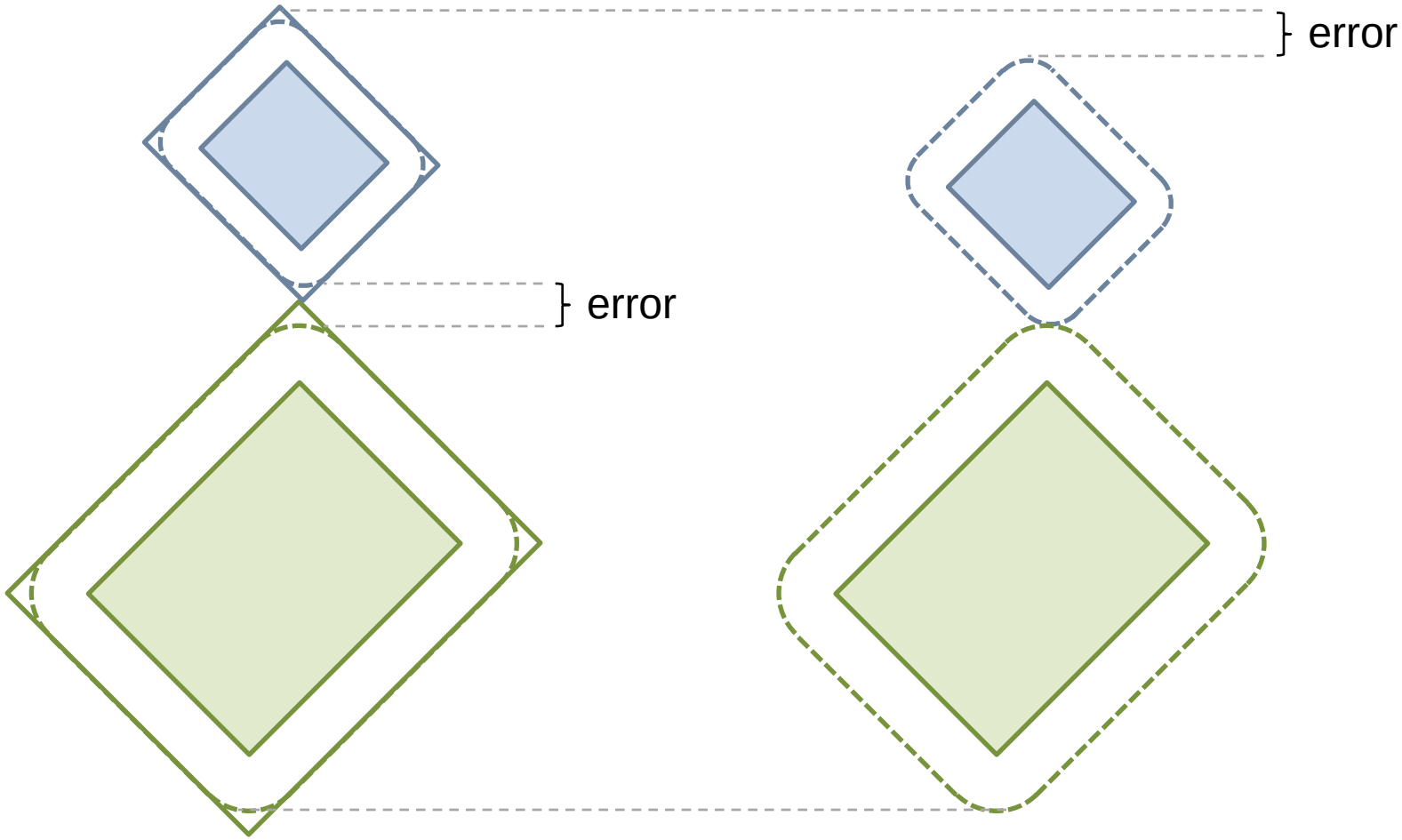
Compensate for margins



Collision margin approximation



Collision margin approximation

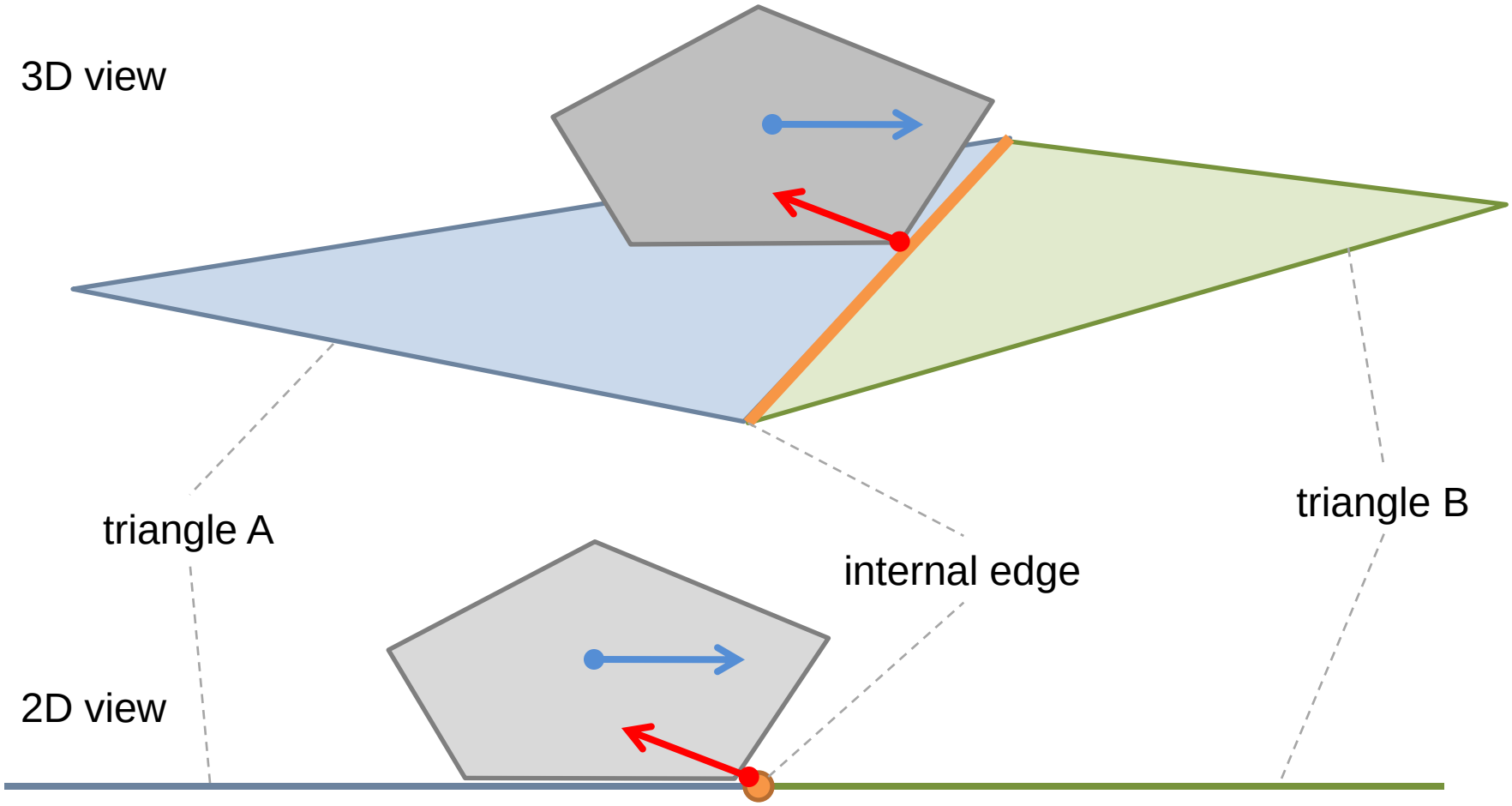


DEMO!!!

Contact generation

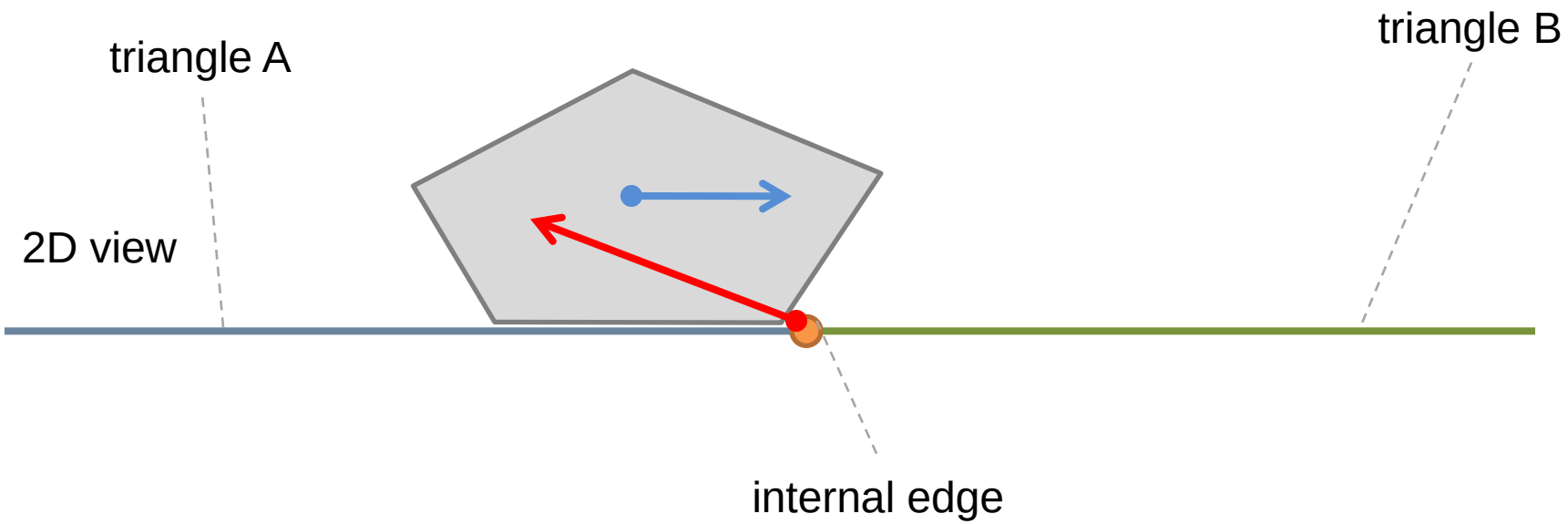
- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

Internal edge collisions



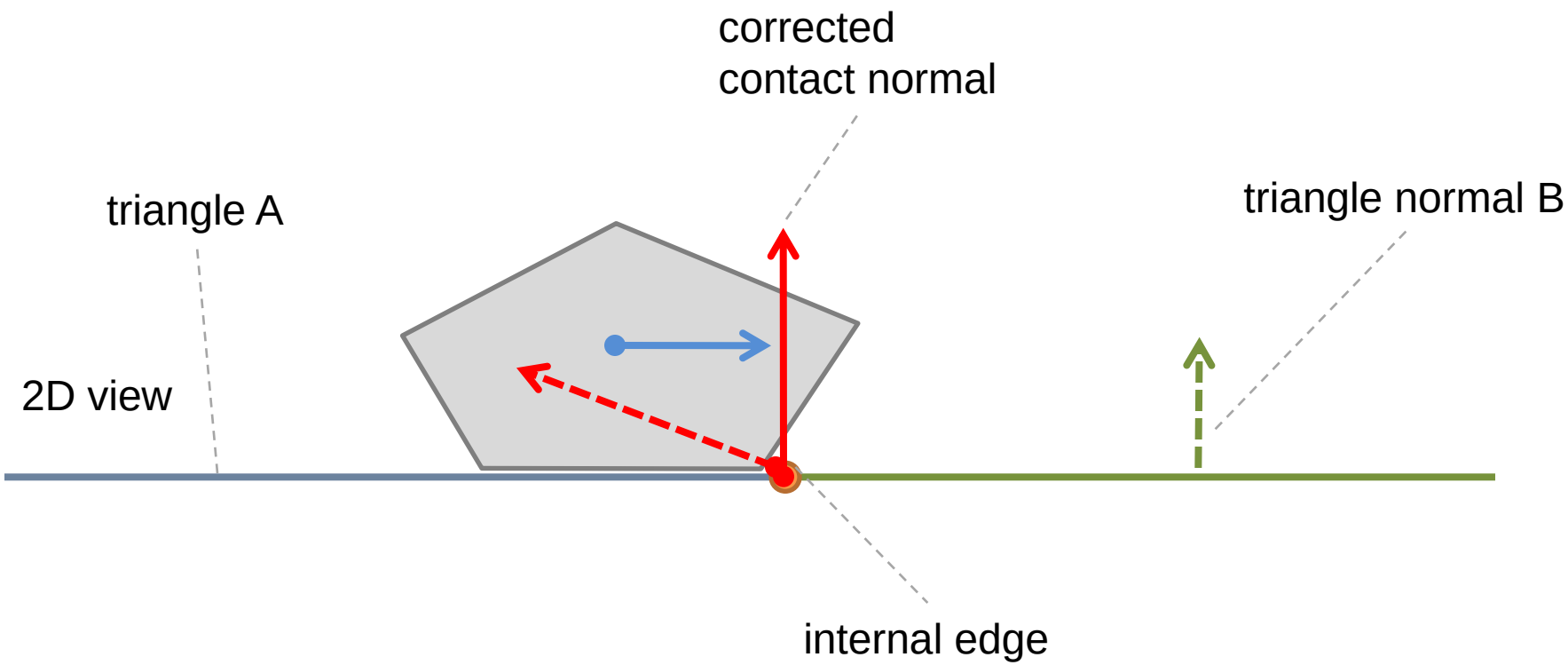
Internal edge collisions

- Object is colliding against triangle B
- Contact normal is pointing against velocity



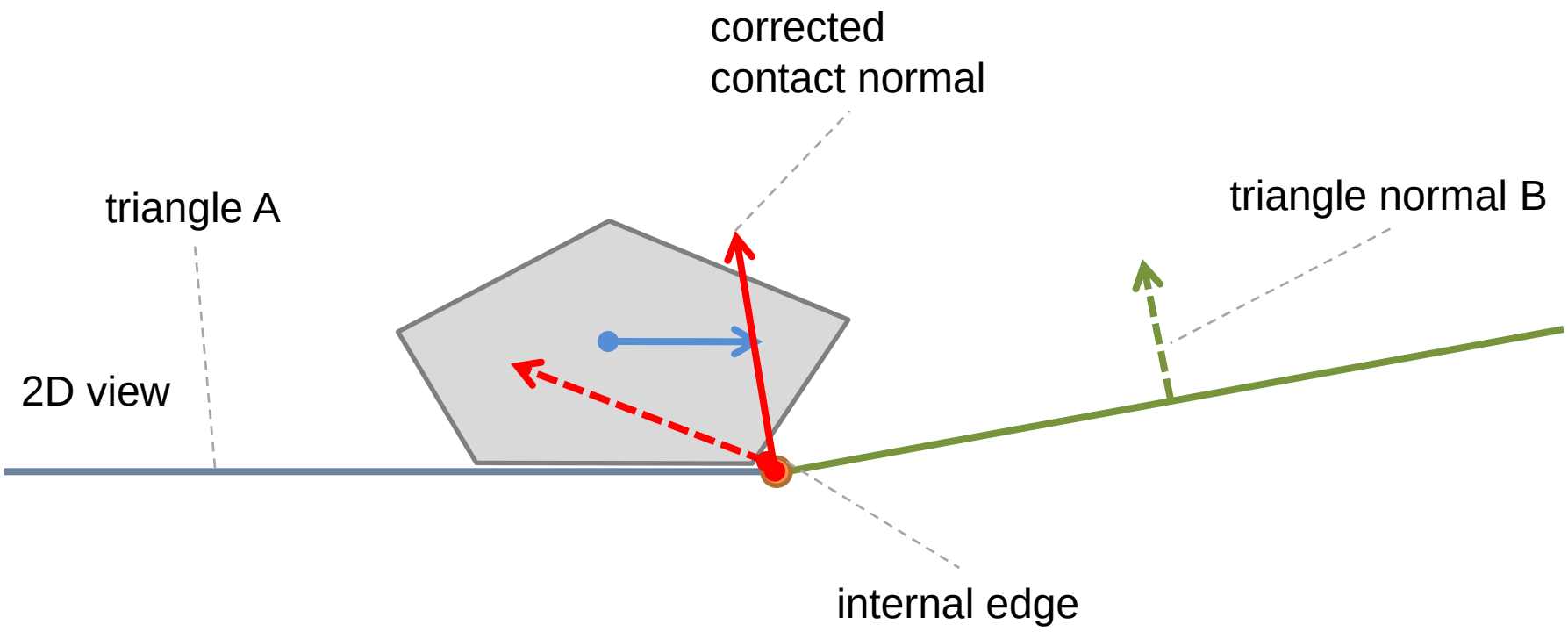
Solution 1: triangle normal

- Works well for flat connected triangles



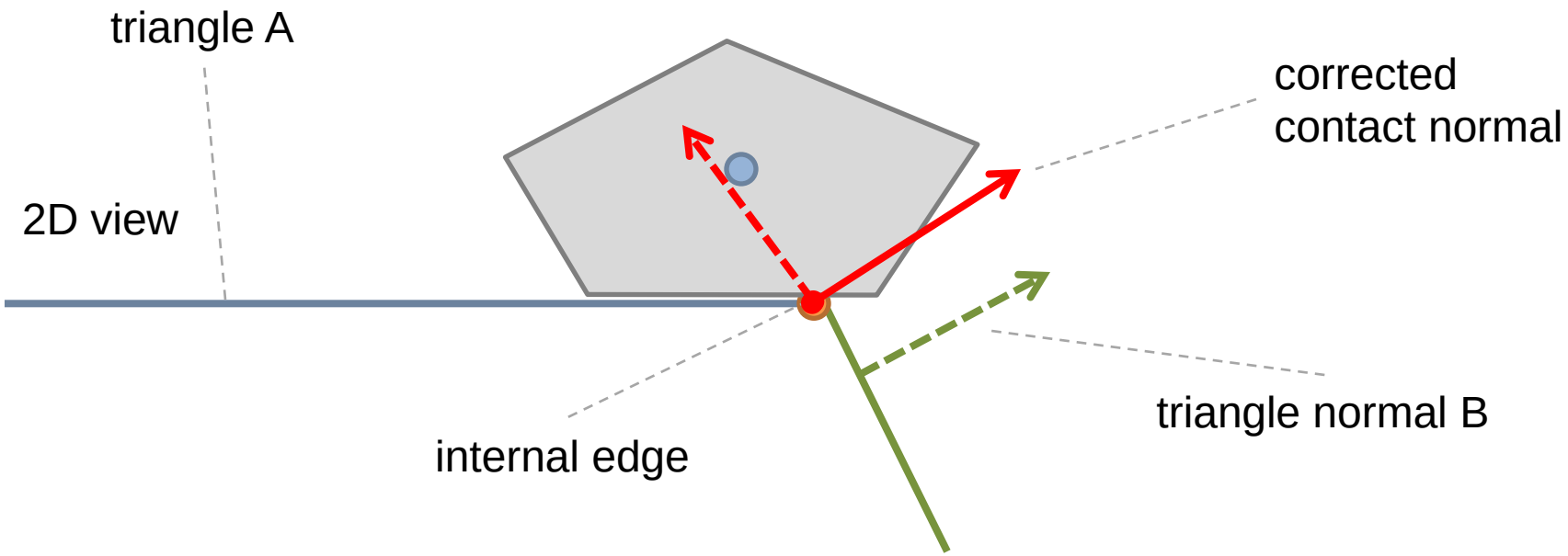
Solution 1: triangle normal

- Works well for concave edges



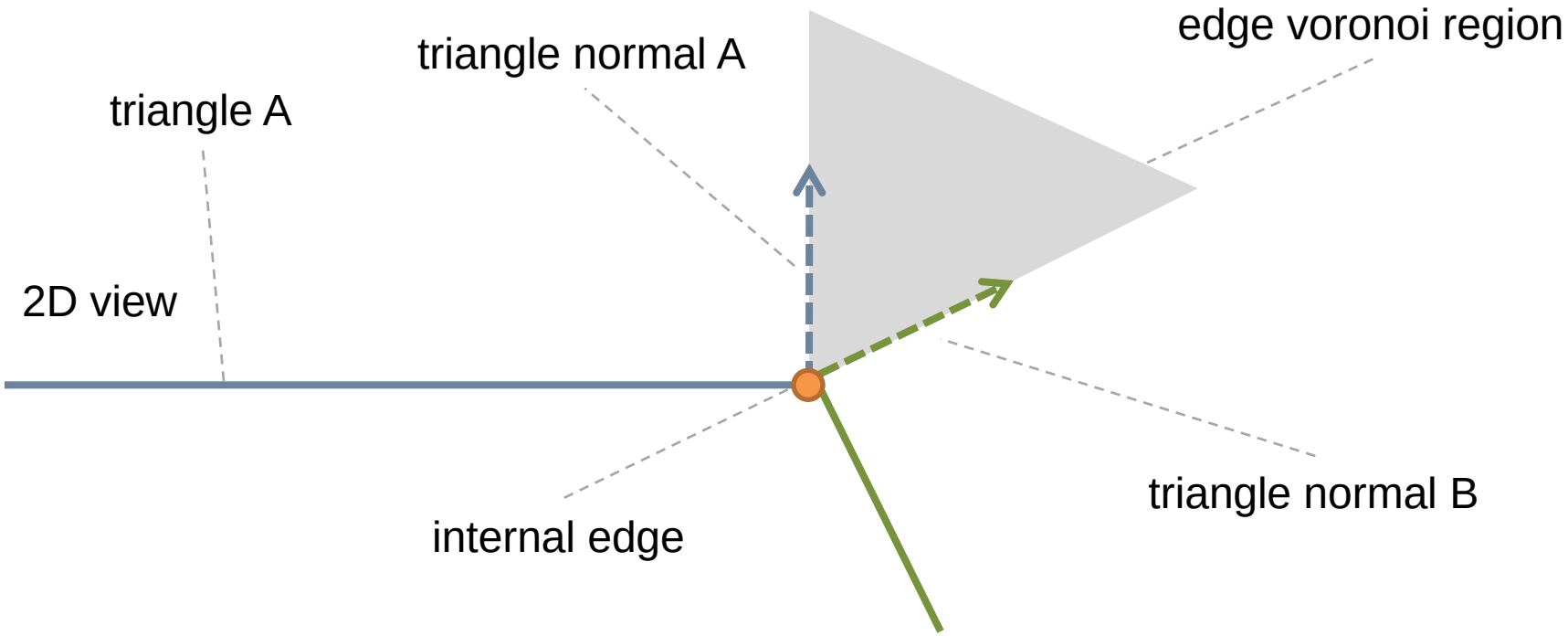
Cliff problem

- Object can be pushed of a cliff



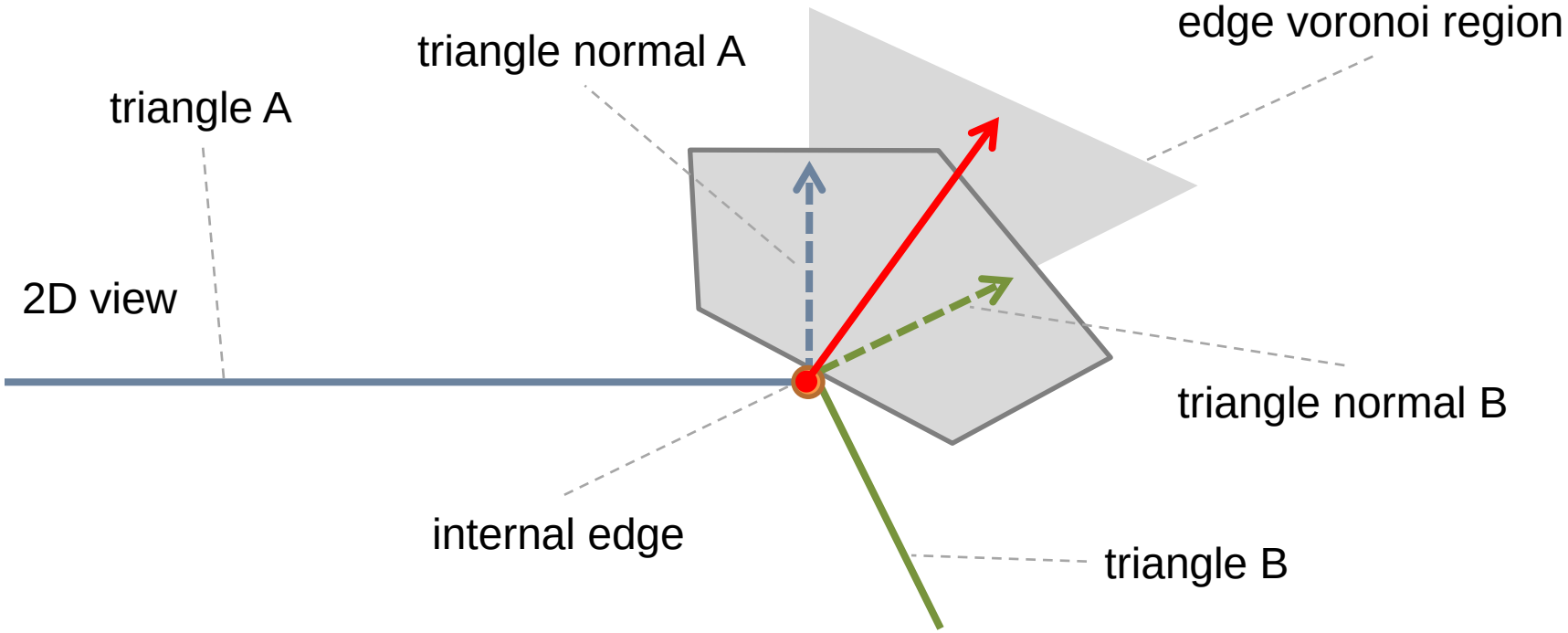
Solution 2: use voronoi regions

- Only adjust normal outside voronoi region



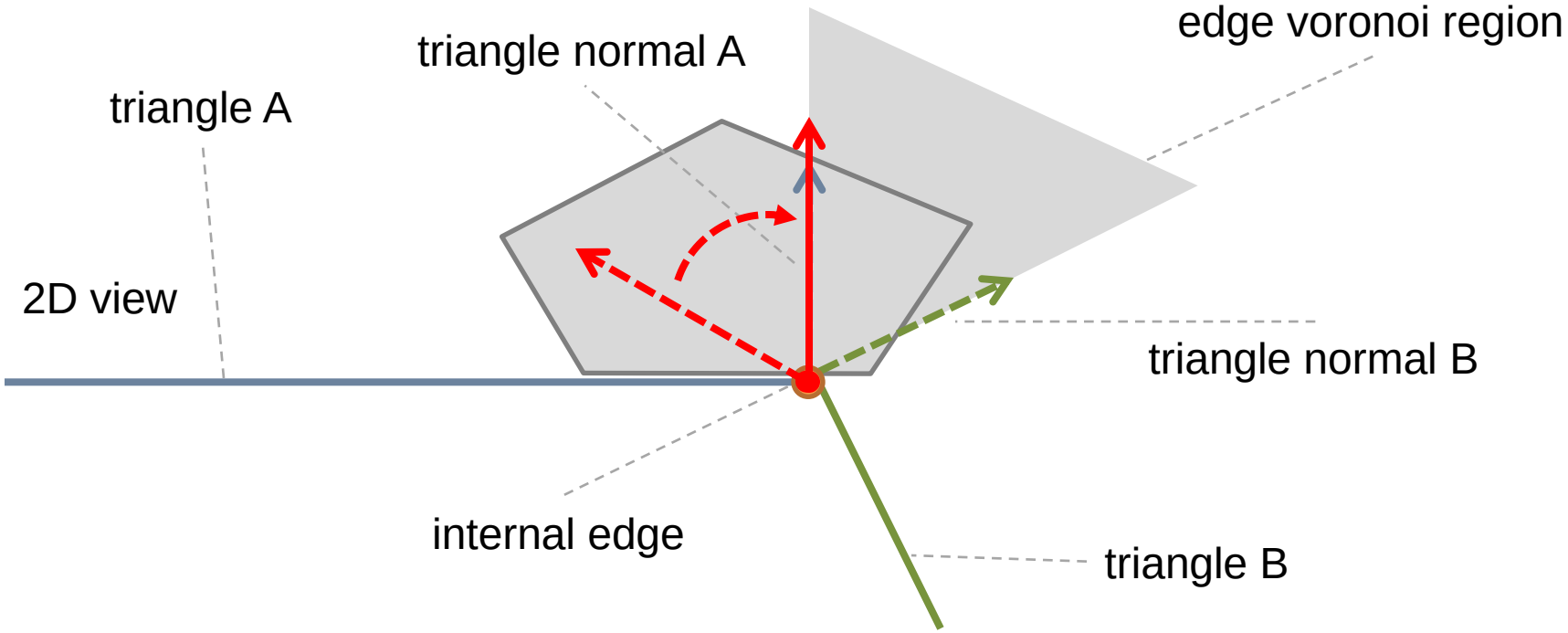
Solution 2: use voronoi regions

- No correction needed



Solution 2: use voronoi regions

- Rotate normal towards voronoi region



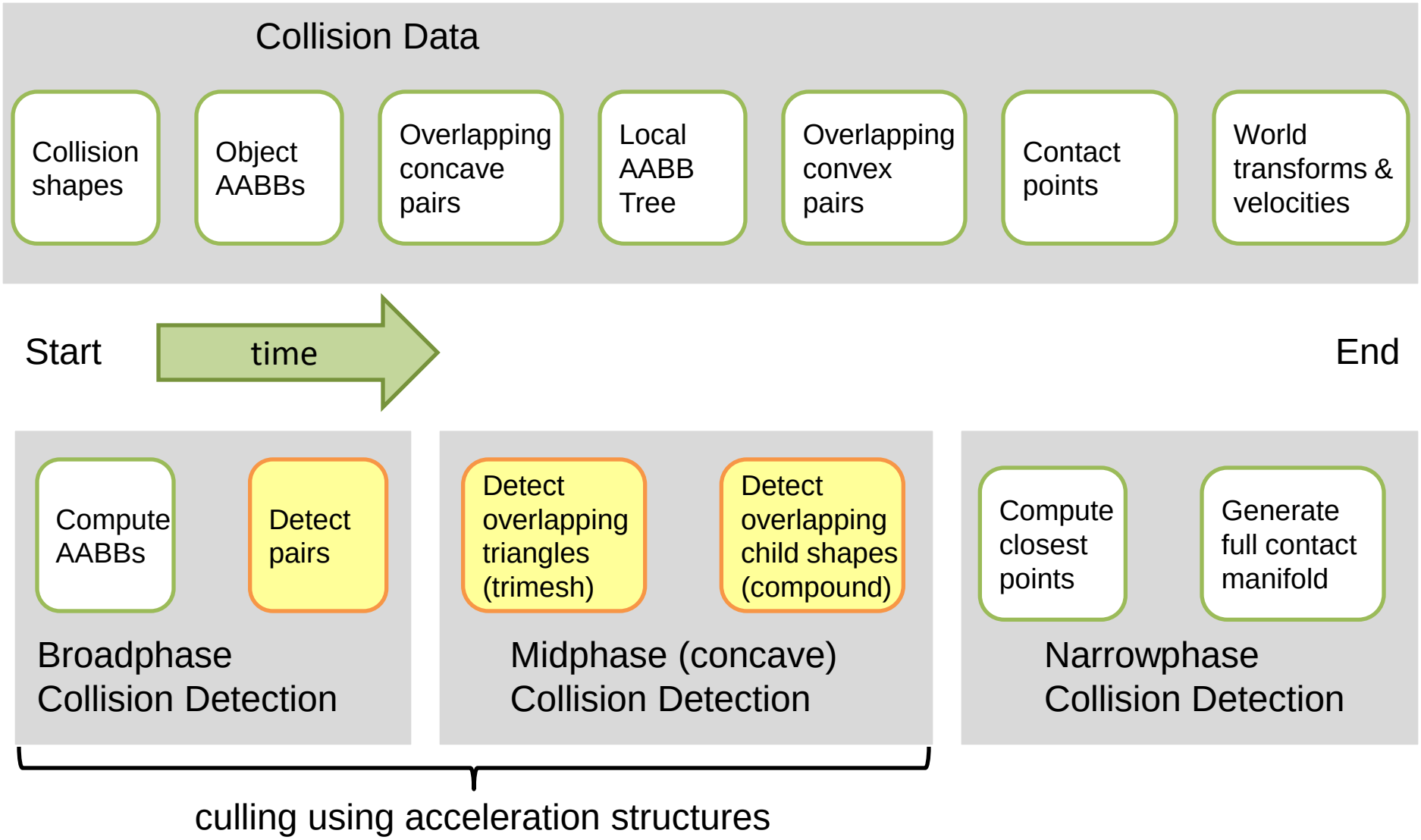
Implementation

- Google for btInternalEdgeUtility

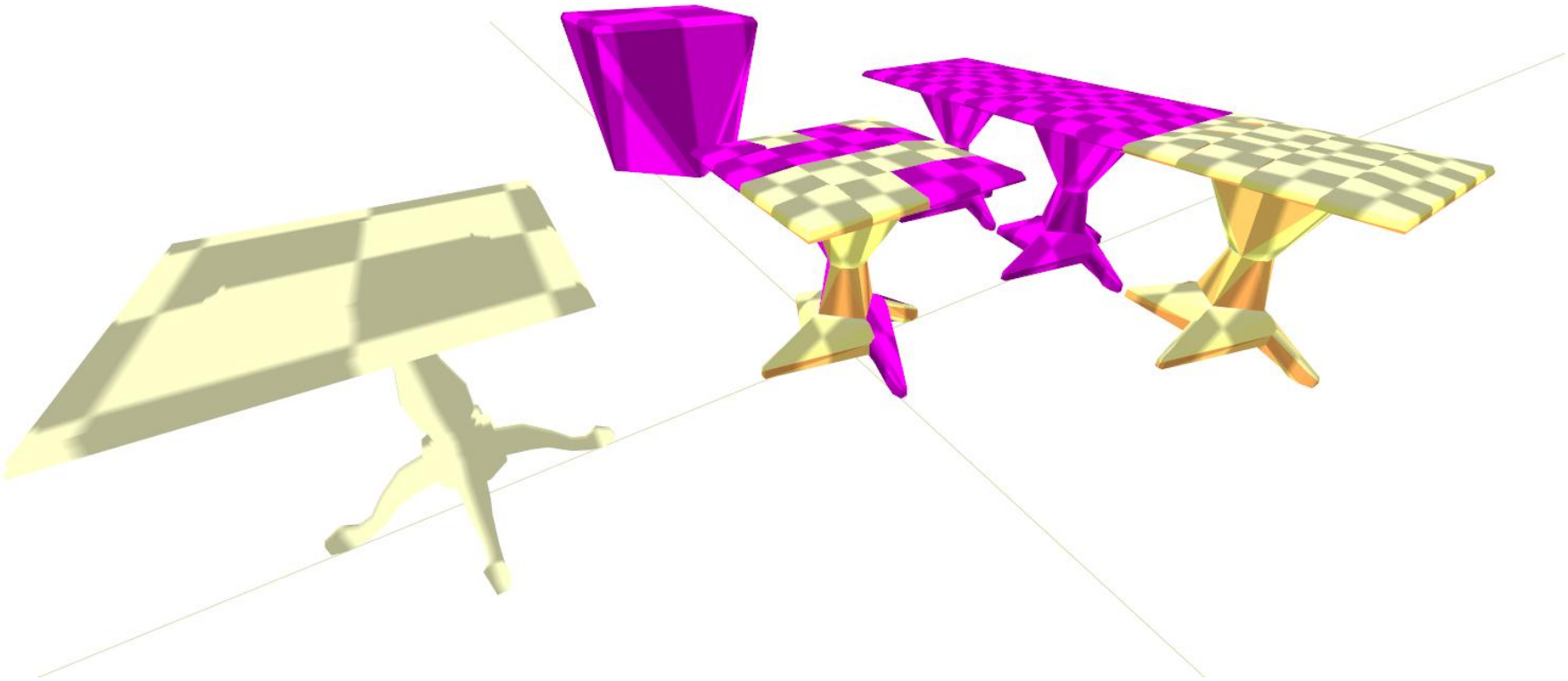
Contact generation

- Pipeline overview
- A single contact point
- Contact clipping
- Multiple contact points using perturbation
- Persistent contact caching
- Internal edges and contact normals
- Dynamic aabb tree acceleration structure

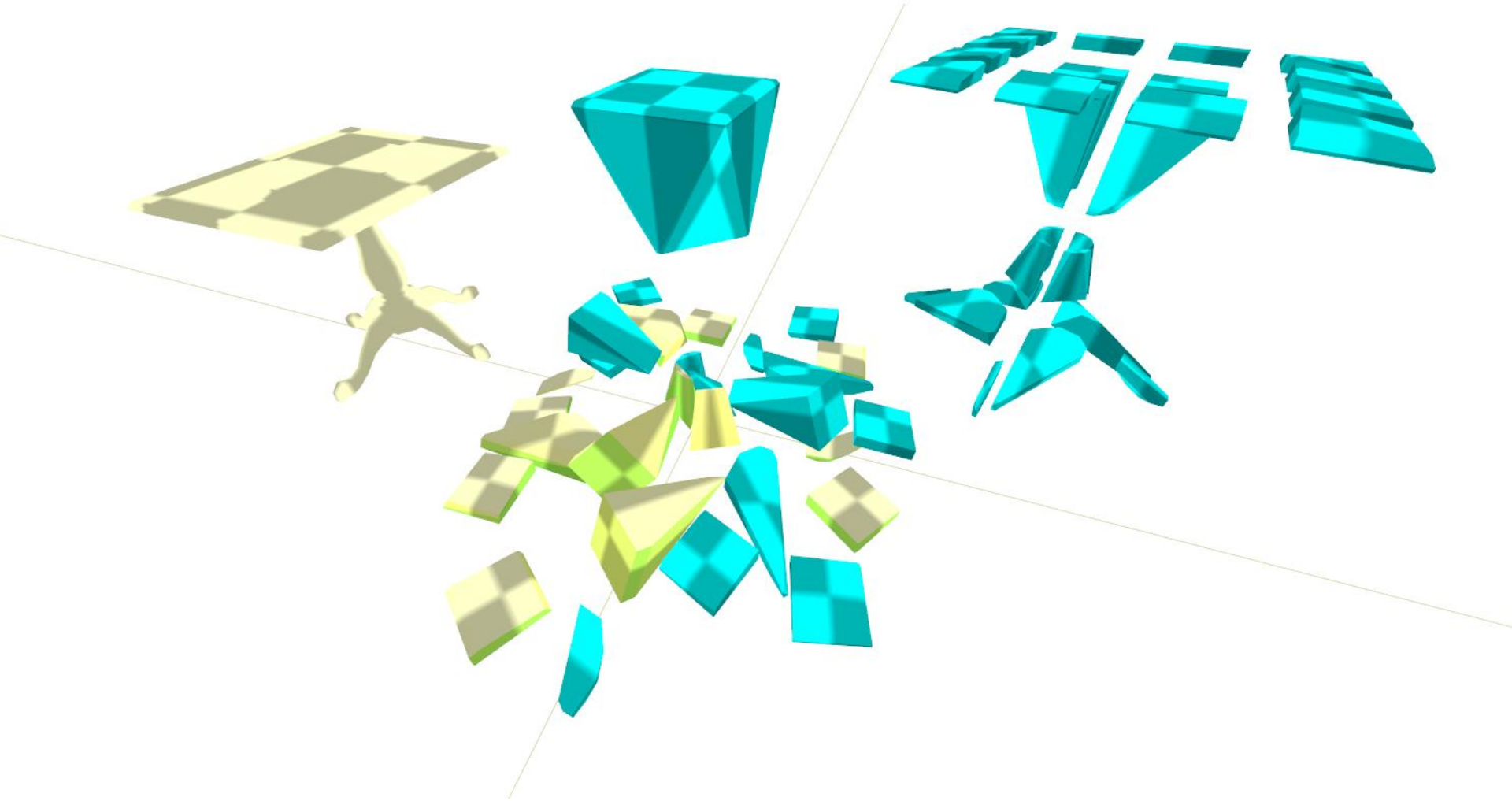
Collision Detection Pipeline



Concave shapes

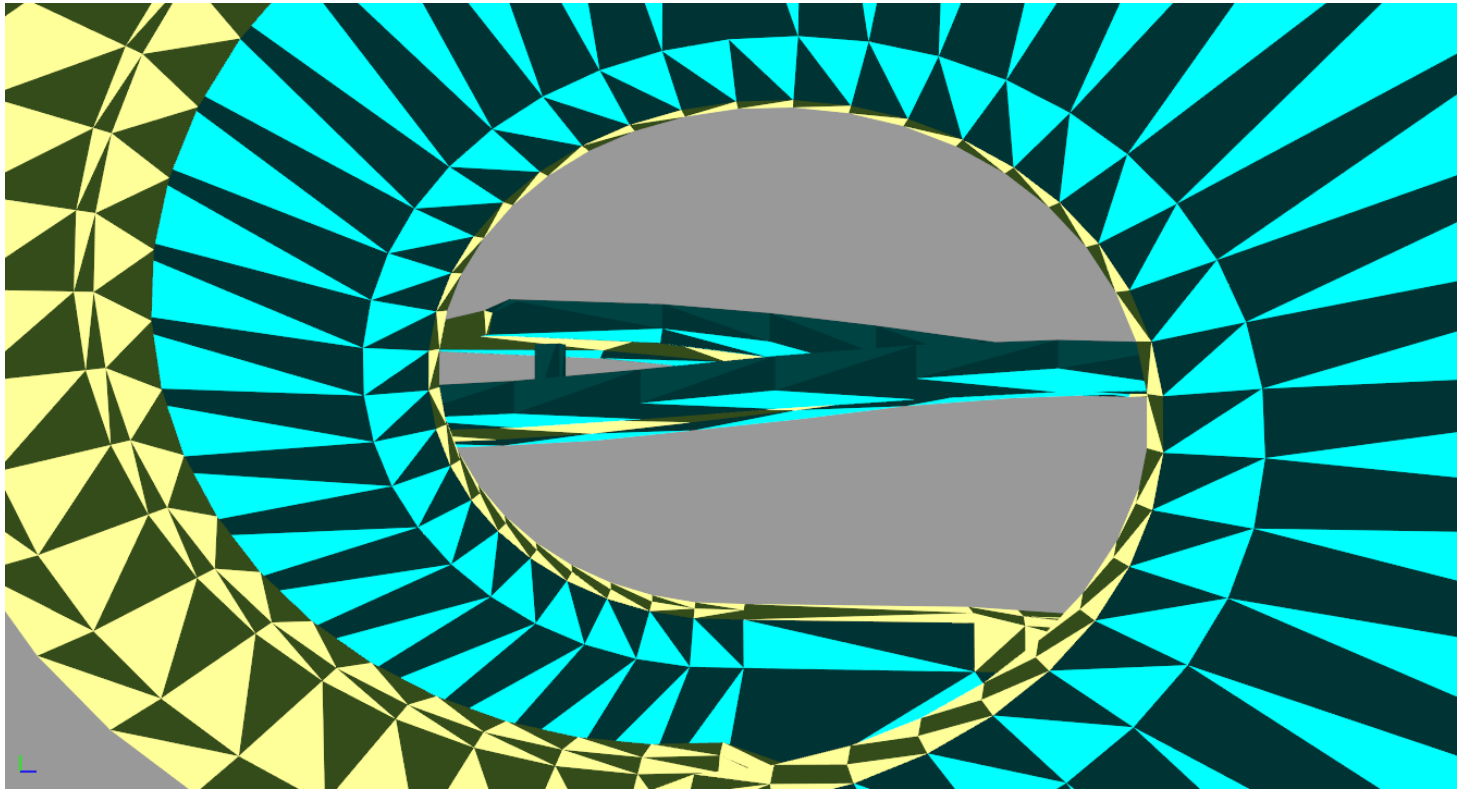


Convex decomposition



Concave triangle mesh

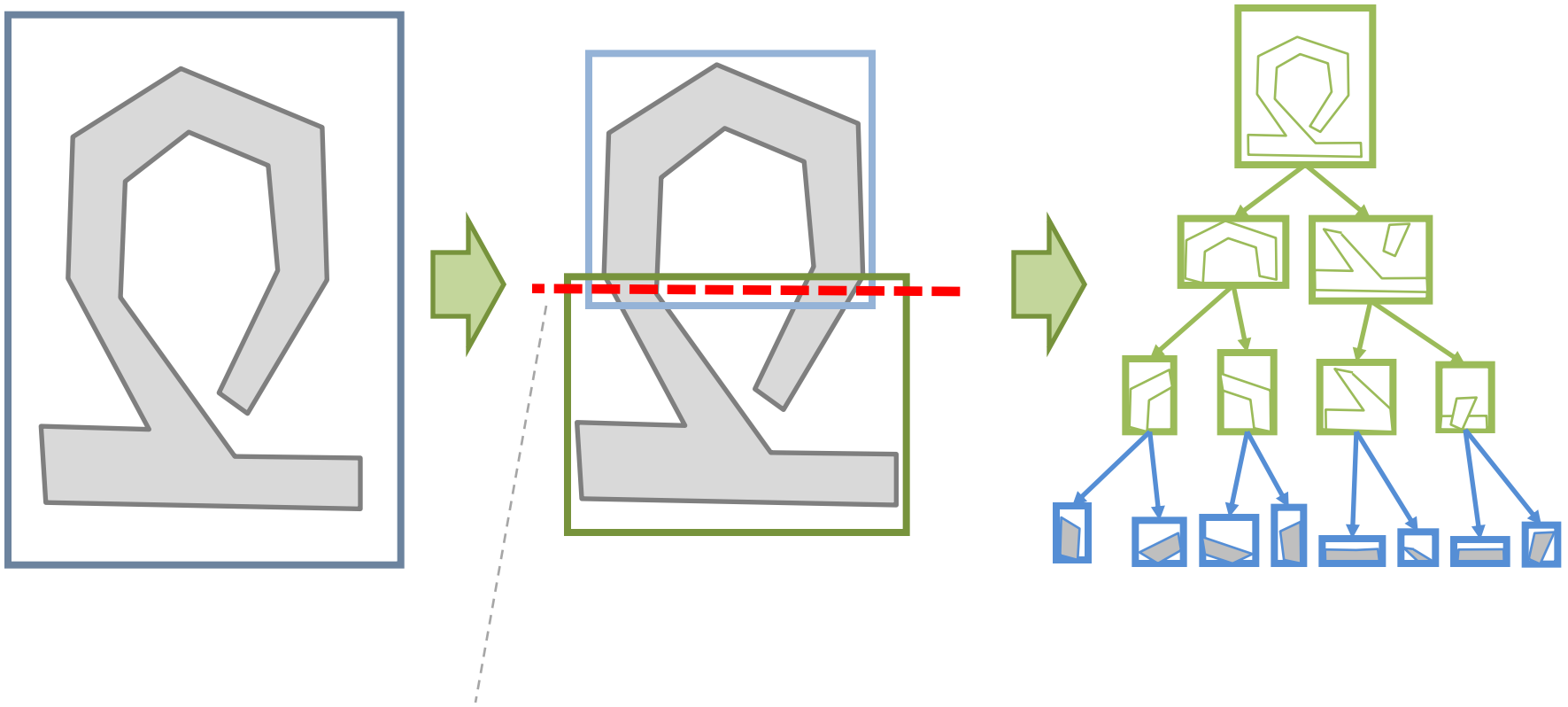
Concave triangle mesh



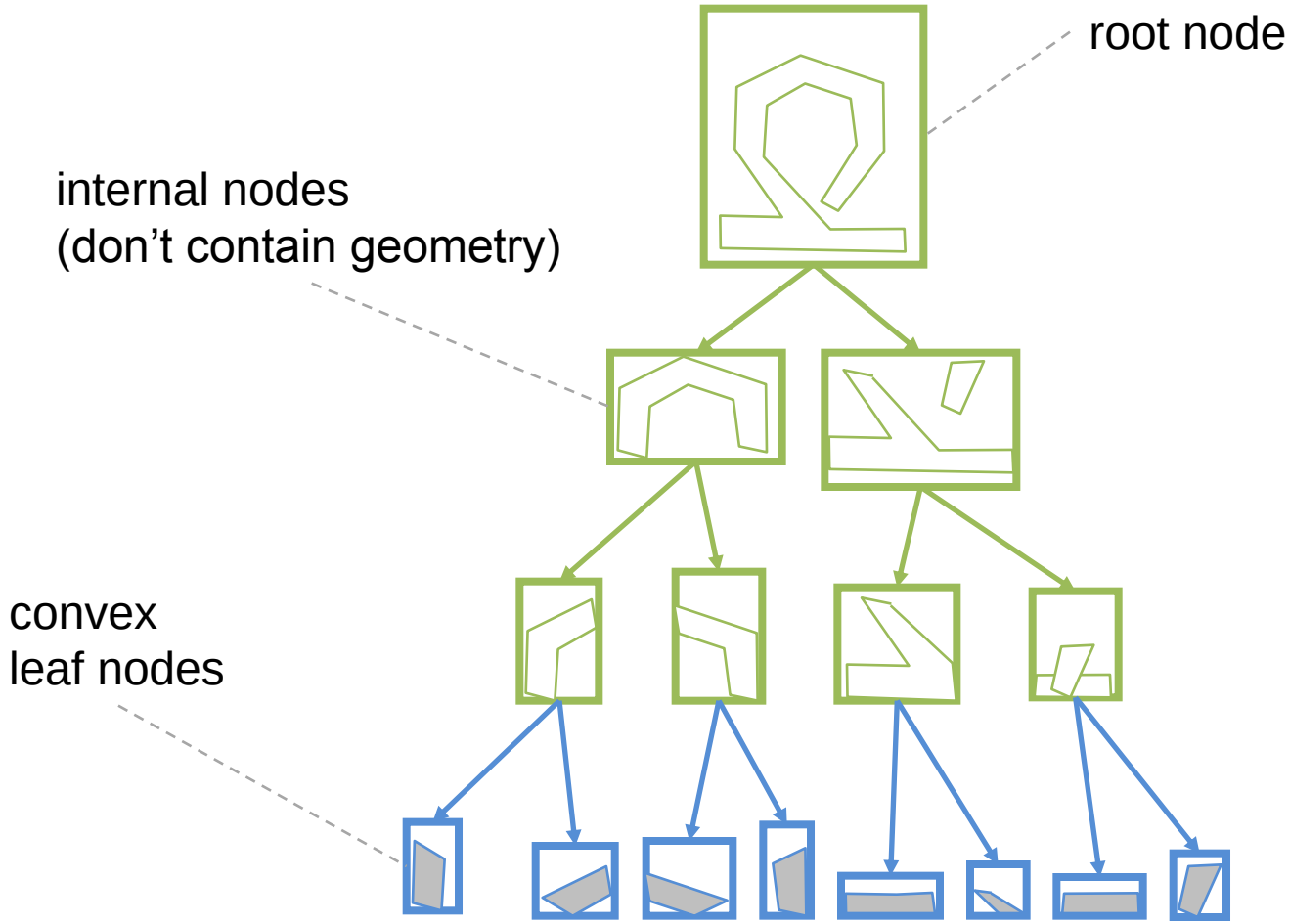
Contact for concave shapes

- GJK and EPA can only deal with convex objects
- Convex decomposition
- Triangle meshes: a single triangle is convex
- Concave mesh is just a collection of triangles
- Deal with each triangle individually

Concave Shapes: AABB trees

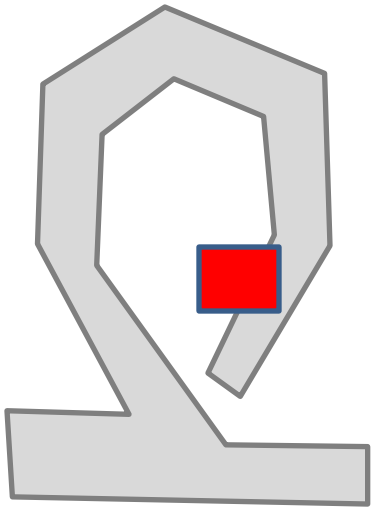


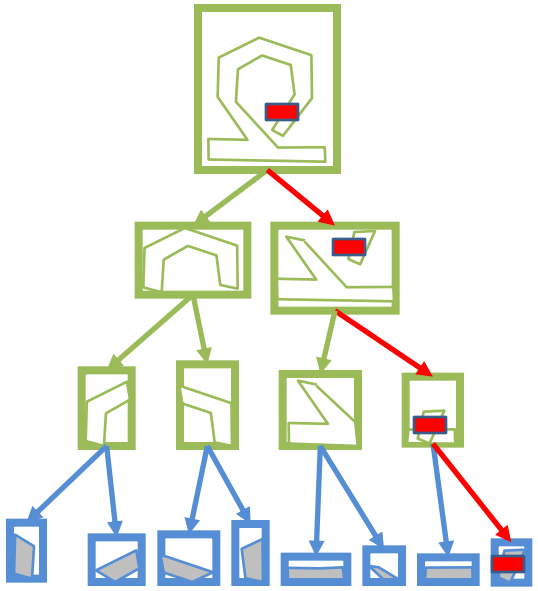
AABB tree structure



AABB tree queries by traversal

- Find overlapping nodes given a AABB
- Find overlapping nodes given a Ray (from,to)





AABB tree traversal

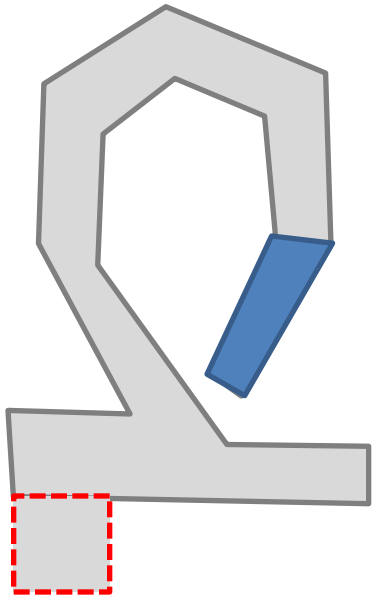
- Recursive
- Stackless with skip indices
- History tracking (see Harada's talk)

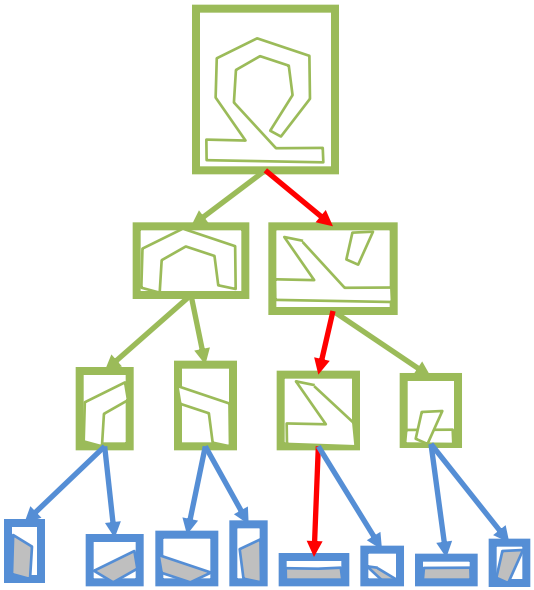
AABB tree

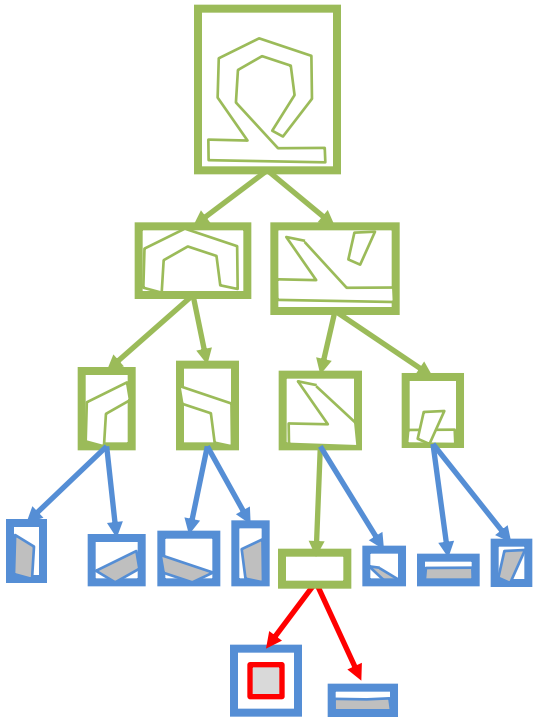
- Static AABB tree
  - Can be optimized and quantized
  - Allows for basic tree refit operator
- Dynamic AABB tree
  - Can deal with change in topology
  - Add and remove nodes
  - Incrementally rebalance tree
  - Is very general purpose

Adding a node

- Find leaf with smallest Manhattan distance





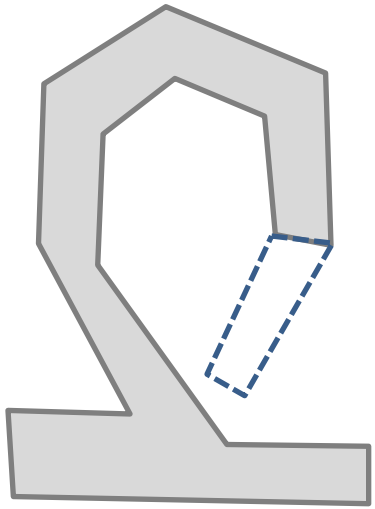


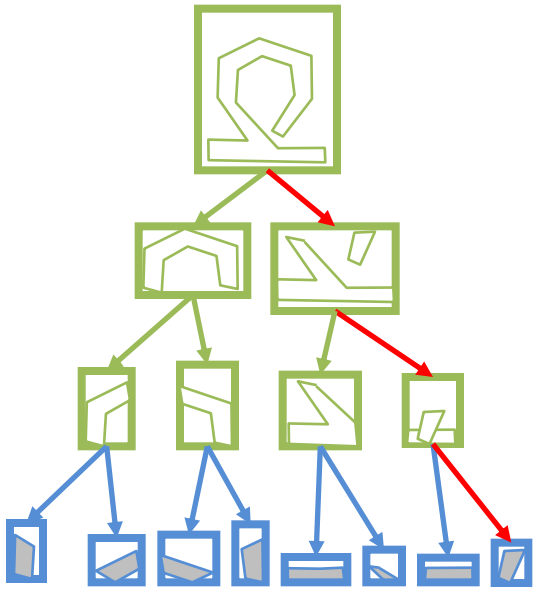
Implementation insert leaf node

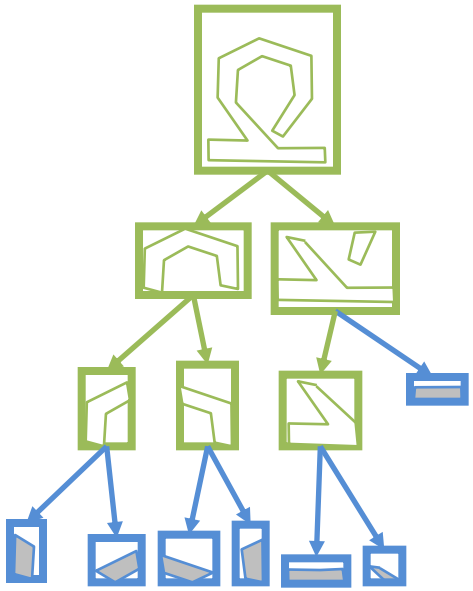
```
void insertleaf(        btDbvt* pdbvt,btDbvtNode* root,btDbvtNode* leaf) {
    if(!pdbvt->m_root){pdbvt->m_root=leaf;leaf->parent=0;}
    if(!root->isleaf()) {
            do {
                    root=root->childs[Select(leaf->volume,root->childs[0]->volume,
                                root->childs[1]->volume)];
            } while(!root->isleaf());
    }
    btDbvtNode* prev=root->parent;
    btDbvtNode*         node=createnode(pdbvt,prev,leaf->volume,root->volume,0);
    if(prev)  {
            prev->childs[indexof(root)]=node;
            node->childs[0] =       root;root->parent=node;
            node->childs[1]  =      leaf;leaf->parent=node;
            do  {
                if(!prev->volume.Contain(node->volume))
                  Merge(prev->childs[0]->volume,prev->childs[1]->volume,prev->volume);
                else
                  break;
                node=prev;
            } while(0!=(prev=node->parent));
    }
    else {
            node->childs[0]         =           root;root->parent=node;
            node->childs[1]         =           leaf;leaf->parent=node;
            pdbvt->m_root           =           node;
    }
}
```

Removing a leaf node

- Find and remove node and relink







Implementation remove leaf

```
1: btDbvtNode*          removeleaf(btDbvt* pdbvt, btDbvtNode* leaf){
2:   if(leaf==pdbvt->m_root){ pdbvt->m_root=0; return(0);}
3:   btDbvtNode*        parent=leaf->parent;
4:   btDbvtNode*        prev=parent->parent;
5:   btDbvtNode*        sibling=parent->childs[1-indexof(leaf)];
6:   if(prev)
7:   {
8:           prev->childs[indexof(parent)]=sibling;
9:           sibling->parent=prev;
10:          deletenode(pdbvt,parent);
11:          while(prev) {
12:                      const btDbvtVolume      pb=prev->volume;
13:                      Merge(prev->childs[0]->volume,prev->childs[1]->volume,prev->volume);
14:                      if(NotEqual(pb,prev->volume)){
15:                                  prev=prev->parent;
16:                      } else break;
17:          }
18:          return(prev?prev:pdbvt->m_root);
19:  }
20:  else
21:  {
22:          pdbvt->m_root=sibling;
23:          sibling->parent=0;
24:          deletenode(pdbvt,parent);
25:          return(pdbvt->m_root);
26:   }
27: }
```

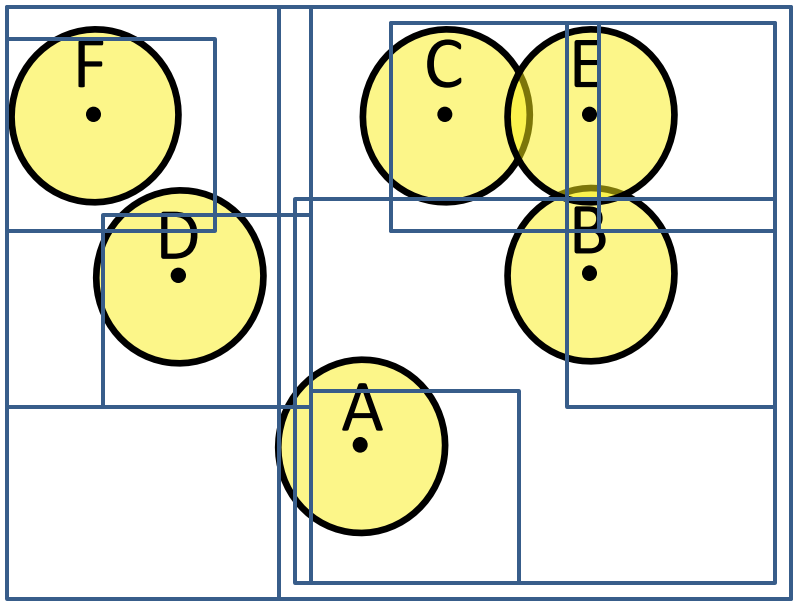
Update/move a leaf node

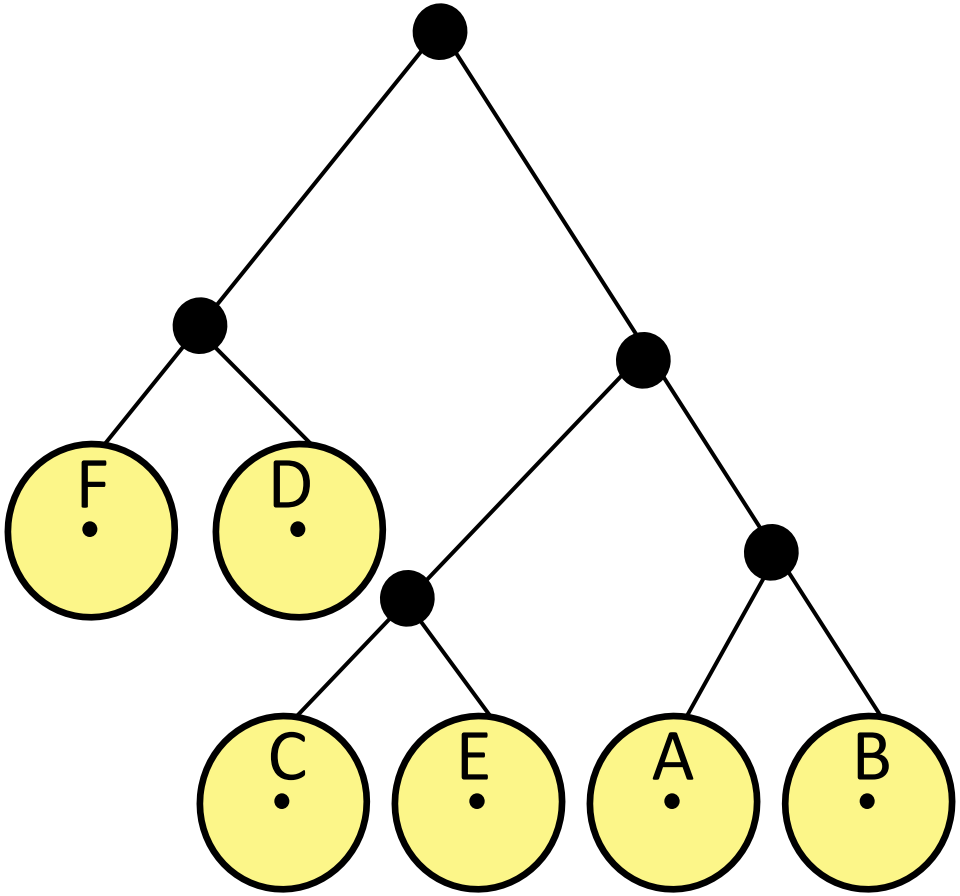
- If new AABB is contained by old do nothing
- Otherwise remove and re-insert leaf
  - Re-insert at closest ancestor that was not resized during remove (see line 18 previous page)
- Expand AABB with margin
  - Avoid updates due to jitter or small random motion
- Expand AABB with velocity
  - Handle the case of linear motion over n frames

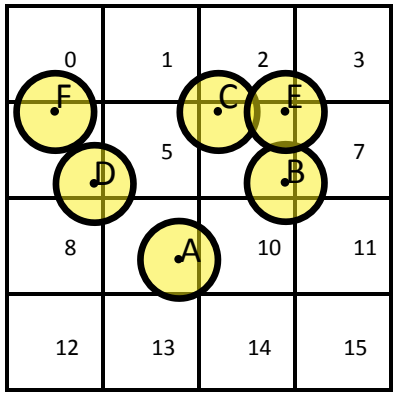
Incremental tree optimization

- Rebalance tree removing and inserting a few leaf nodes at a time

Dynamic AABB tree broadphase

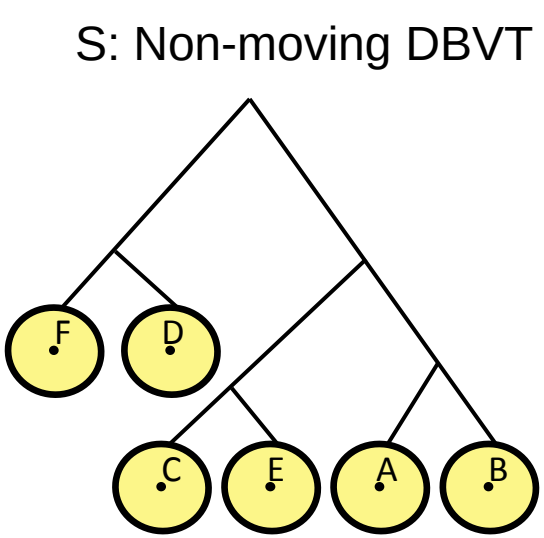


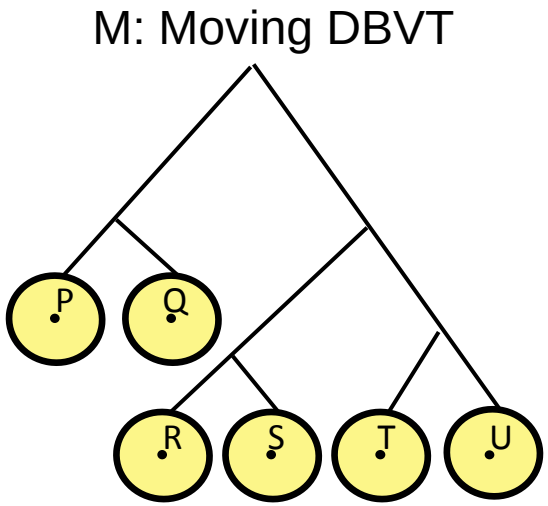




Dynamic BVH tree broadphase

- Keep two dynamic trees, one for moving objects, other for objects (sleeping/static)
- Find neighbor pairs:
  - Overlap M versus M and Overlap M versus S





DEMO!!!

Summary

- Use a persistent manifold for multiple points
  - keep contacts in local space and update distance
- Adjust normals to avoid internal edge collisions
  - only if normal outside edge voronoi region
- Dynamic AABB trees are fast and versatile acceleration structure for
  - broadphase pair search and ray test
  - midphase for triangle meshes, cloth, deformables
  - occlusion and view frustum culling

Bullet

- An open source 3D physics engine
- http://bulletphysics.org
- Written in C++