Collision Detection

contact generation and GPU acceleration

Erwin Coumans, AMD

http://bulletphysics.org

In a nutshell

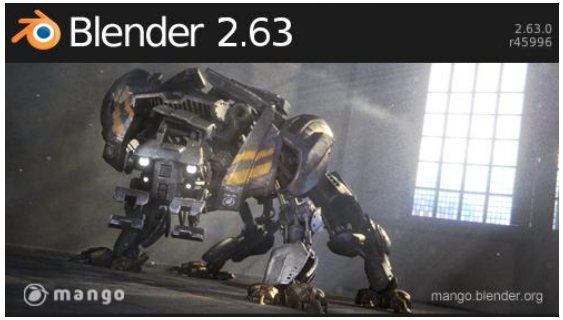
- Intro to the Bullet physics engine
- Broadphase acceleration
  - Parallel sweep and prune broadphase
  - Dynamic AABB tree general purpose acceleration structure
- Midphase acceleration
- Narrowphase collision detection
  - Separating Axis Test
  - Contact Generation
  - GJK closest points, EPA , CCD Conservative Advancement

Bullet physics engine

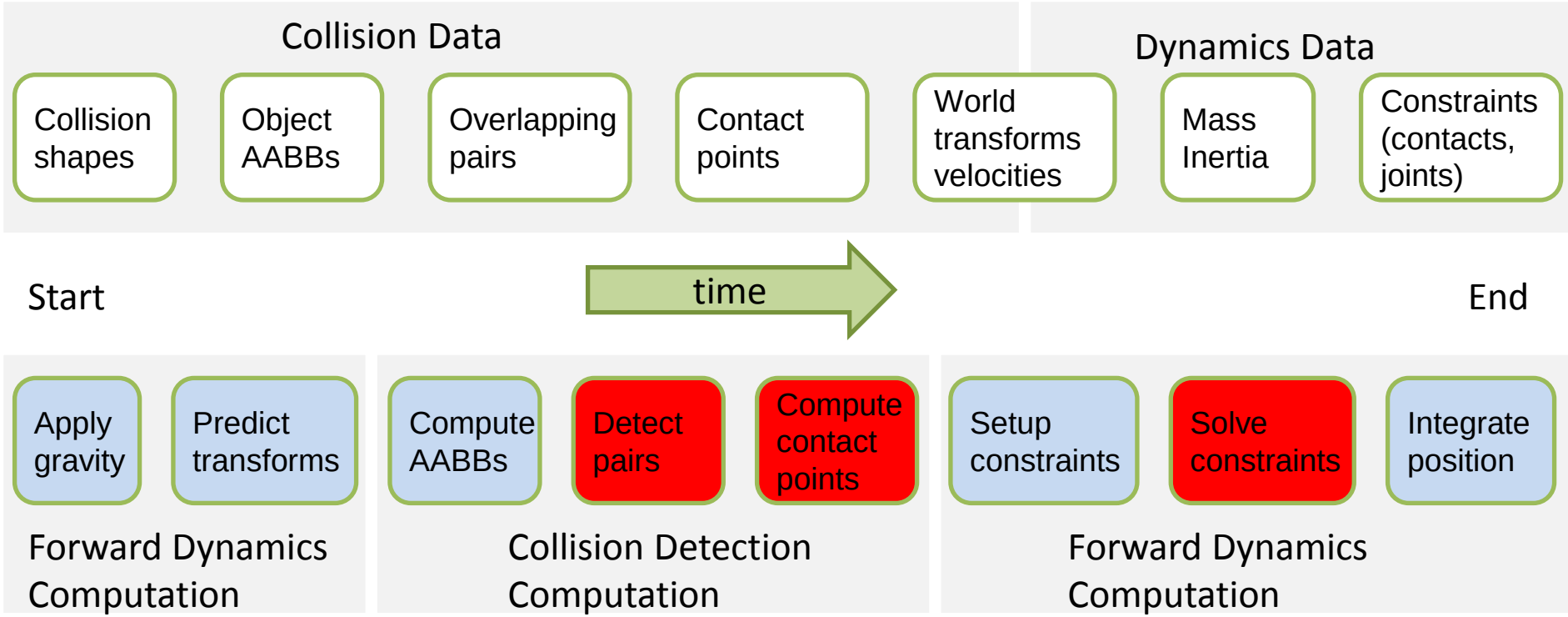
- Simulate Rigid Body, Cloth, Deformables
- Discrete and Continuous Collision Detection
- Open source using the Zlib license
- Free for commercial use
- Written in C++
- OpenCL and Direct Compute for GPU

Bullet Authoring tools

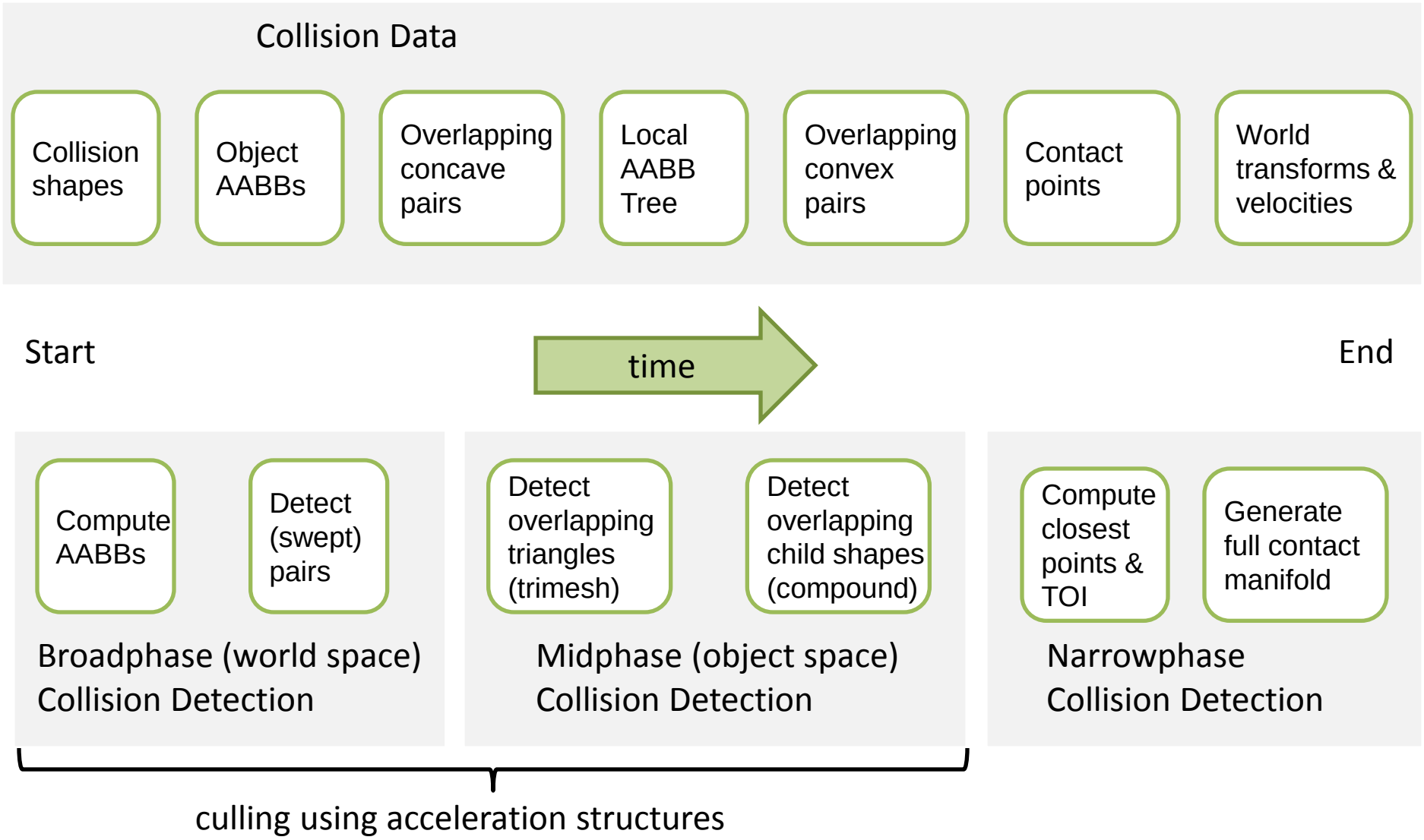
- Maya 2013
- Dynamica (open source)
- Cinema 4D
- Lightwave
- Blender
- Houdini Plugin



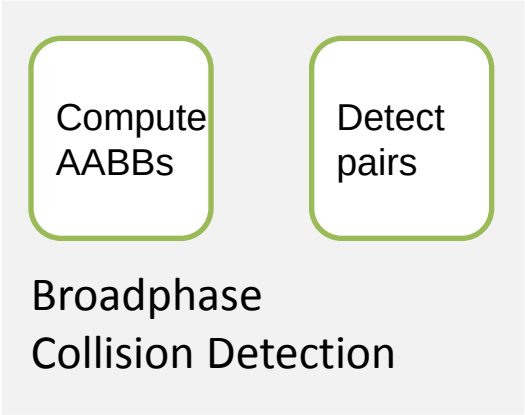
Physics Simulation Pipeline



Collision Detection Pipeline

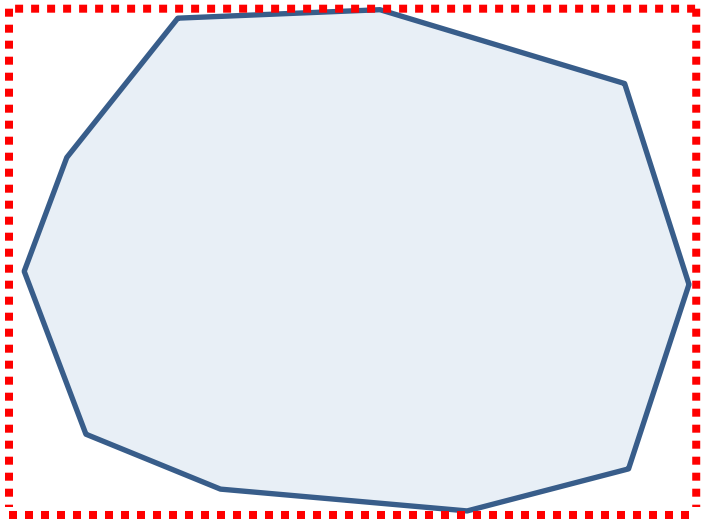


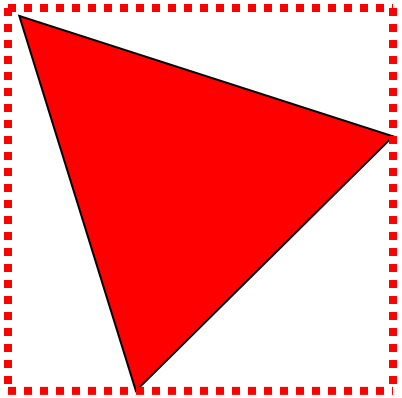
Broadphase N-body problem



- Input: world space BVs and unique IDs
- Output: array of potential overlapping pairs
  - also ray intersection , swept volumes and CCD

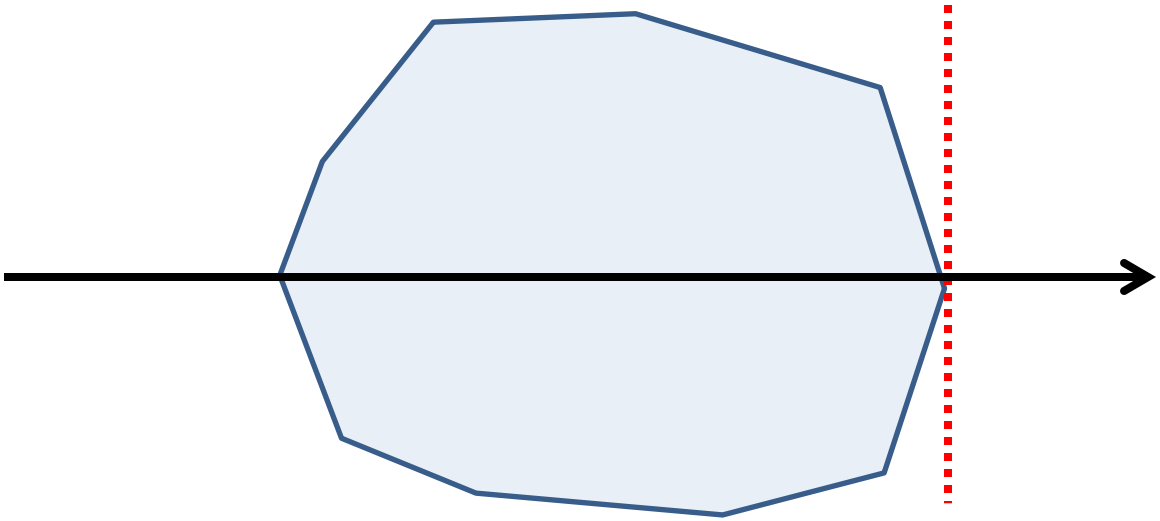
Axis Aligned Bounding Boxes





Support Mapping

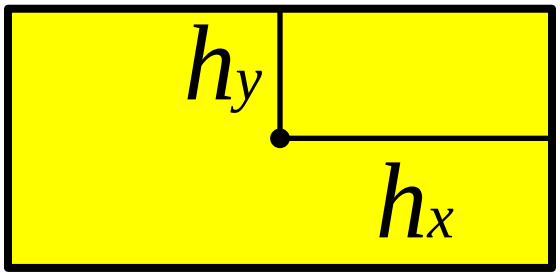
$$S_c(v) = \max\{v \cdot x : x \in C\}$$



Support mapping for primitives

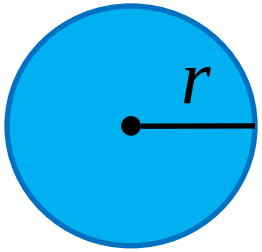
- Box with half extents h

$$S_{box}(v) = (sign(v_x)h_x, sign(v_y)h_y, sign(v_z)h_z)$$



- Sphere with radius r

$$S_{sphere}(v) = \frac{r}{|v|}v$$



- Affine transform

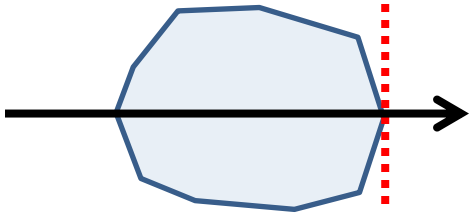
$$S_{Bx+c}(v) = B(S(B^t v) + c$$

See the book "Collision Detection in Interactive 3D Environments", 2004, Gino Van Den Bergen

Support mapping: convex polyhedra

- Brute force search
  - O(n), cache friendly, SIMD
  - Best for < 200 vertices

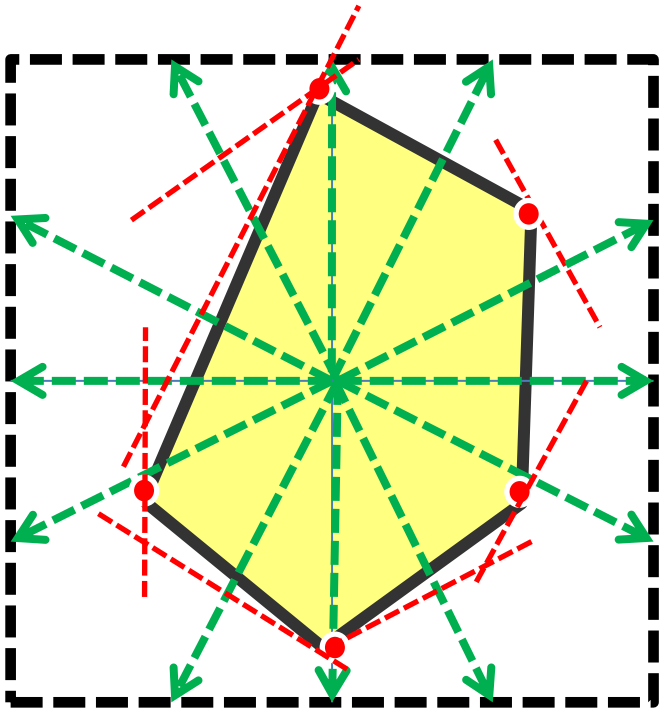
$$S_c(v) = \max\{v \cdot x : x \in C\}$$



- Dobkin-Kirkpatrick Hierarchy
  - O(log n), see 7.10 in "Computational Geometry in C" by Joseph O'Rourke

Support mapping approximation

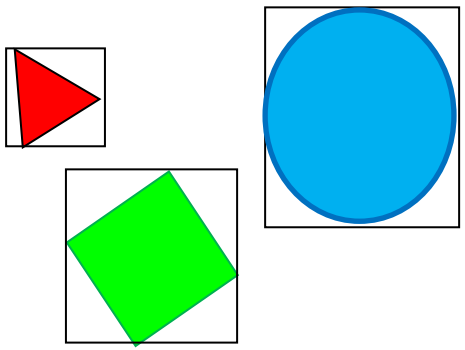
- Cube map
  - O(1), approximate



- Software or GPU cube mapping hardware

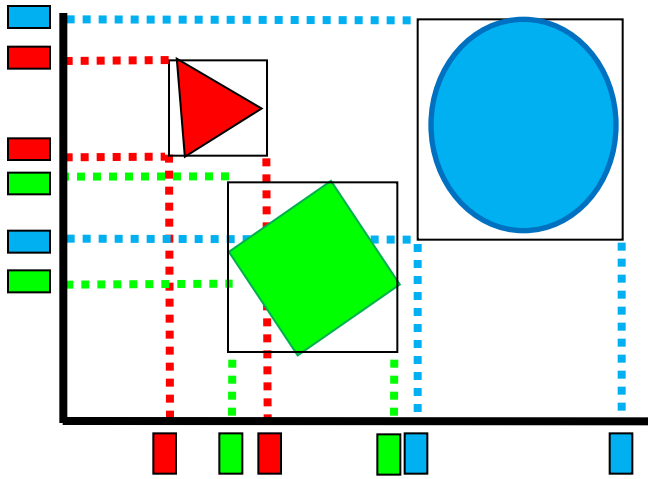
Brute force overlap test

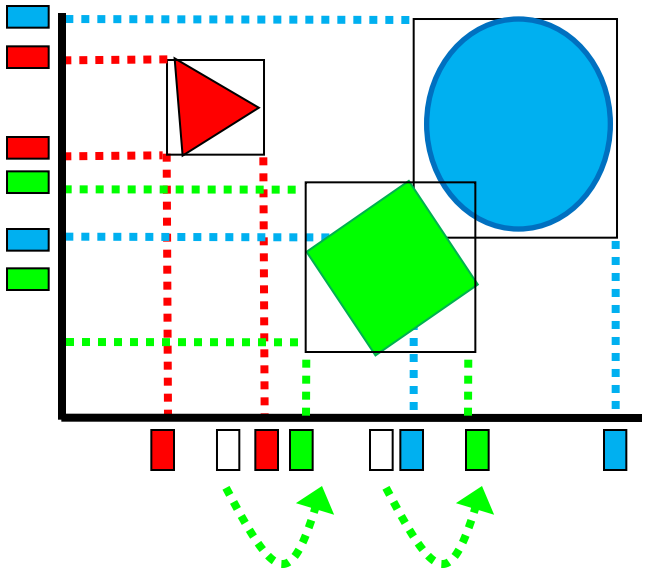
```
bool TestAabbAgainstAabb (__global const btAabbCL* aabb1, __global const btAabbCL* aabb2) {
                bool overlap = true;
                overlap = (aabb1->m_min.x > aabb2->m_max.x || aabb1->m_max.x < aabb2->m_min.x) ? false : overlap;
                overlap = (aabb1->m_min.z > aabb2->m_max.z || aabb1->m_max.z < aabb2->m_min.z) ? false : overlap;
                overlap = (aabb1->m_min.y > aabb2->m_max.y || aabb1->m_max.y < aabb2->m_min.y) ? false : overlap;
                return overlap;
}
__kernel void   computePairsBruteForceKernel( __global const btAabbCL* aabbs, volatile __global int2* pairsOut,
                                                volatile  __global int* pairCount, int numObjects, int axis, int maxPairs)
{
   int i = get_global_id(0);
   if (i>=numObjects)
     return;
   for (int j=i+1;j<numObjects;j++)
   {
      if (TestAabbAgainstAabb (&aabbs[i],&aabbs[j]))
      {
         int2 myPair;
         myPair.x = aabbs[i].m_minIndices[3];
         myPair.y = aabbs[j].m_minIndices[3];
         int curPair = atomic_inc (pairCount);
         if (curPair<maxPairs)
         {
            pairsOut[curPair] = myPair; //flush to main memory
} } } }
```



Incremental sweep and prune

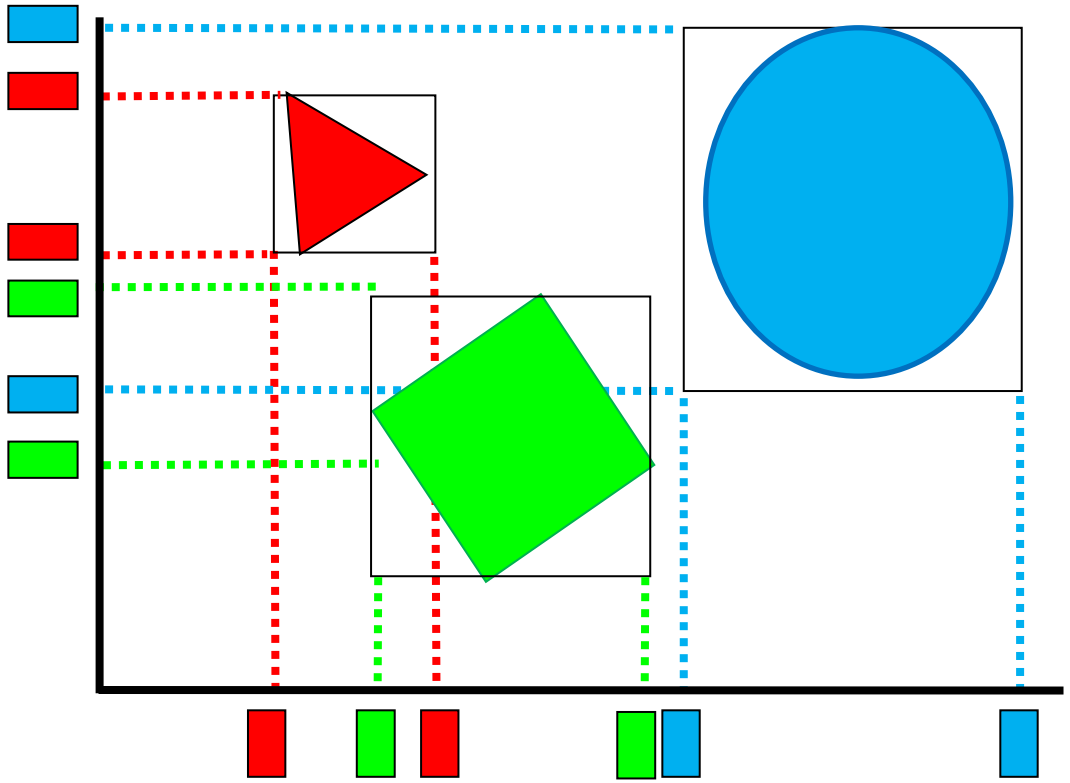
- Update 3 sorted axis and overlapping pairs





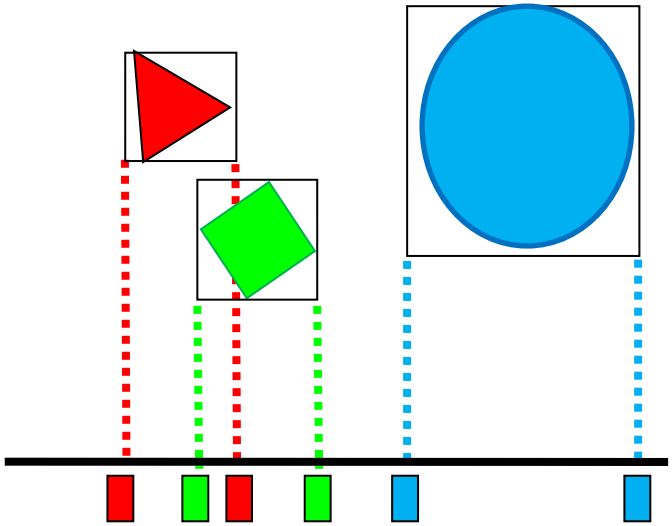
- Performs best if most objects hardly move
- Difficult to parallelize

3-axis SAP ray query/swept AABB

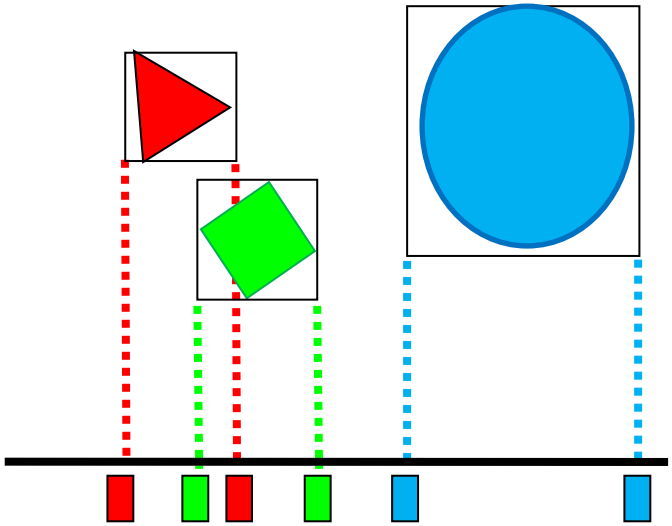


1-axis Sweep and Prune (SAP)

- Find best projection axis
- Sort aabbs along this axis
- For each object, find and add overlapping pairs



GPU parallel SAP



- Find best projection axis
  - Parallel prefix sum
  - Principal Component Analysis
- Sort aabbs along this axis
  - Parallel radix sort
- For each object, find and add overlapping pairs
  - One work item (thread) per object

Parallel SAP implementation

```
__kernel void   computePairsSAPKernel( __global const btAabbCL* aabbs, volatile __global int2* pairsOut,
                                                    volatile  __global int* pairCount, int numObjects, int axis, int maxPairs)
{
   int i = get_global_id(0);
   if (i>=numObjects)
     return;
   for (int j=i+1;j<numObjects;j++)
   {
      if(aabbs[i].m_maxElems[axis] < (aabbs[j].m_minElems[axis]))
         break;
      if (TestAabbAgainstAabb2GlobalGlobal(&aabbs[i],&aabbs[j]))
      {
         int2 myPair;
         myPair.x = aabbs[i].m_minIndices[3];
         myPair.y = aabbs[j].m_minIndices[3];
         int curPair = atomic_inc (pairCount);
         if (curPair<maxPairs)
         {
            pairsOut[curPair] = myPair; //flush to main memory
         }
      }
   }
}
```

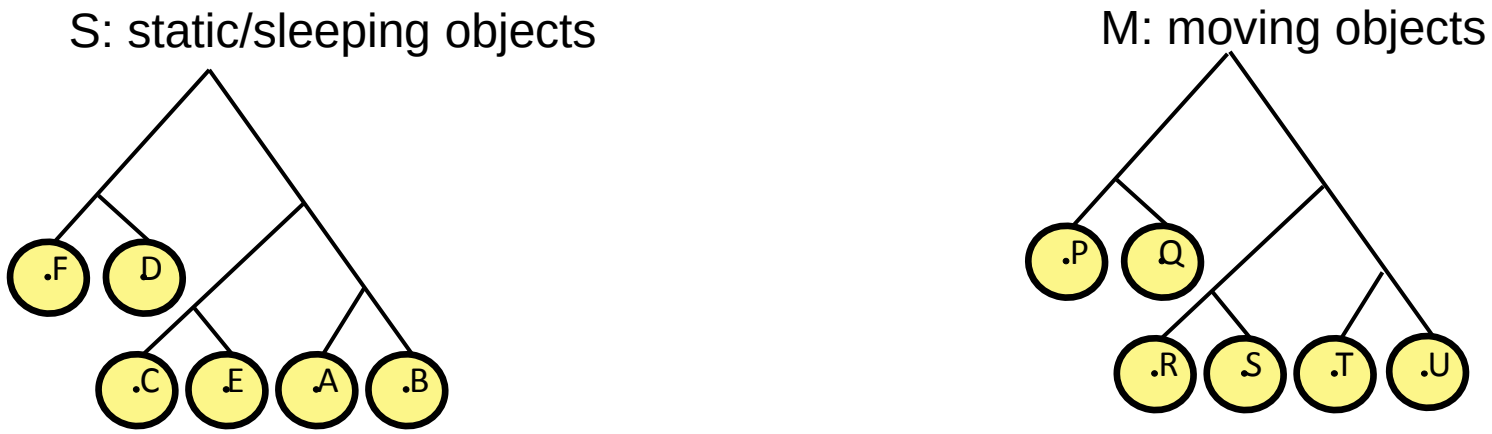
See https://github.com/erwincoumans/experiments/blob/master/opencl/broadphase_benchmark/sap.cl

Further GPU SAP optimizations

- Shared sliding window of AABBs per workgroup
  - store 128 AABBs in local memory
- Buffer the output pairs in private memory
  - reduce the use of global atomics (atomic_add)
- Load balancing
  - split work of large objects into multiple work items

See http://graphics.ewha.ac.kr/gSaP

Dynamic AABB tree broadphase



- Find overlapping pairs:
  - overlap M versus M and Overlap M versus S

Incremental tree update

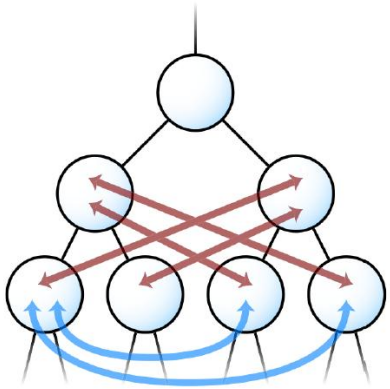
- If new AABB is contained by old do nothing
- Otherwise remove and re-insert leaf
  - Re-insert at closest ancestor that was not resized during remove
- Expand AABB with margin
  - Avoid updates due to jitter or small random motion
- Expand AABB with velocity
  - Handle the case of linear motion over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

Tree balancing

- Incrementally optimize tree

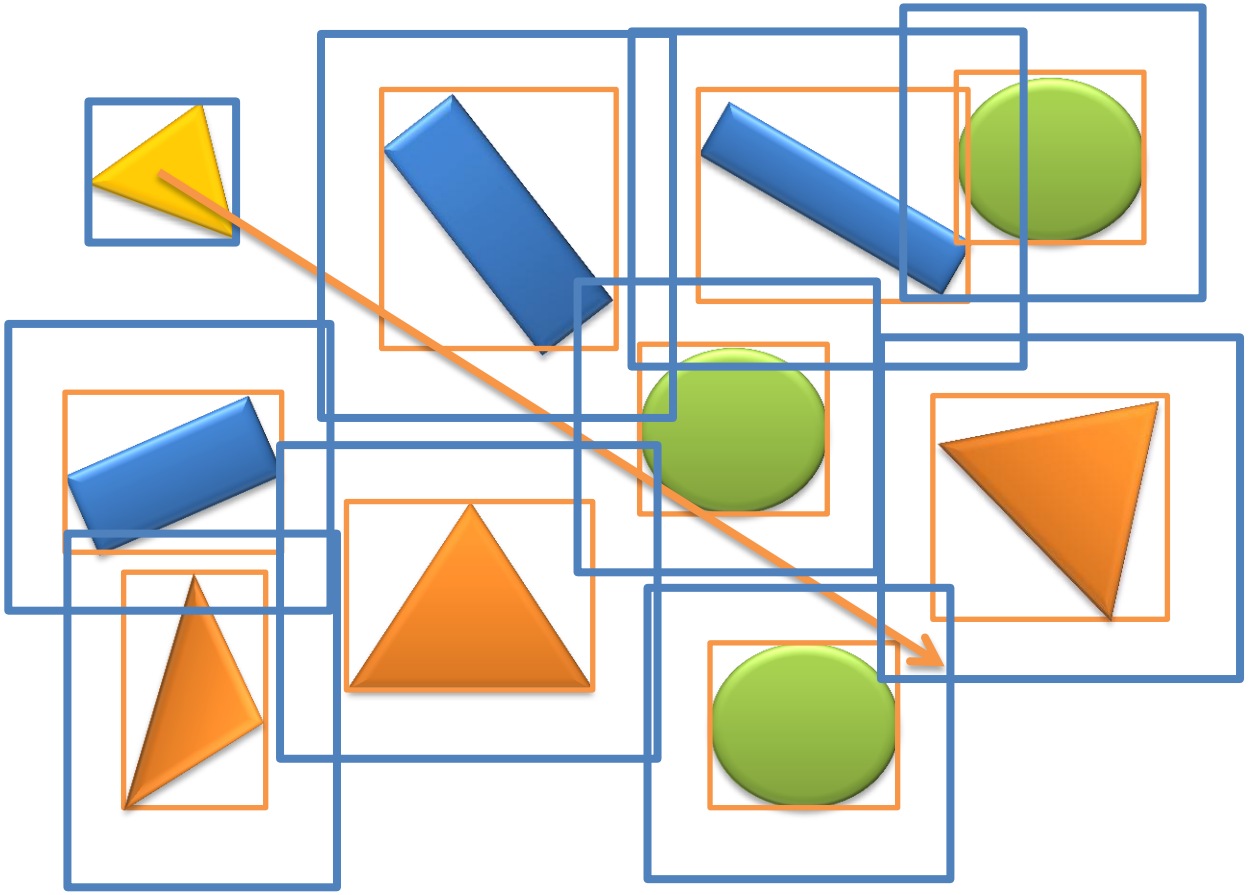
  removal/re-inserting small percentage of nodes each step

- Could use tree rotations instead



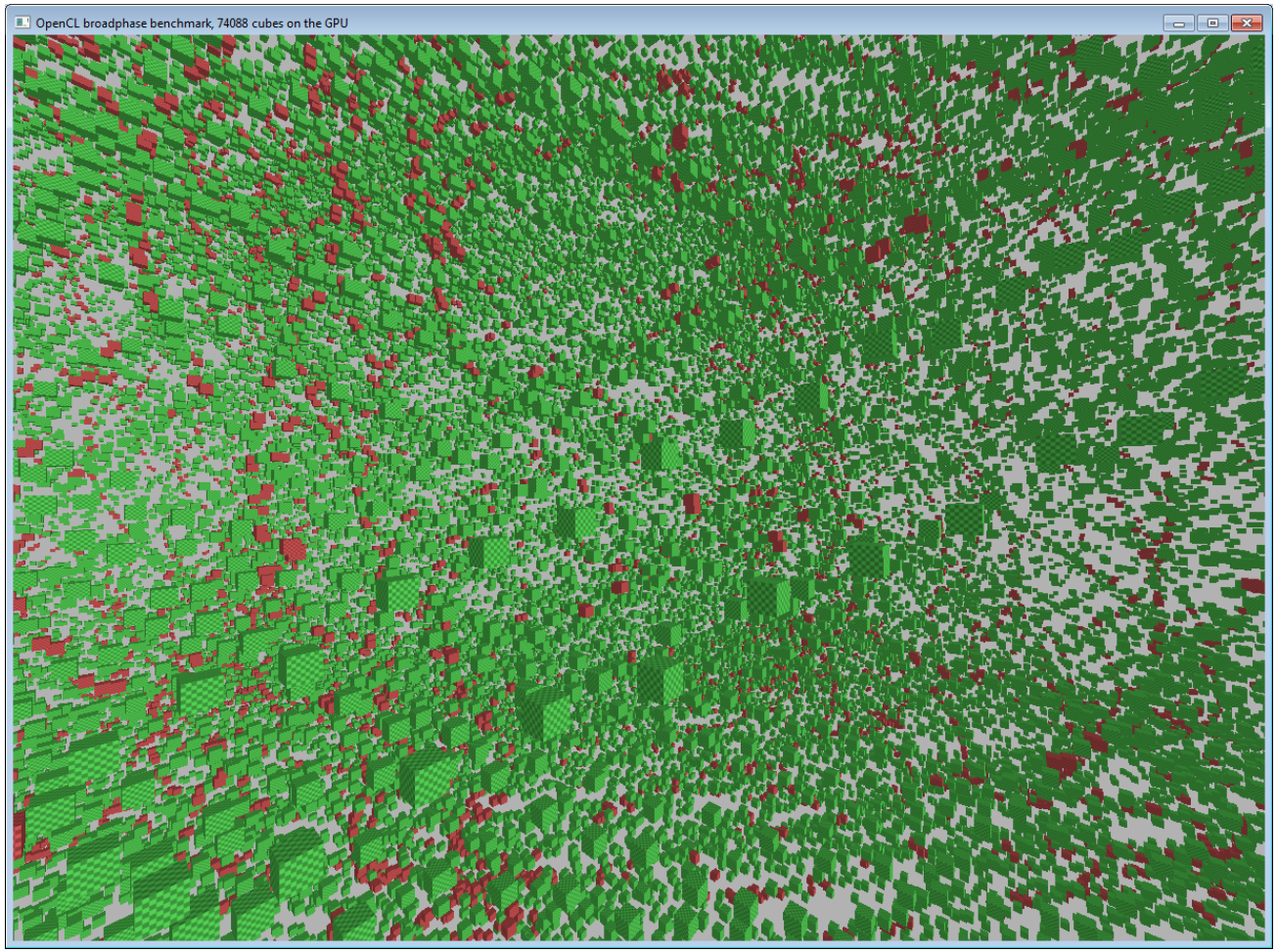
- See "Fast, Effective BVH Updates for Animated Scenes", Kopta, I3D 2012

AABB tree swept query



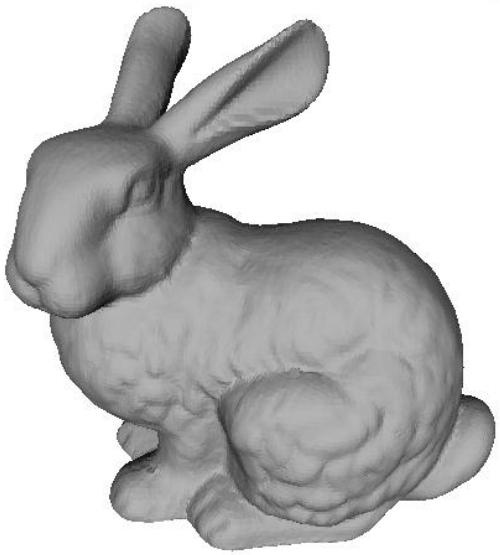
- See Bullet's btDbvtBroadphase::aabbTest

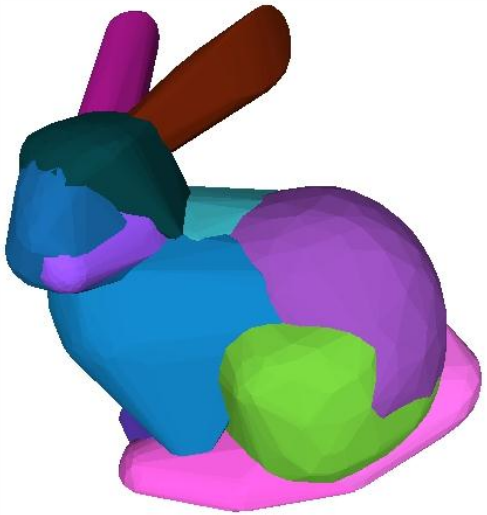
Broadphase benchmark (GPU)

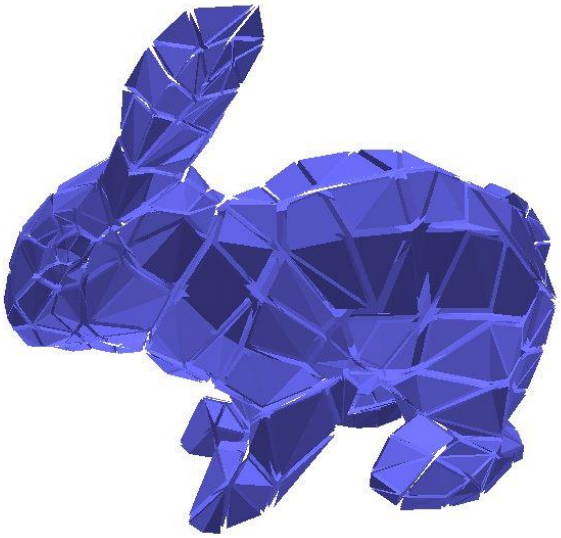


Midphase

- An object-space acceleration structure to cull parts of a complex/concave model

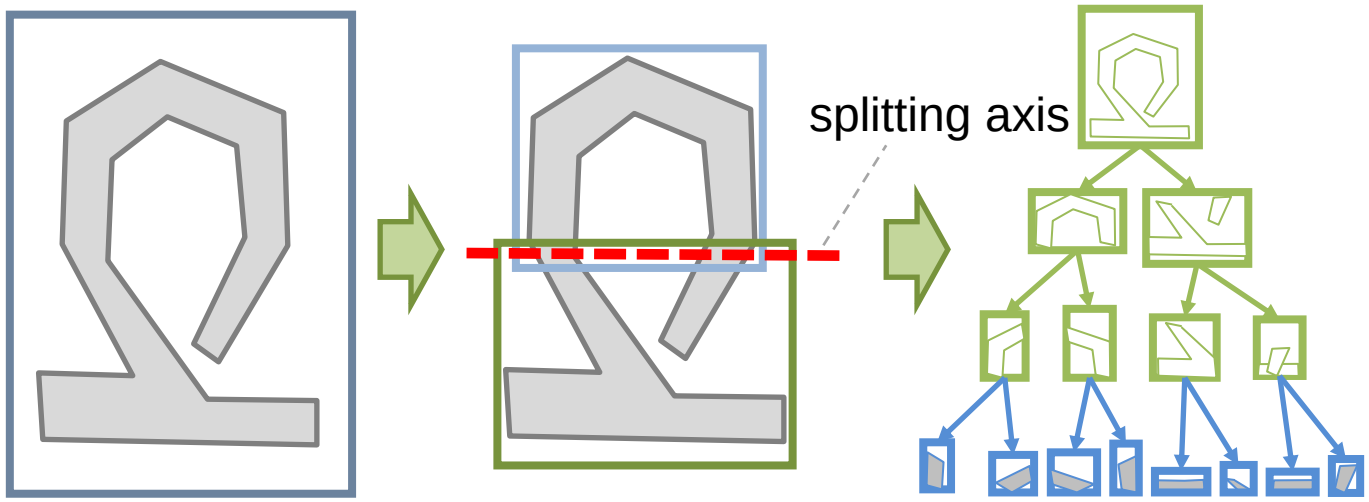


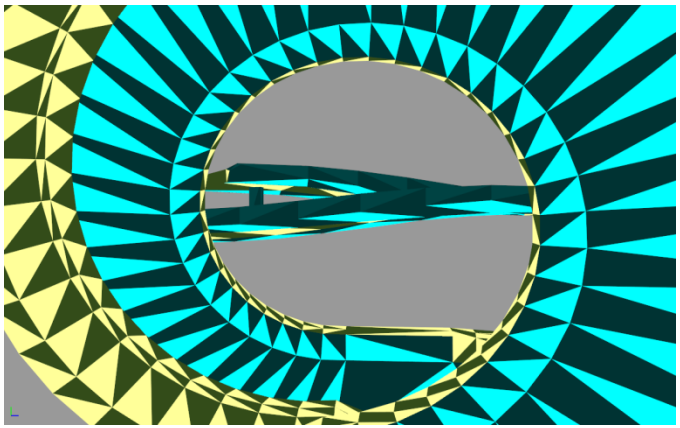




- See Hierarchical Approximate Convex Decomposition, K. Mamou, ICIP 2009 http://sourceforge.net/projects/hacd

Midphase: concave triangle meshes



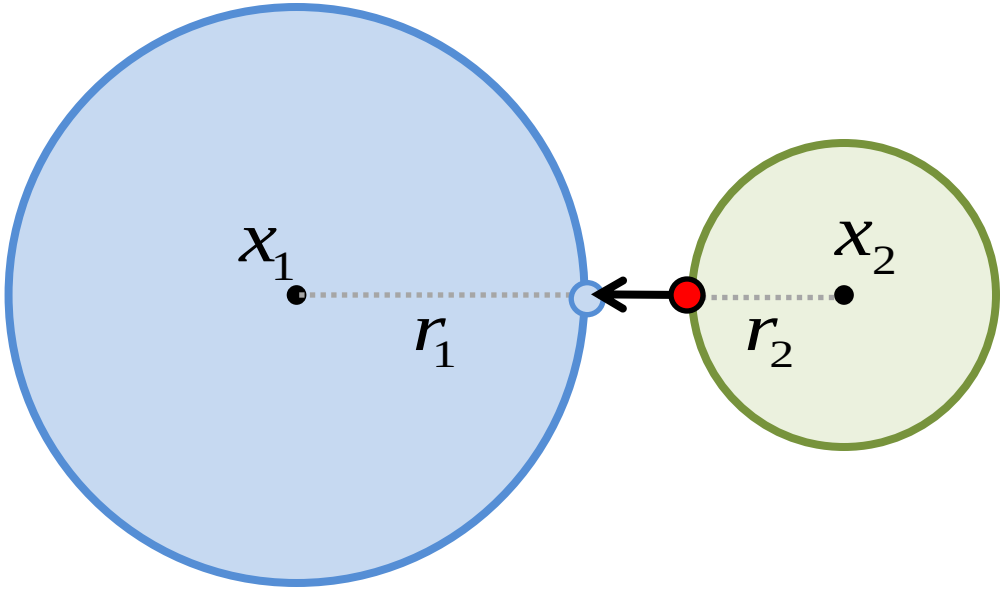


Narrowphase

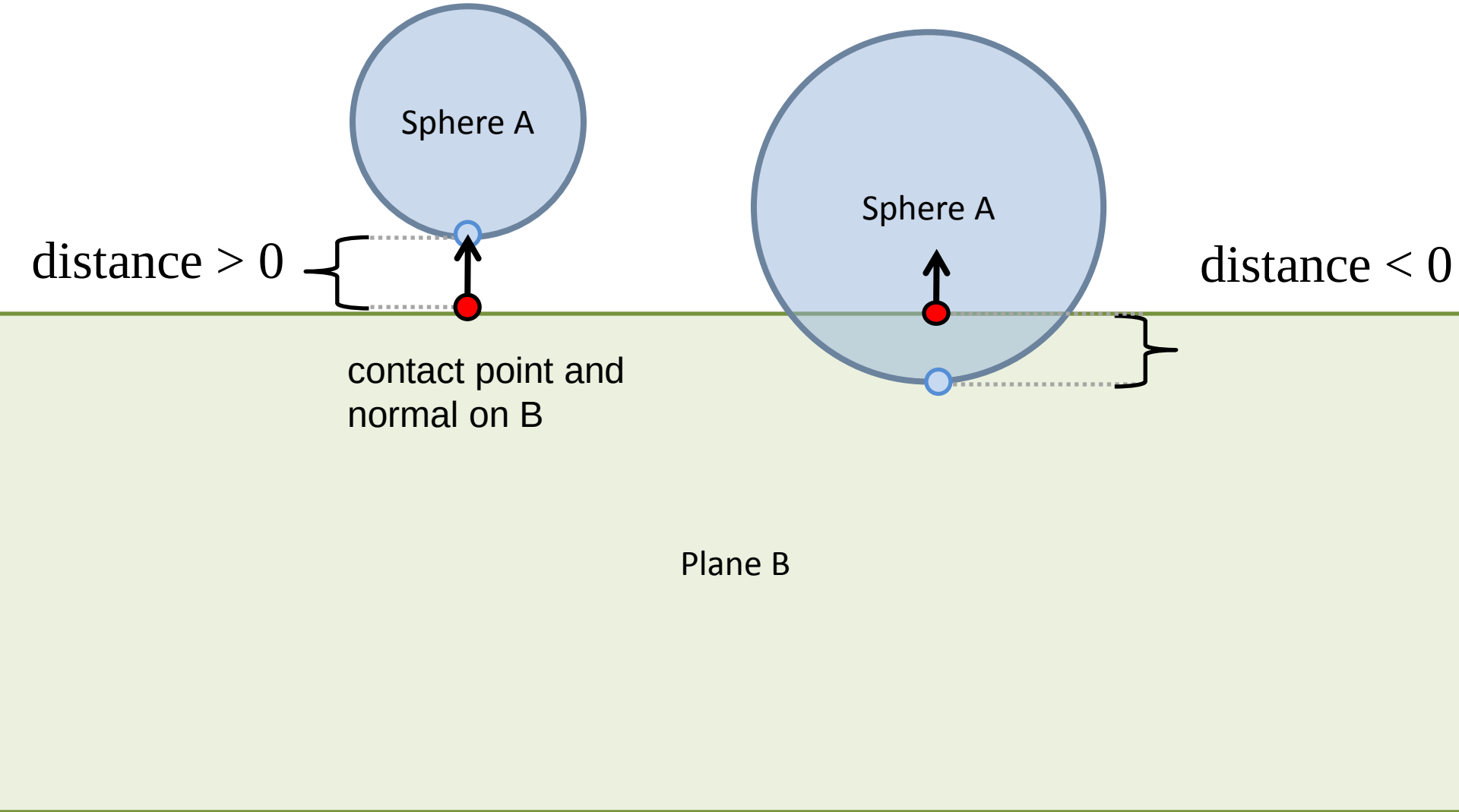
- closest point, normal, distance/penetration
- contact (manifold) generation
- time of impact computation

Closest points

$$C(x_1, x_2) = \| x_1 - x_2 \| - (r_1 + r_2)$$

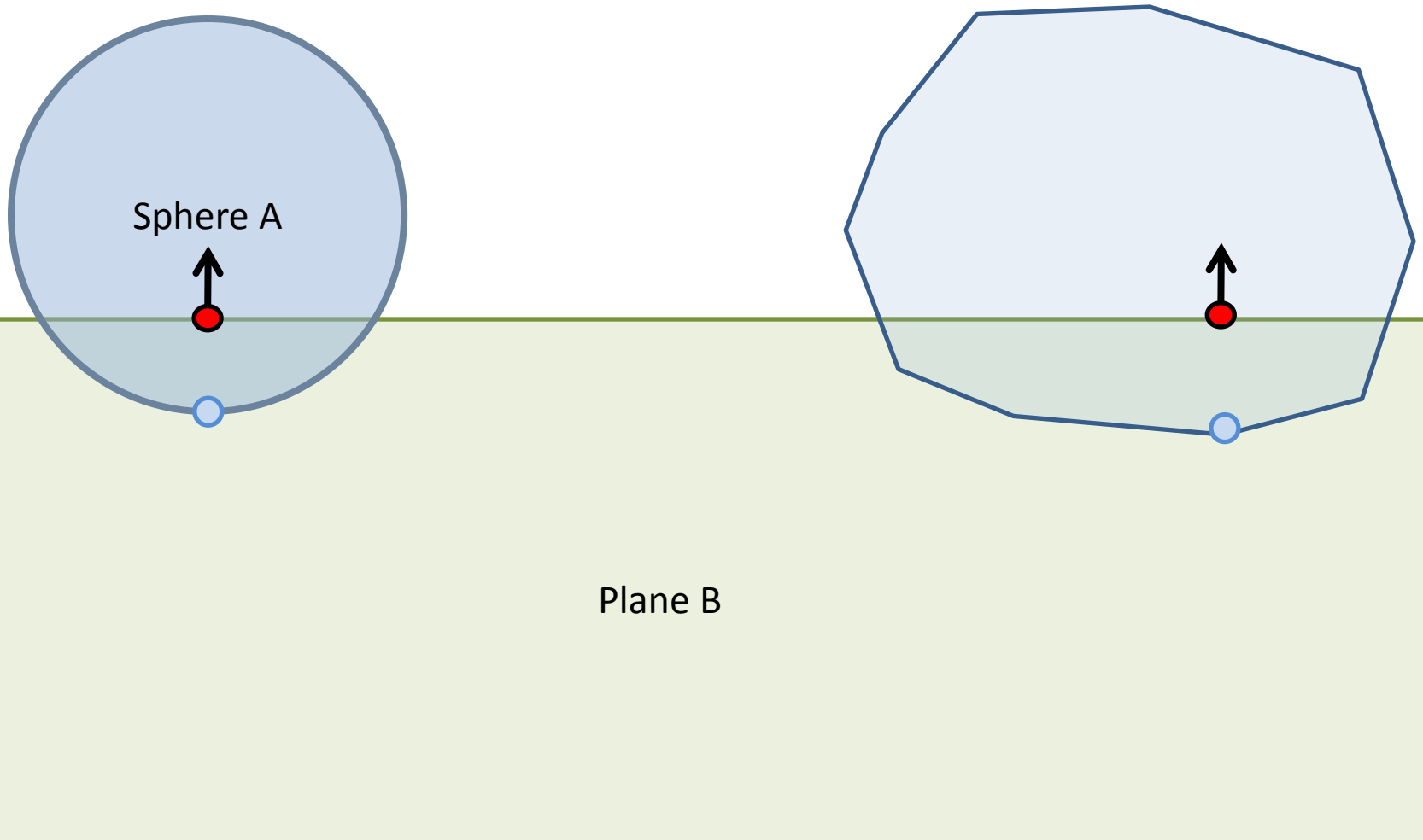


Closest points and penetration



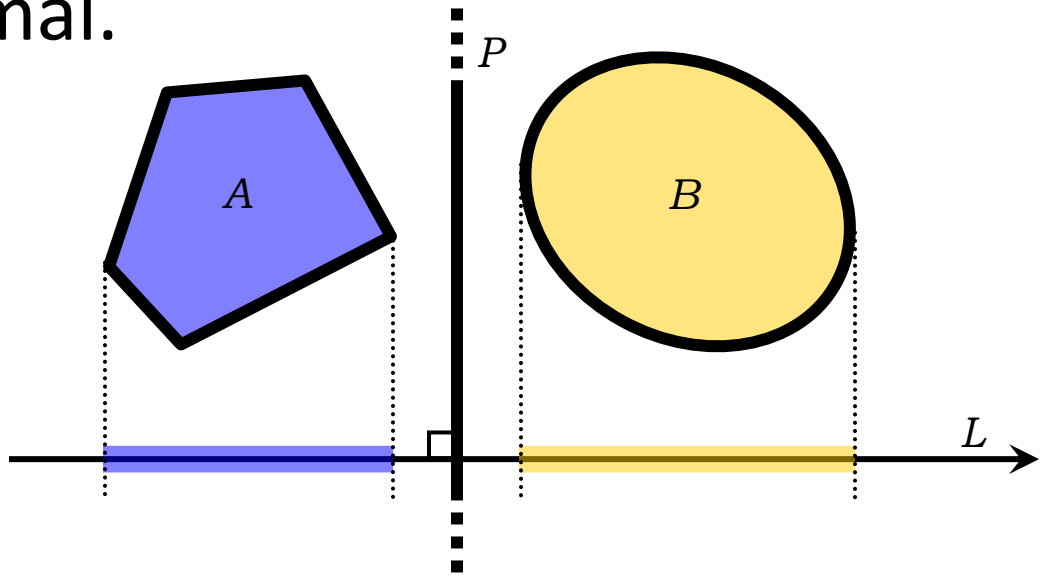
Support mapping

$$S_c(v) = \max\{v \cdot x : x \in C\}$$



Separating Axis

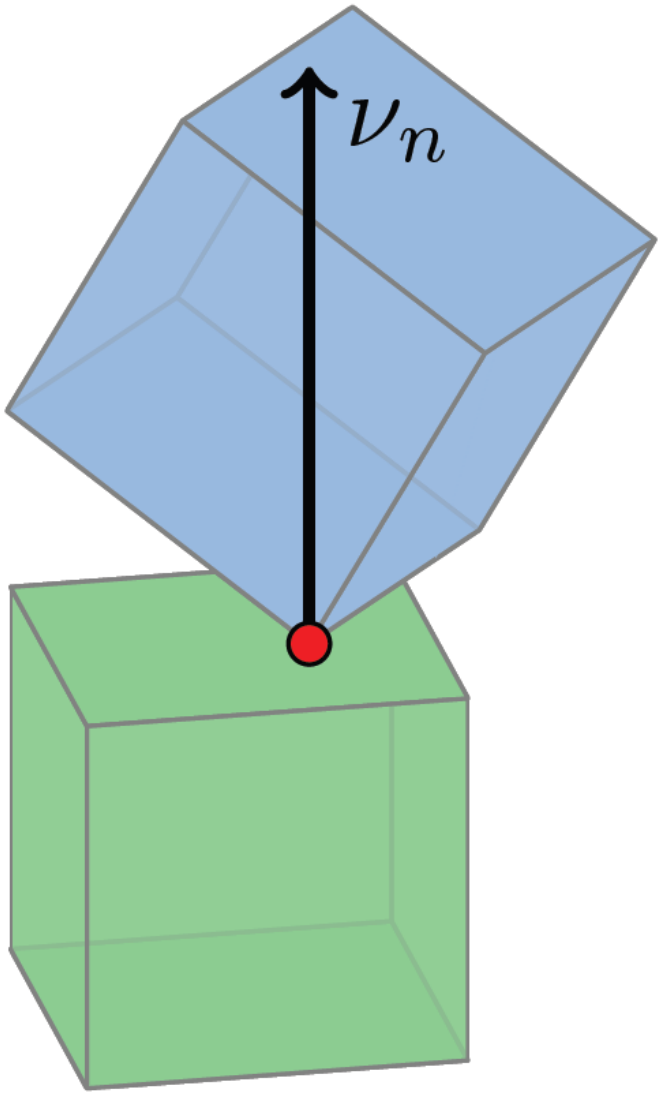
- Separation w.r.t a plane P $\Leftrightarrow$ separation of the orthogonal projections onto any line L parallel to plane normal.



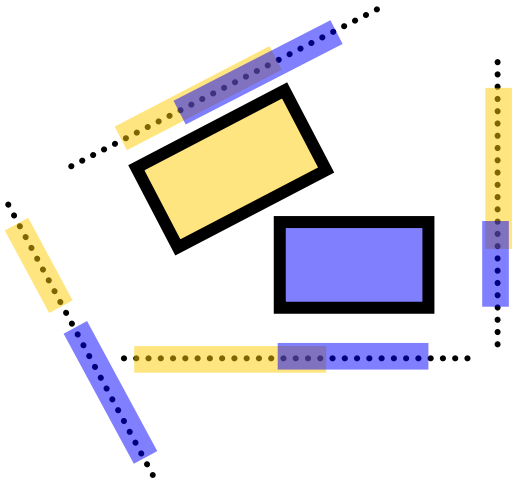
- We can use the support mapping for this projection

Separating Axis for Convex Polyhedra

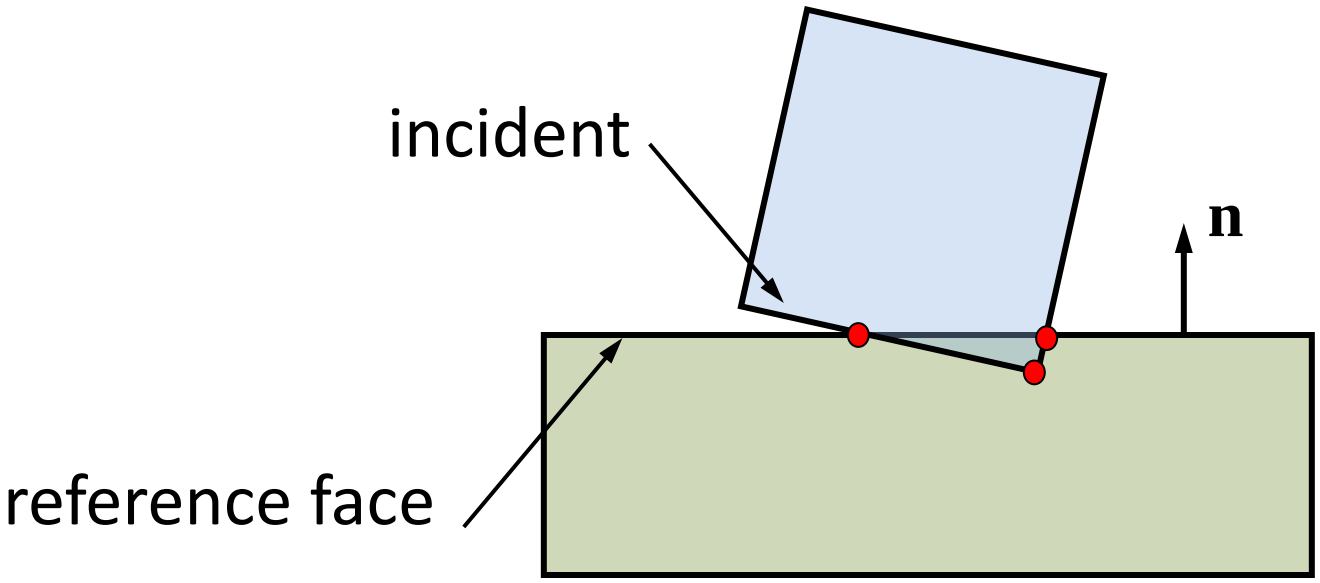
- Face normal A
- Face normal B
- Edge-Edge normal



- Four axes for two 2D OBBs:

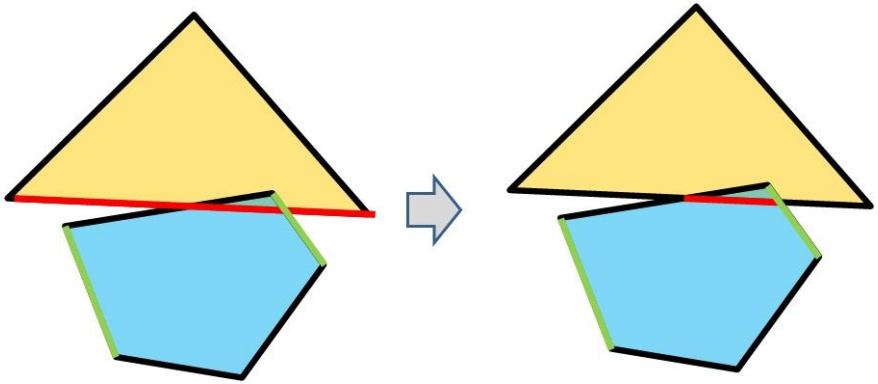


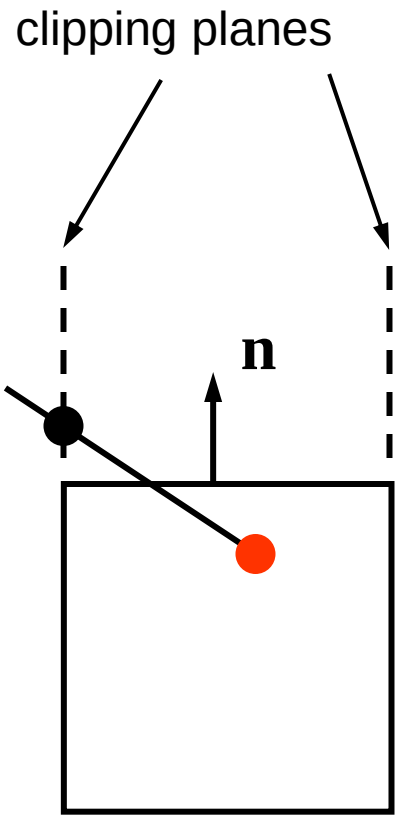
Multiple Contact Points



Sutherland Hodgman clipping

- Clip incident face against reference face side planes
- Consider clip points with positive penetration

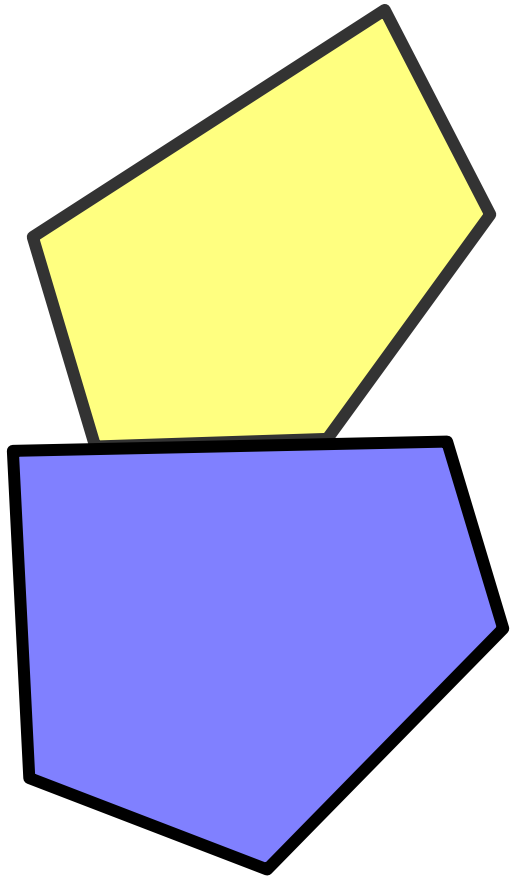




GPU parallel SAT implementation

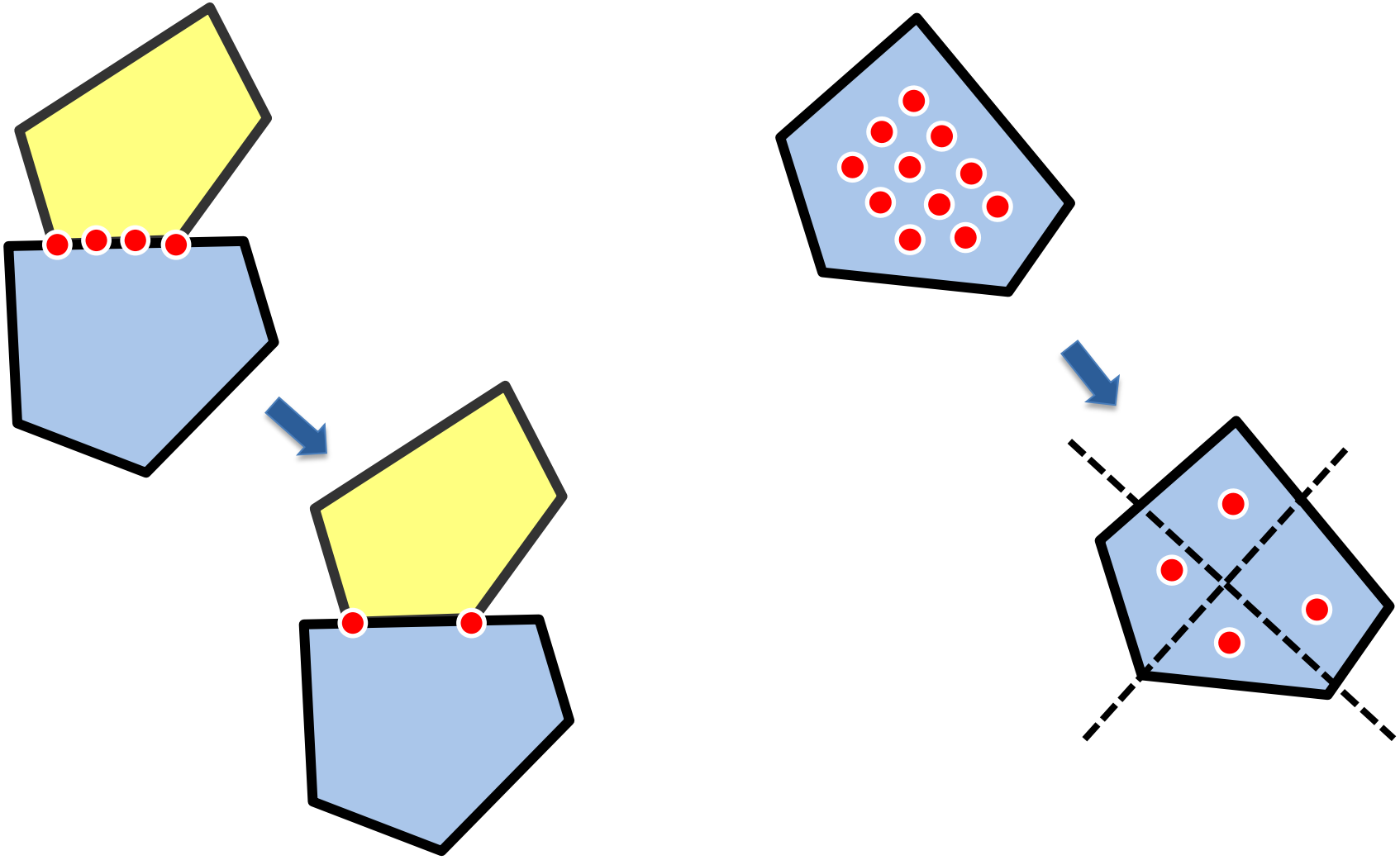
- Test all separating axis in parallel

  >90% of time spend here
- Clip features in parallel
- Parallel contact reduction

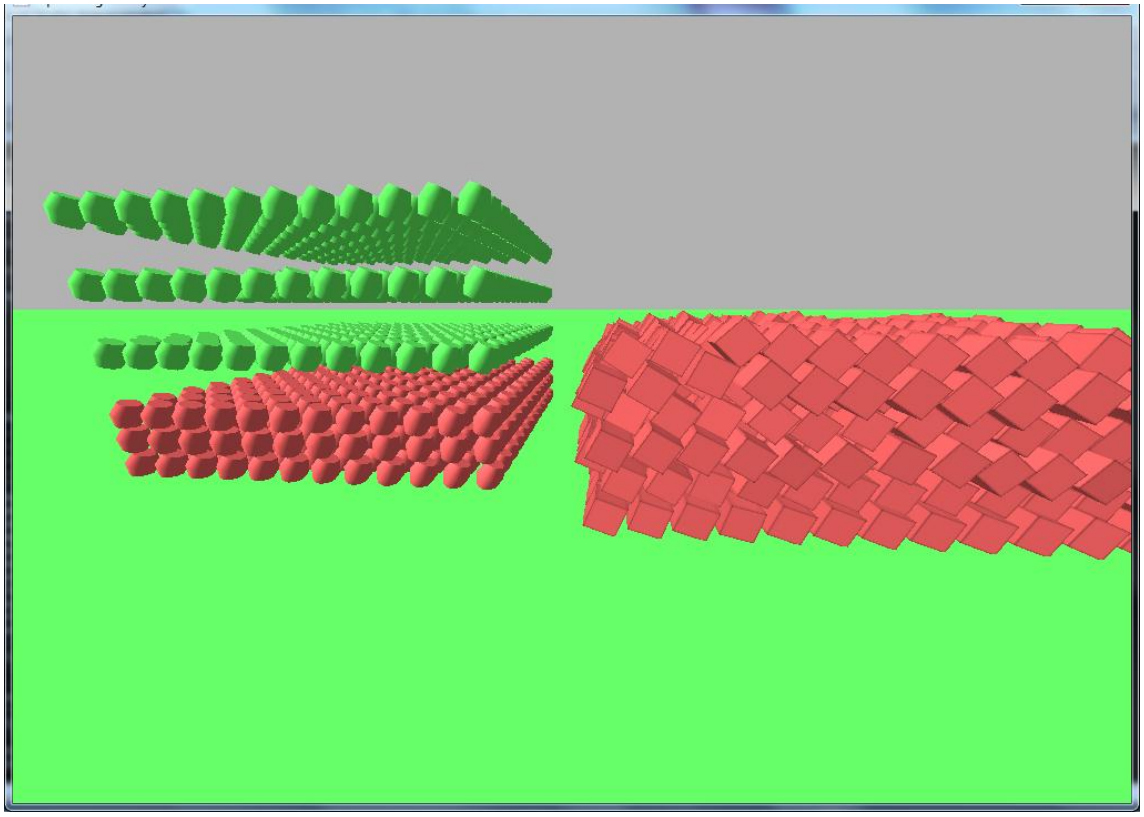


- See https://github.com/erwincoumans/experiments/blob/master/opencl/gpu_rigidbody_pipeline2/sat.cl

Contact Reduction

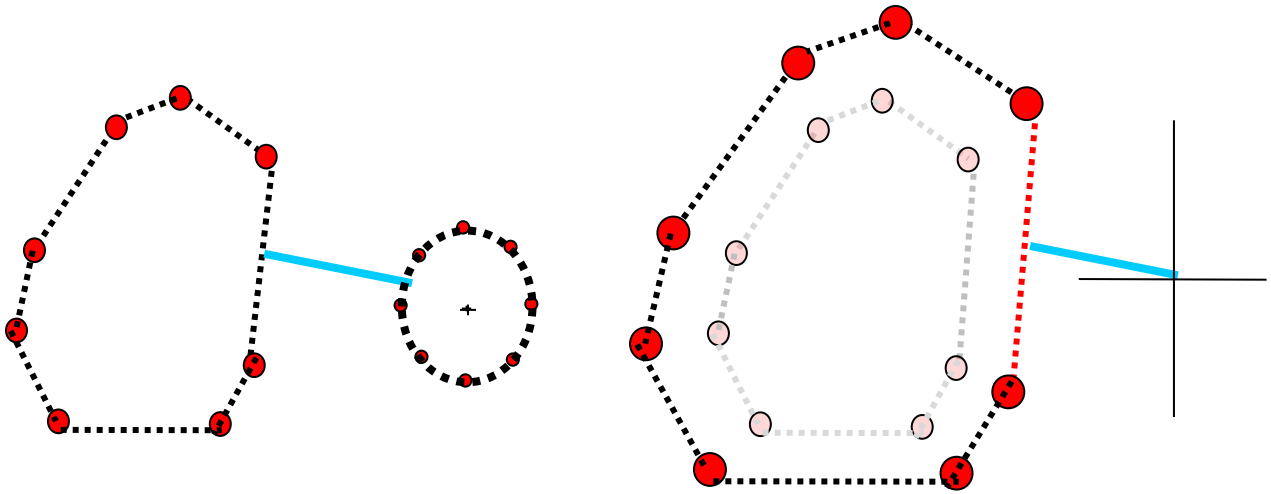


GPU SAT collision detection



- Full source code and windows precompiled executable at https://github.com/erwincoumans/experiments

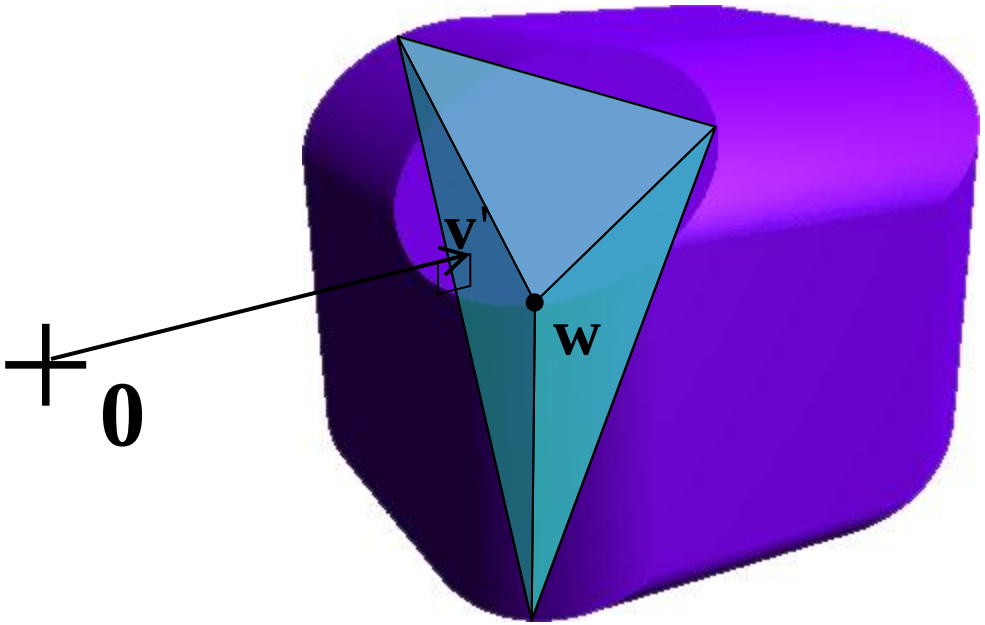
GJK and Minkowski Sum



$$s_{A+B}(\mathbf{v}) = s_A(\mathbf{v}) + s_B(\mathbf{v})$$

$$s_{A-B}(\mathbf{v}) = s_A(\mathbf{v}) - s_B(-\mathbf{v})$$

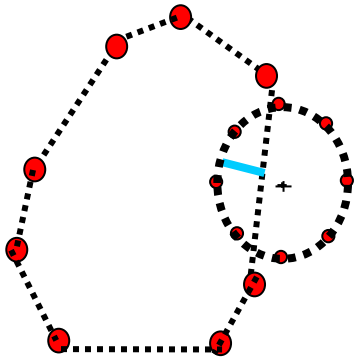
GJK closest point computation



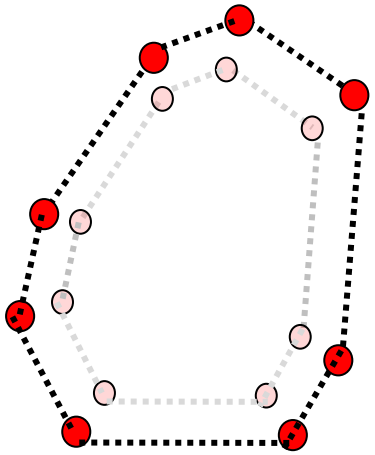
- Slide taken from "gdc2006_vandenBergen_Gino_Physics_Tut.ppt"
  See http://dtecta.com

Expanding Polytope Algorithm (EPA)

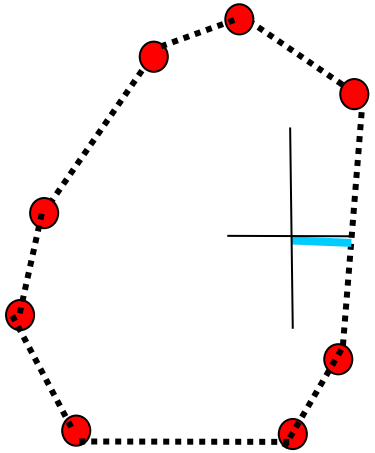
Overlapping objects



Minkowksi Sum

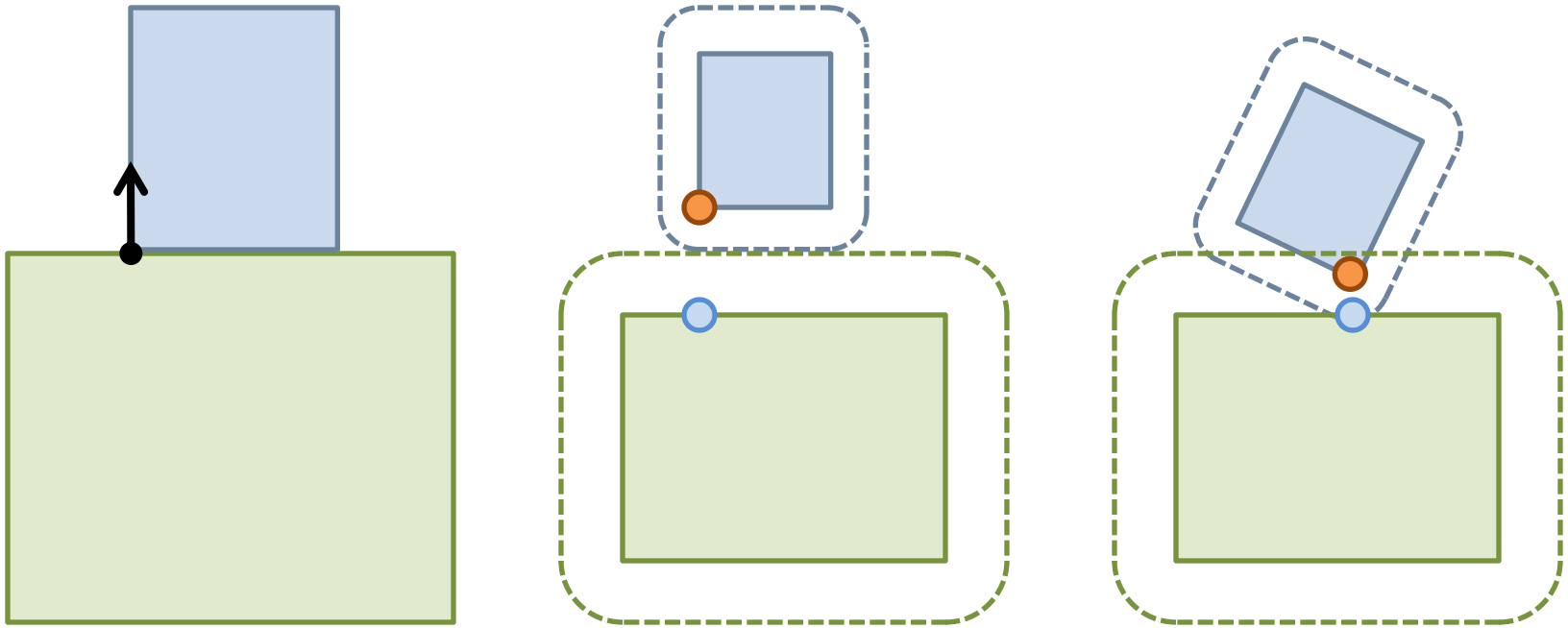


Minimal Translational Distance

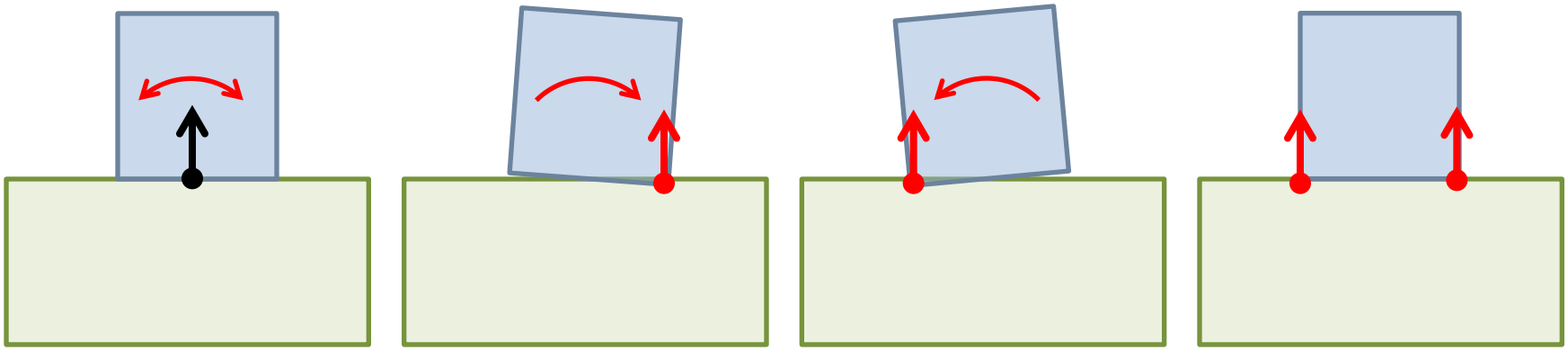


Collision margins

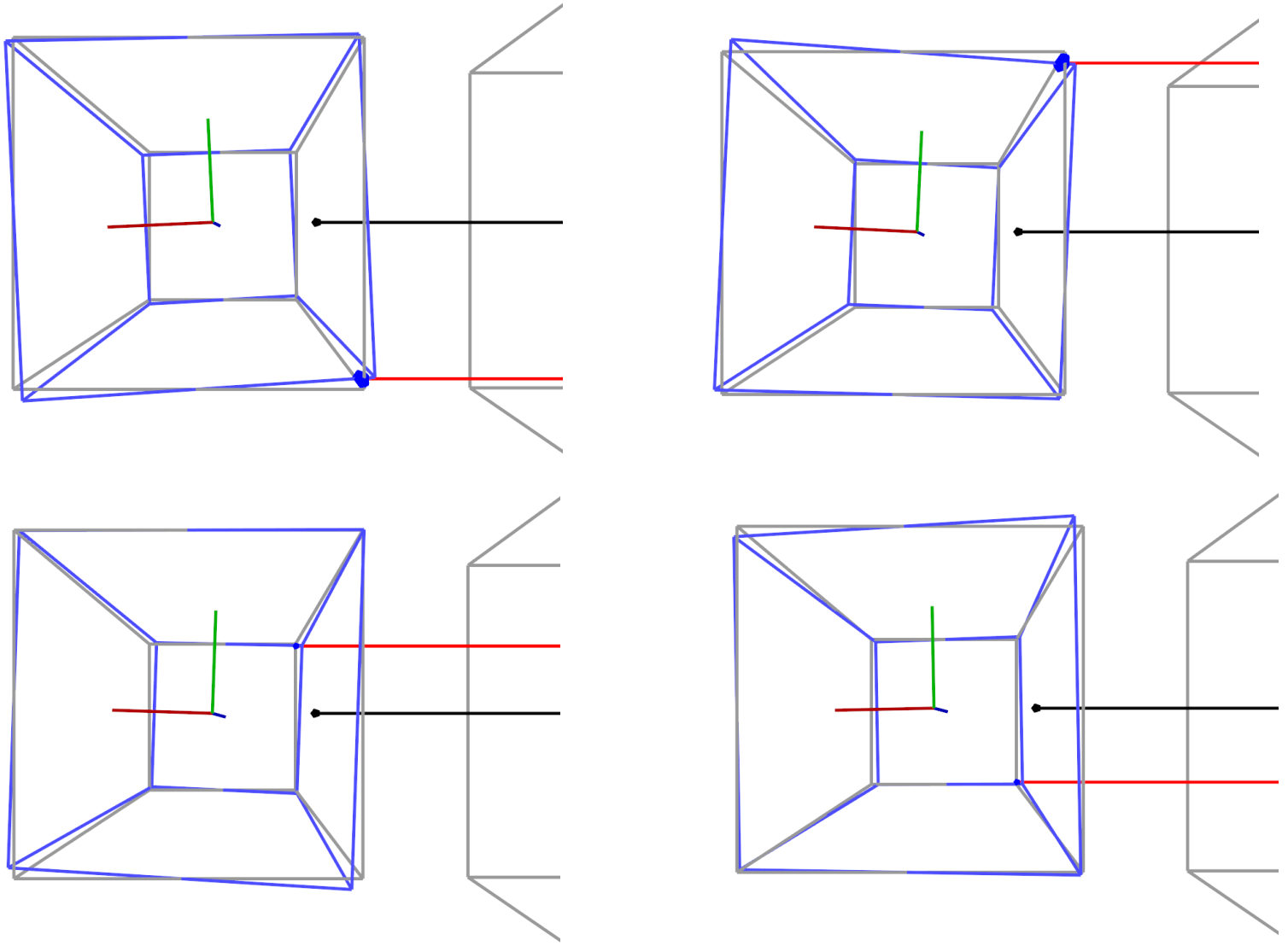
- GJK doesn’t work in penetrating cases
  - and penetration depth calculation is a bit slower



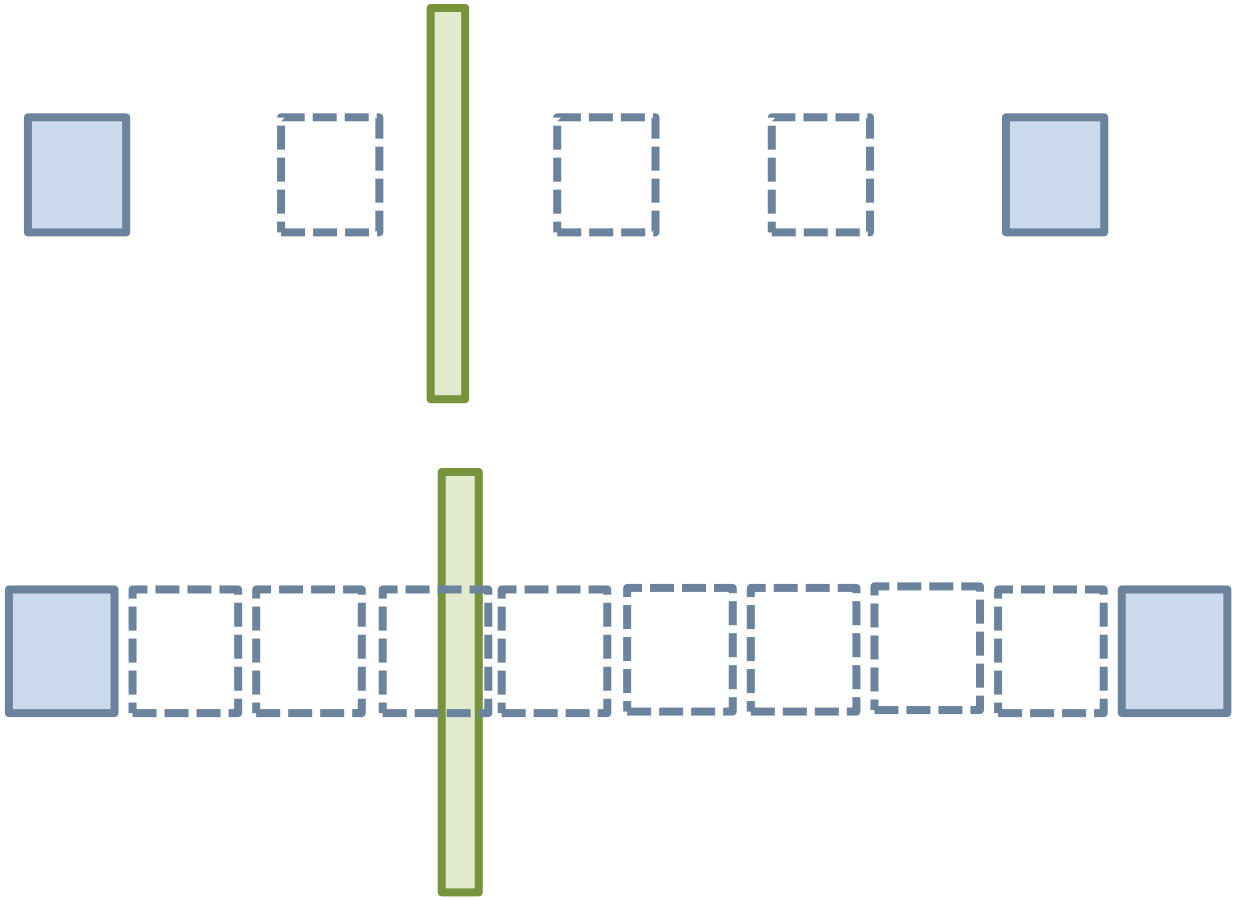
Contact caching and perturbation



Perturbating in 3D

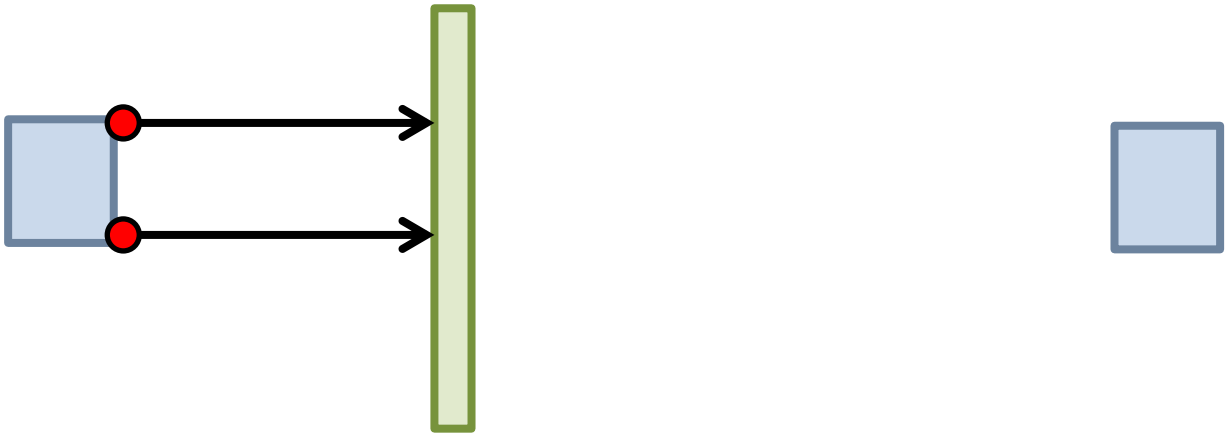


Tunneling in Discrete Physics

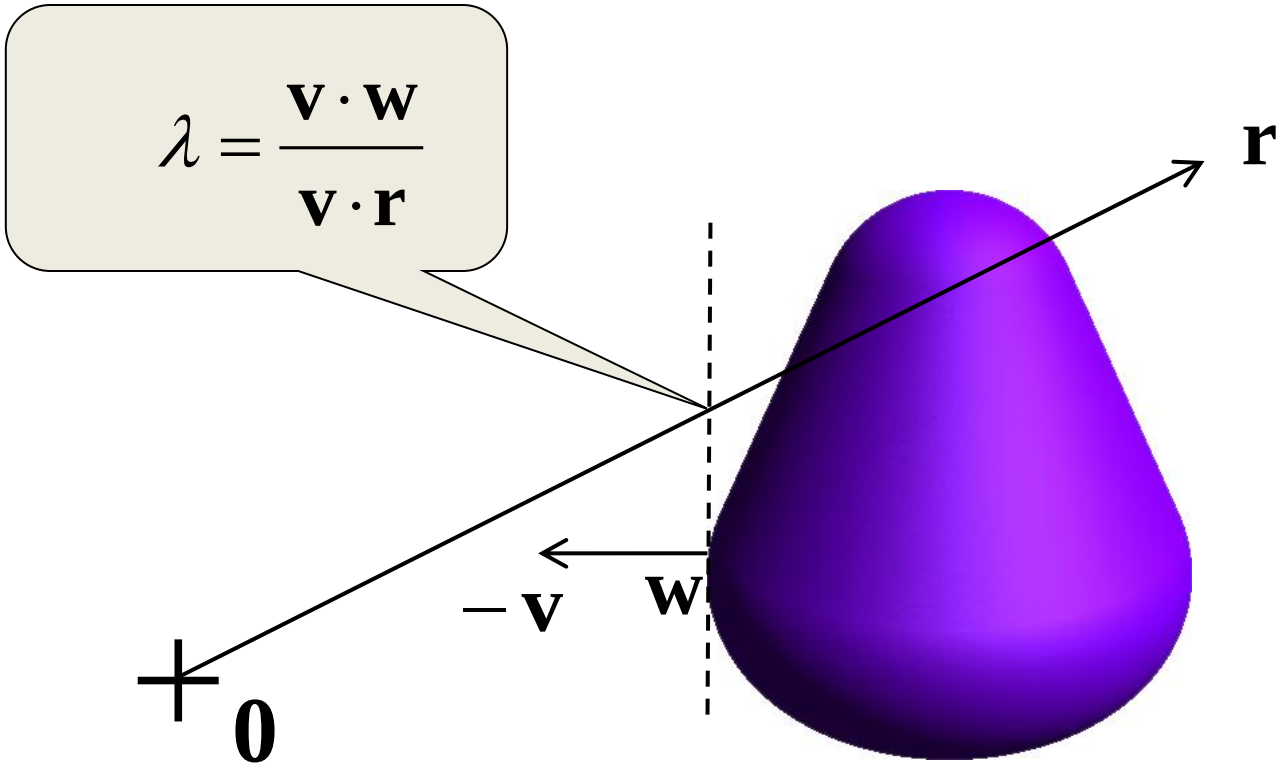


Continuous Collision Detection

- Add potential (future) contact constraints

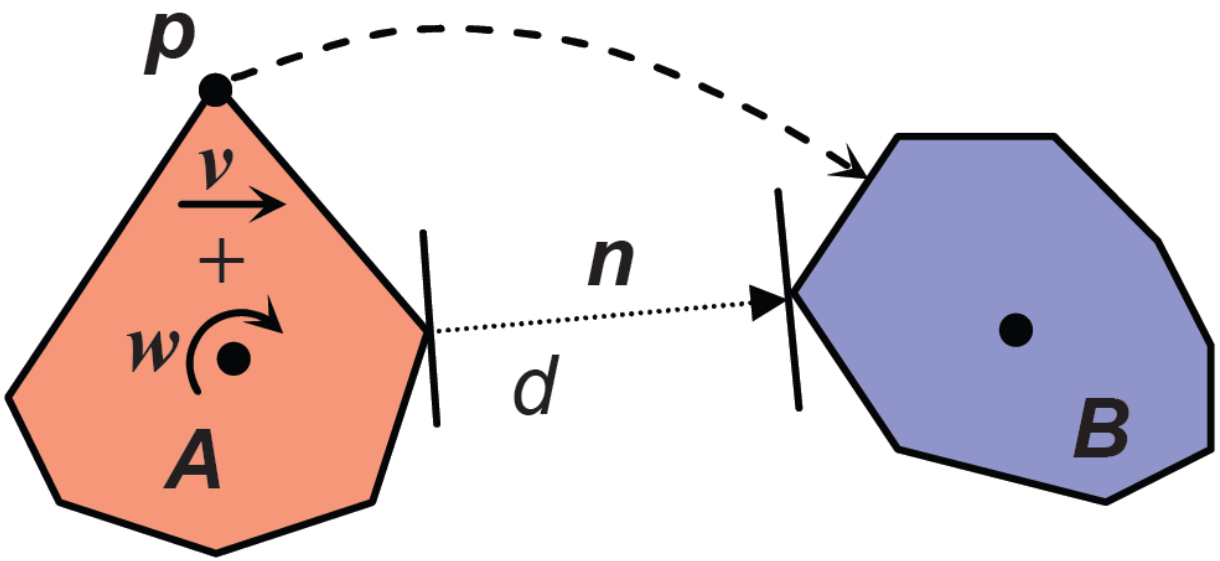


Ray Clipping



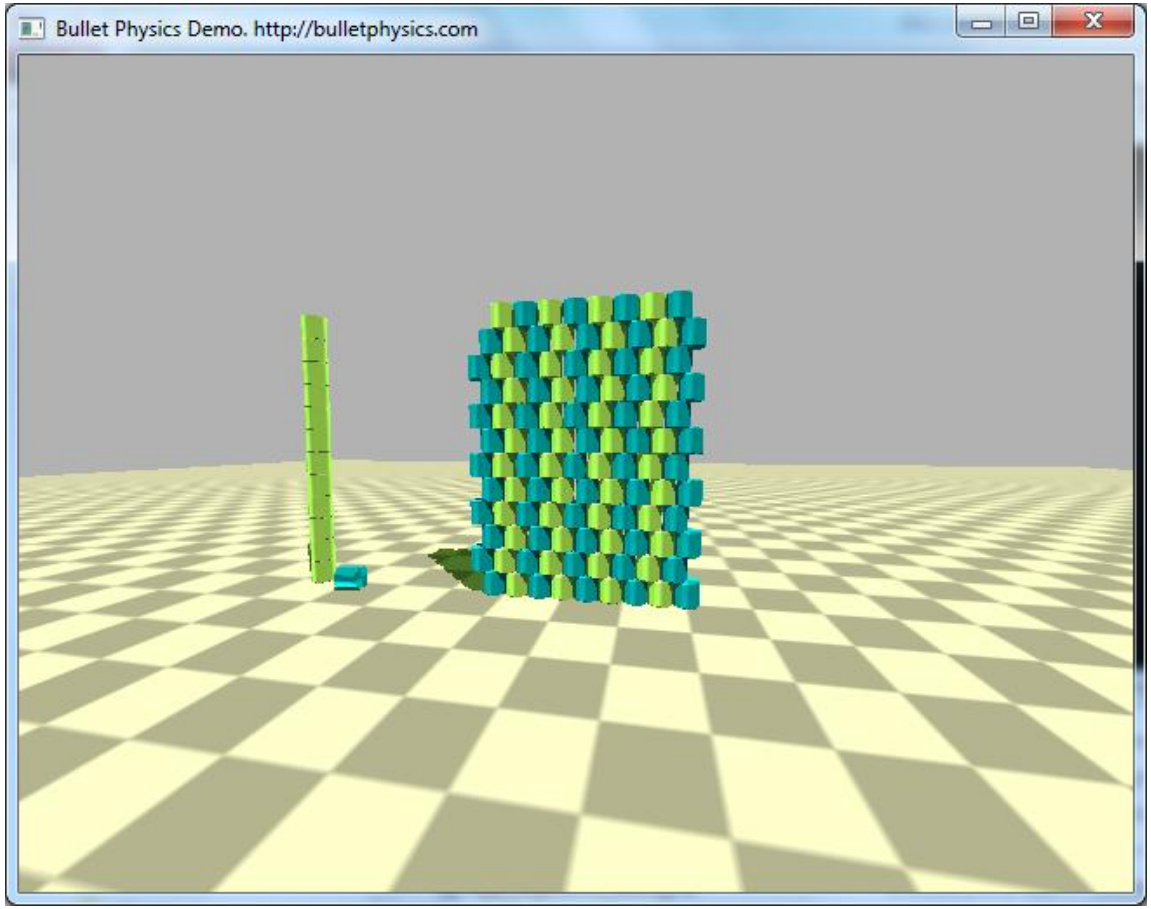
Slide taken from "gdc2006_vandenBergen_Gino_Physics_Tut.ppt"
See http://dtecta.com
http://code.google.com/p/bullet/source/browse/trunk/src/BulletCollision/NarrowPhaseCollision/btGjkConvexCast.cpp

Conservative Advancement



- See http://graphics.ewha.ac.kr/CATCH/
- http://code.google.com/p/bullet/source/browse/trunk/src/BulletCollision/NarrowPhaseCollision/btContinuousConvexCollision.cpp

Demo



Thanks!

Contact

- erwin.coumans@gmail.com
- http://bulletphysics.org
- GPU research: http://github.com/erwincoumans/experiments

References

- Game Physics Pearls, Gino van den Bergen, A.K. Peters
- Real-time Collision Culling of a Million Bodies on Graphics Processing Units, http://graphics.ewha.ac.kr/gSaP
- http://graphics.ewha.ac.kr/CATCH/