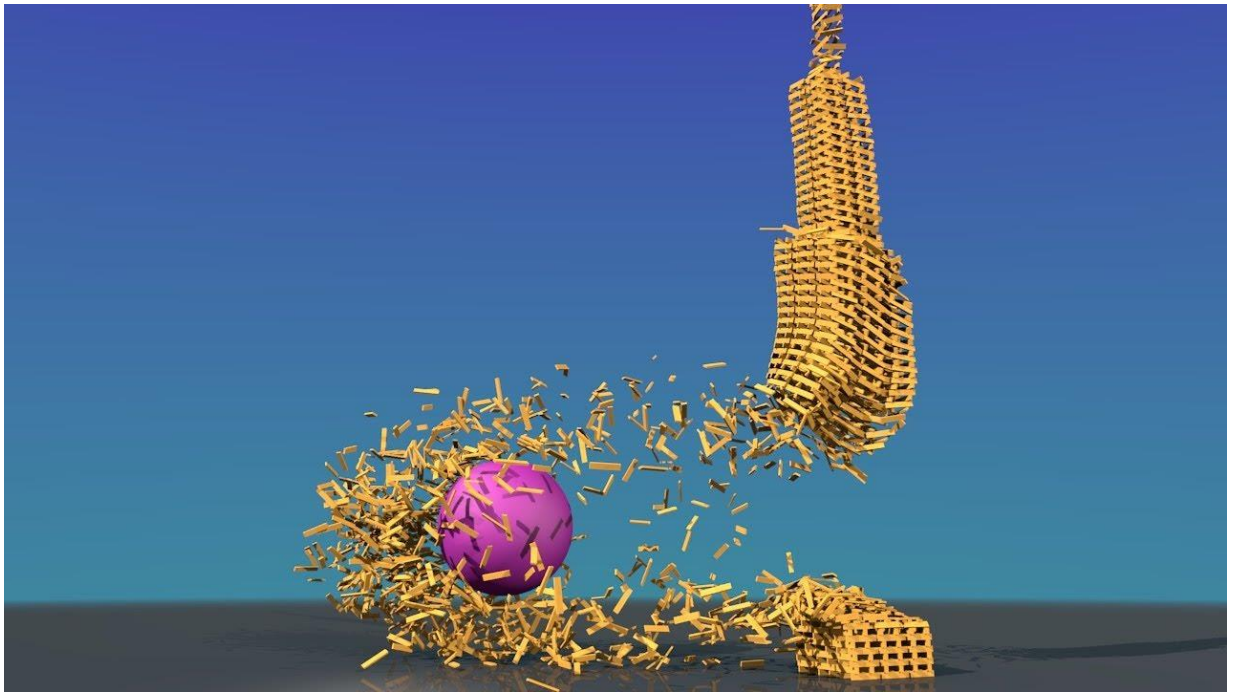
Physics Engines III

Collisions



Stefan Bornhofen

Collisions

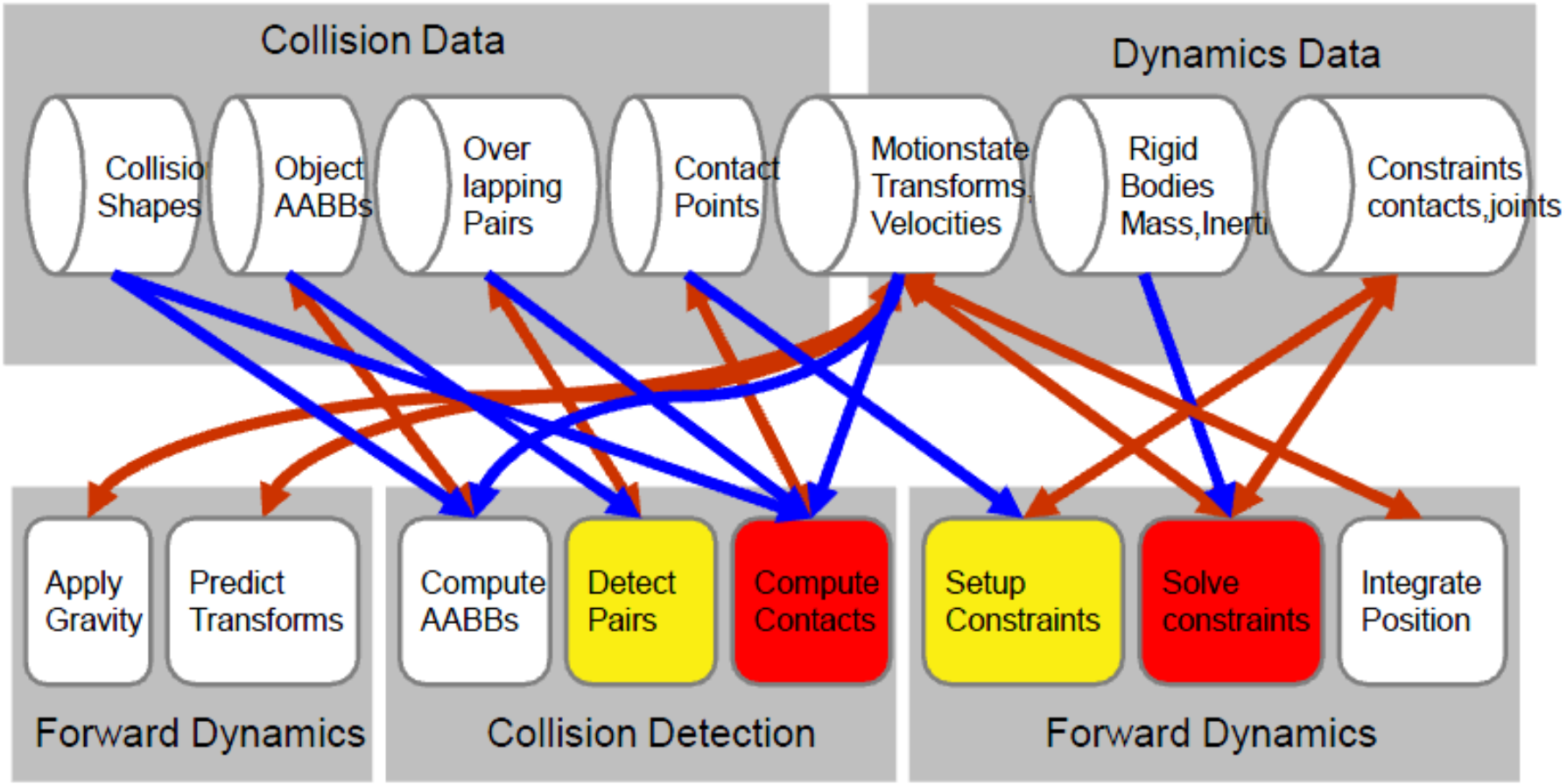
Collisions Detection

Did a collision occur? Where are the collision points?

Collision Resolution

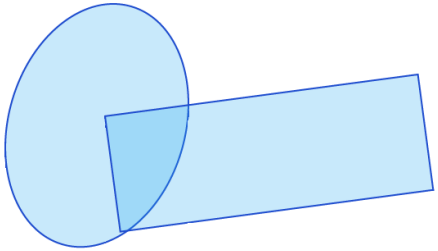
Do the objects bounce? Where do they go?

Performance Bottlenecks

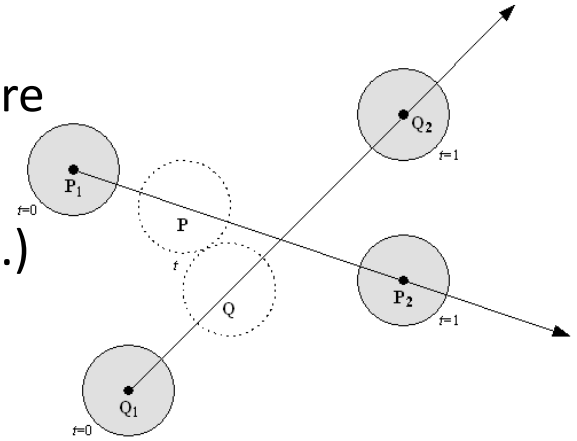


Collision detection

1. Overlap testing (a posteriori)
   - Detects whether a collision has already occurred
   - For every simulation step, test every pair of objects to see if they overlap
   - Broad phase – Narrow phase
   - Most common technique used in games
   - Not accurate but fast

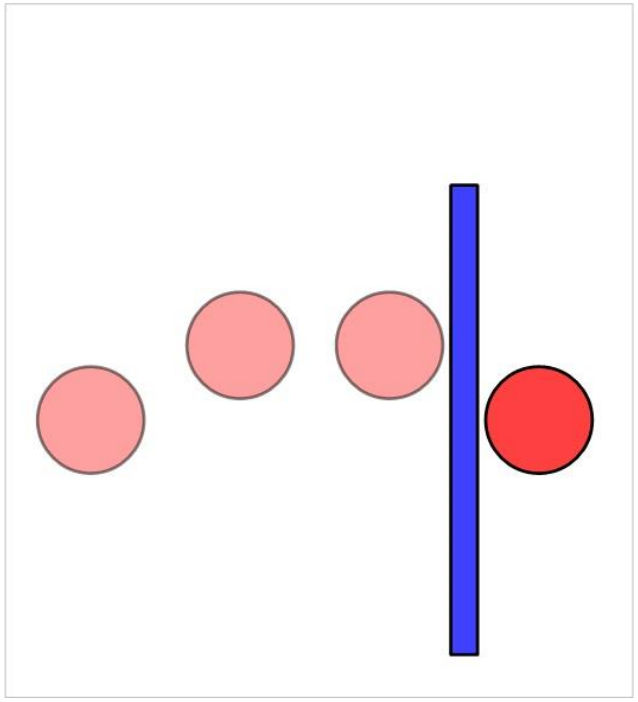


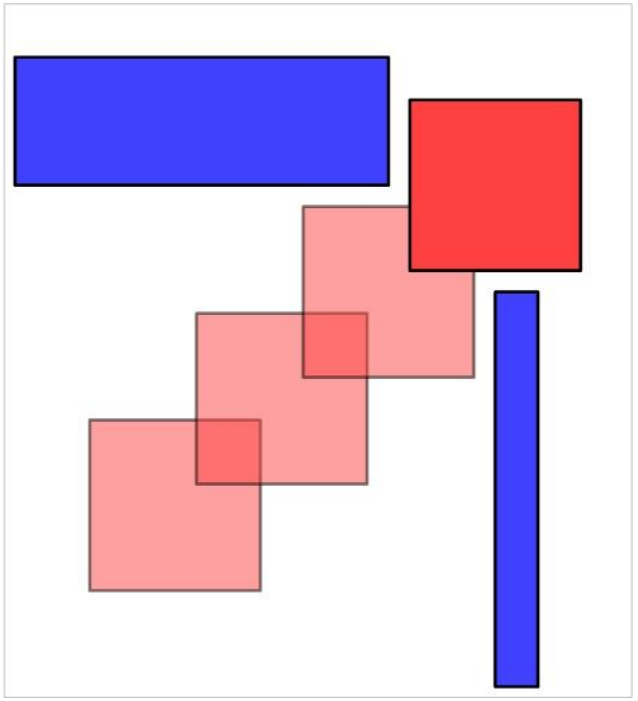
2. Intersection testing, CCD (a priori)
   - Predicts whether a collision will occur in the future
   - Extrude geometry in direction of movement
   - Many approaches (ray casting, swept shapes, etc.)
   - Accurate but expensive



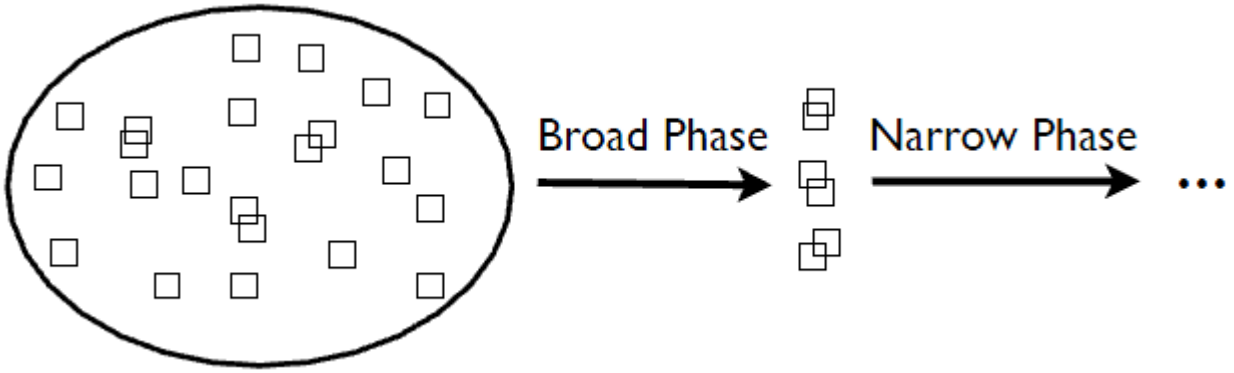
Tunneling

- Tunneling is a false negative for small and fast objects in the overlap testing method.
- This effect is a known limitation of most real-time physics engines and is due to the fact they detect collisions in a discrete manner.
- Collisions of very fast objects such as projectiles of a fire arm should be handled using CCD methods (ray casting).





Broad and Narrow Phase



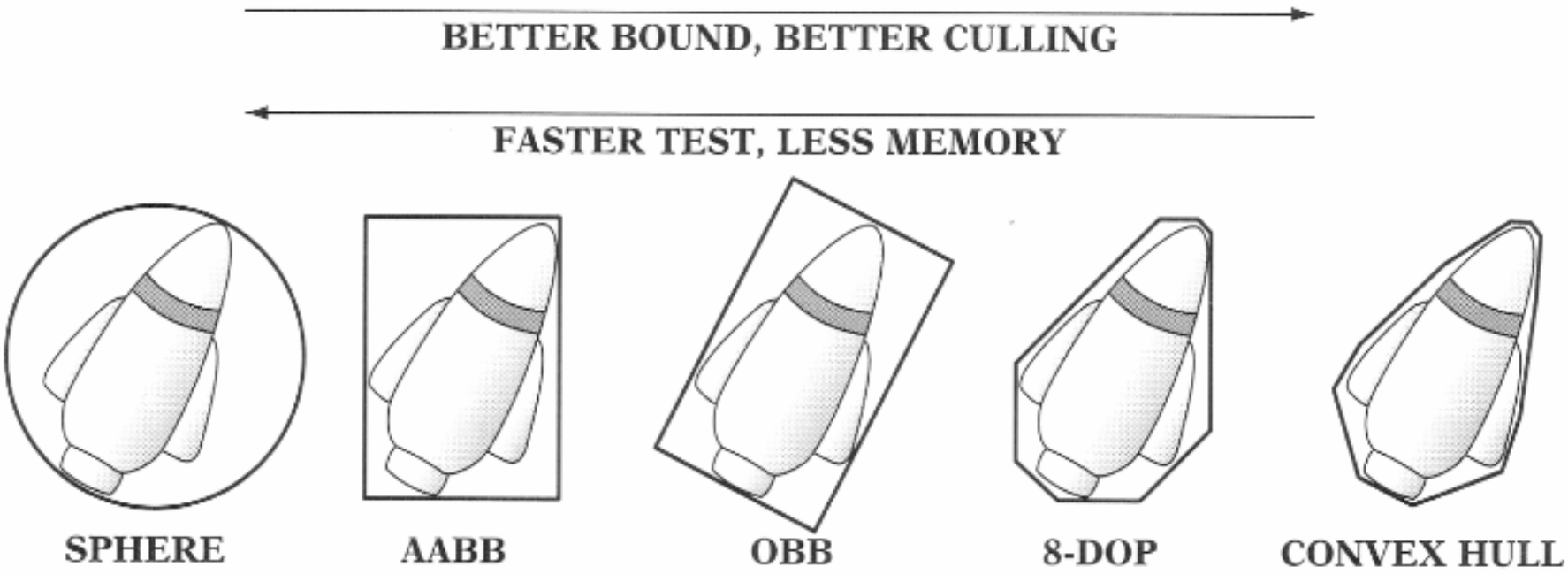
Broad Phase

- Objective: avoid $O(n^2)$ in the narrow phase
- Quickly determine which object pairs potentially collide, *allow false positives*
- Similar to frustum culling in graphics engines
- Uses simple bounding volumes to approximate objects
- Two common optimizations: Spatial subdivision, sweep and prune

Narrow phase

- Determine exact contact information between two shapes
- Computationally expensive

Broad Phase



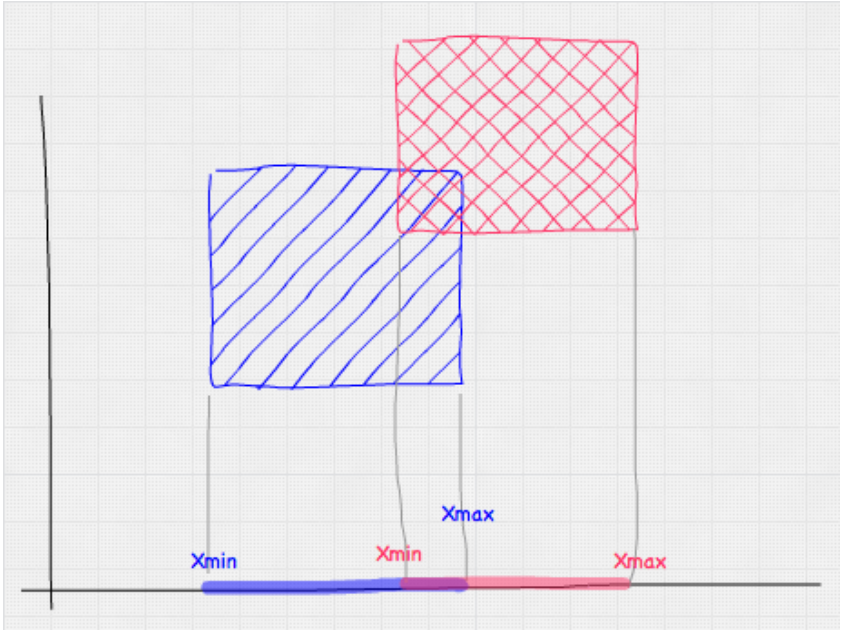
Types of bounding volumes: sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull.

There is a tradeoff between simple and tight bounding volumes.

AABB Intersection Test

Do the ranges A_{minX}–A_{maxX} and B_{minX}–B_{maxX} overlap?

Do this test for each dimension.



$$f(A,B) = (A_{minX} \leq B_{maxX} \wedge A_{maxX} \geq B_{minX}) \wedge (A_{minY} \leq B_{maxY} \wedge A_{maxY} \geq B_{minY}) \wedge (A_{minZ} \leq B_{maxZ} \wedge A_{maxZ} \geq B_{minZ})$$

Narrow Phase: Collision dispatcher

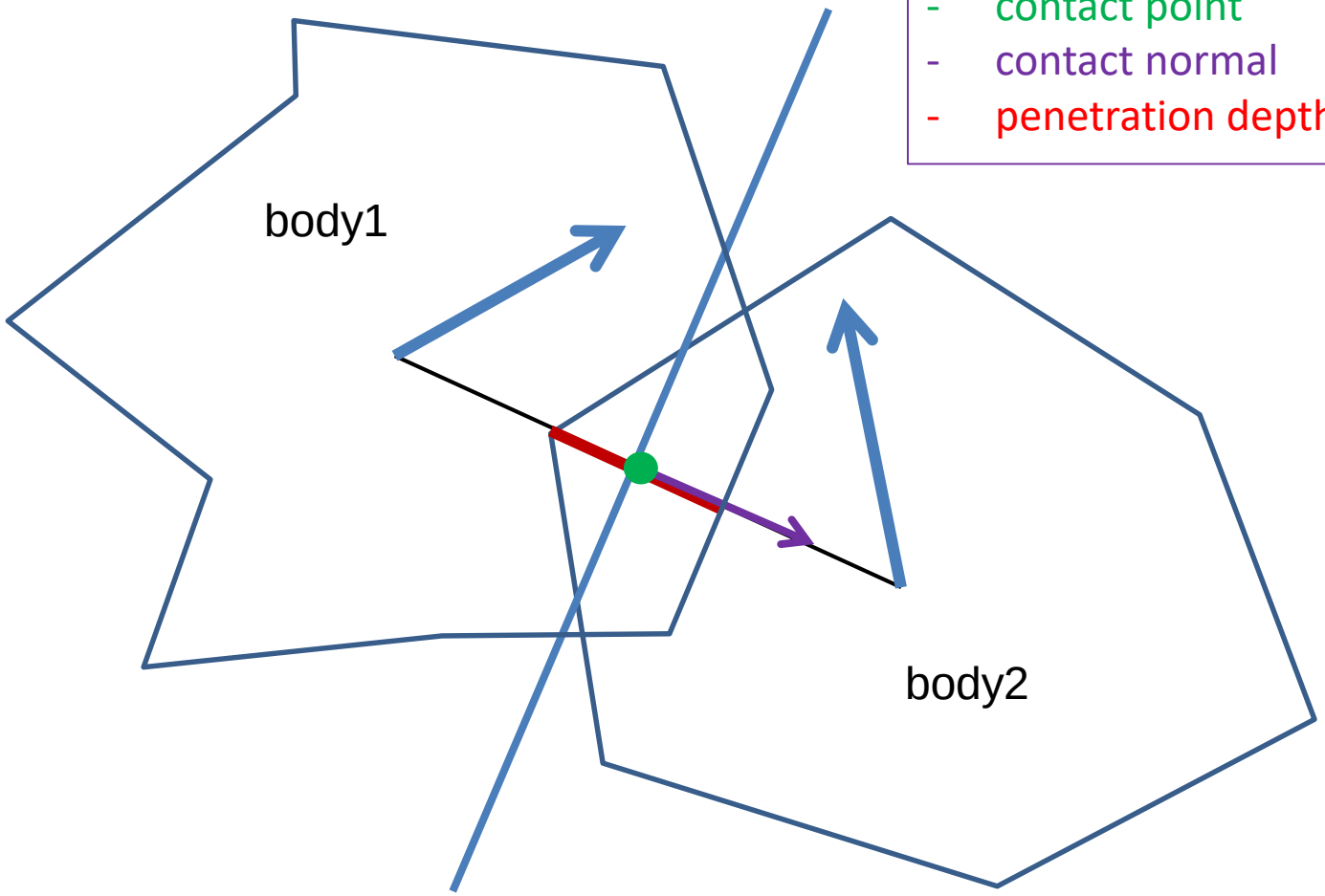
(example: Bullet physics engine)

	BOX	SPHERE	CONVEX, CYLINDER, CONE, CAPSULE	COMPOUND	TRIANGLE MESH
BOX	boxbox	spherebox	gjk	compound	concaveconvex
SPHERE	spherebox	spheresphere	gjk	compound	concaveconvex
CONVEX, CYLINDER, CONE, CAPSULE	gjk	gjk	gjk	compound	concaveconvex
COMPOUND	compound	compound	compound	compound	compound
TRIANGLE MESH	concaveconvex	concaveconvex	concaveconvex	compound	gimpact

gjk: The Gilbert–Johnson–Keerthi distance algorithm is a method of determining the minimum distance between two convex sets. The algorithm provides a linear time complexity, dependent on the number of vertices of which the pair of objects consists.

Narrow Phase

Compute <u>contact information</u>
- contact point
- contact normal
- penetration depth



Collision response

- When two solid objects collide, forces are generated at the impact location that prevent the objects from interpenetrating
- These forces act over a very small time and as far as the simulation is concerned, it is easier to treat it as an instantaneous event
- Therefore, instead of the impact applying a force, we use an impulse which directly affects the velocity

Elastic vs. Inelastic Collisions

- The collision can be thought of as having two phases: compression & restitution
- In the compression phase, the energy of the two objects is changed from kinetic energy of motion into deformation energy in the solids
- If the collision is perfectly elastic, then all of the deformation energy will be turned back into kinetic energy in the restitution phase and the velocity along the normal will be the opposite of what it was before the collision
- If the collision is perfectly inelastic, then all of the energy is lost and there will be no relative motion along the collision normal after the collision
- We model inelastic collisions via a restitution coefficient per object

The kinetic energy of an object is the energy that it possesses due to its motion. It is defined as

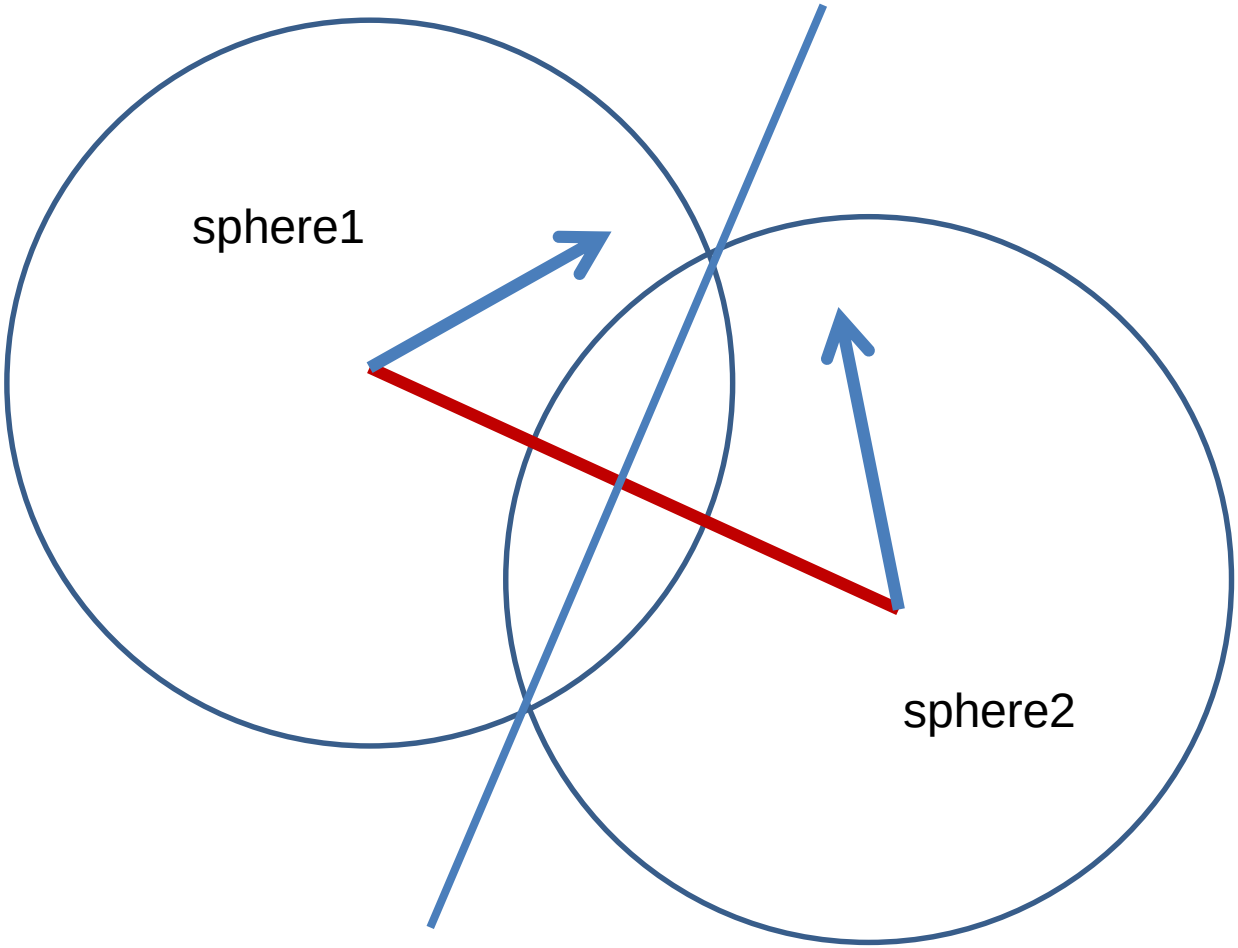
$$\frac{1}{2} * m * v^2$$

Collision Resolution: Resolving Overlap Testing

1. Compute contact information
2. Move the two objects apart
3. Compute new velocities

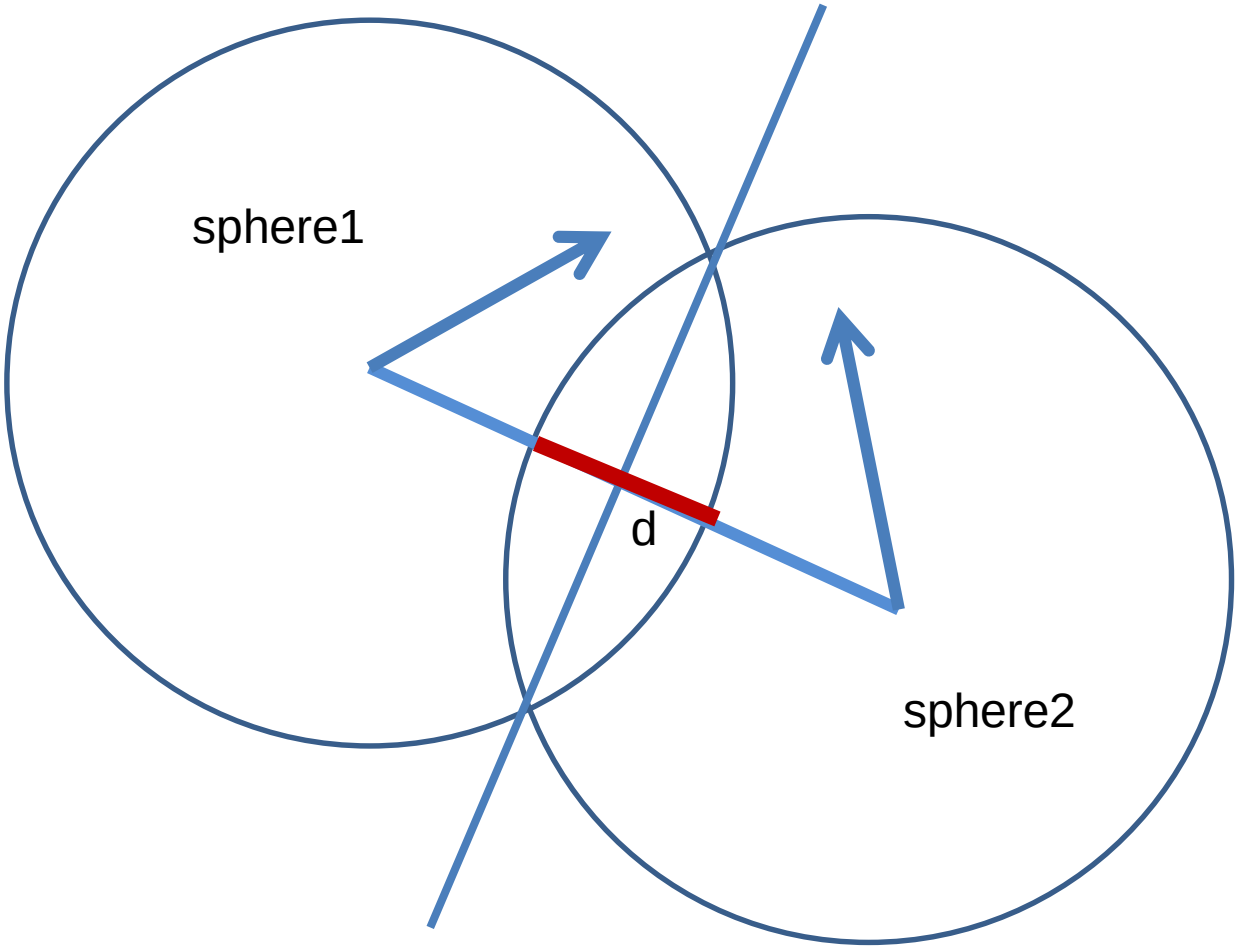
Collision Resolution: Sphere-Sphere

1. Extract collision normal



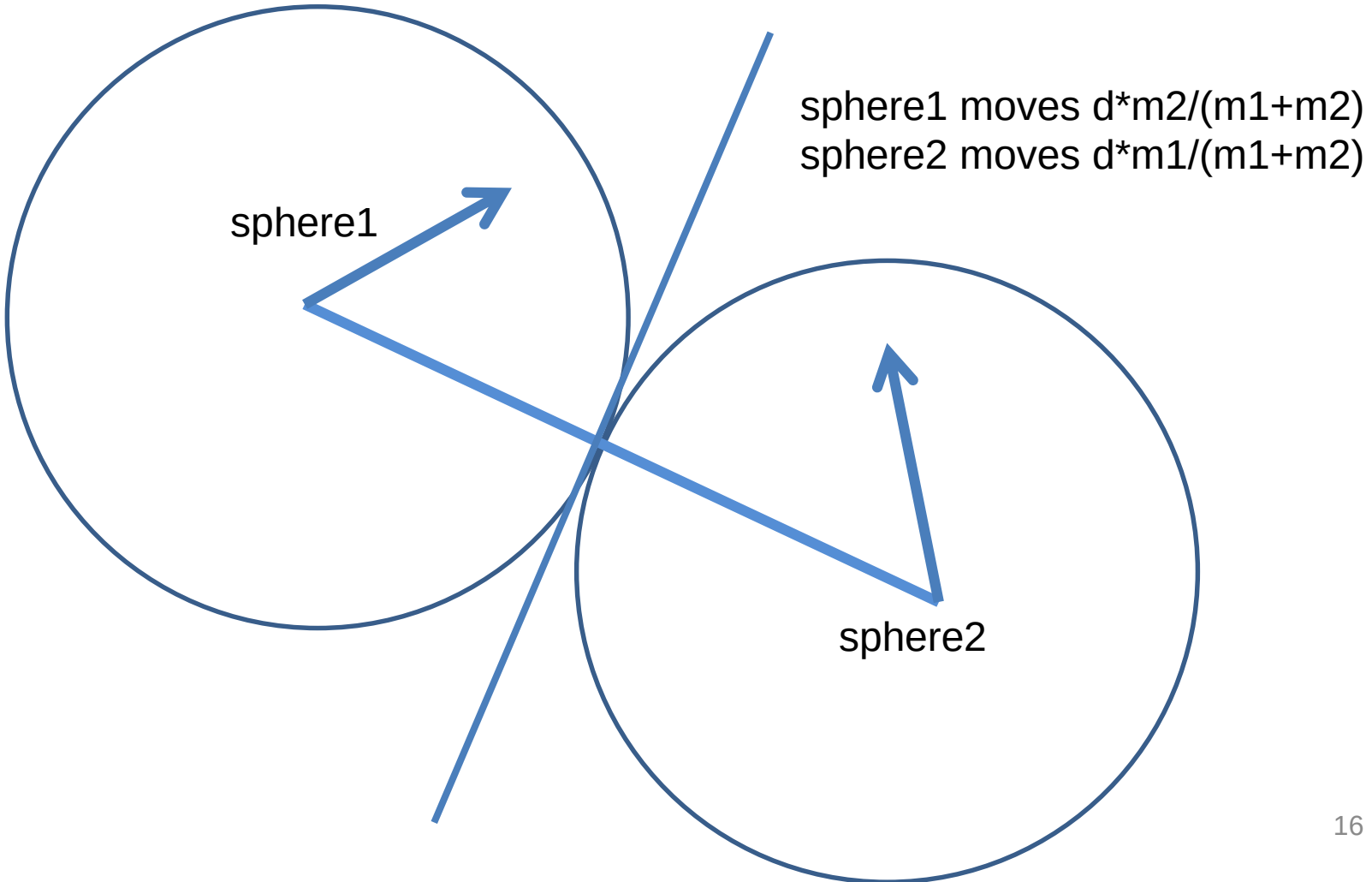
Collision Resolution: Sphere-Sphere

2. Extract penetration depth



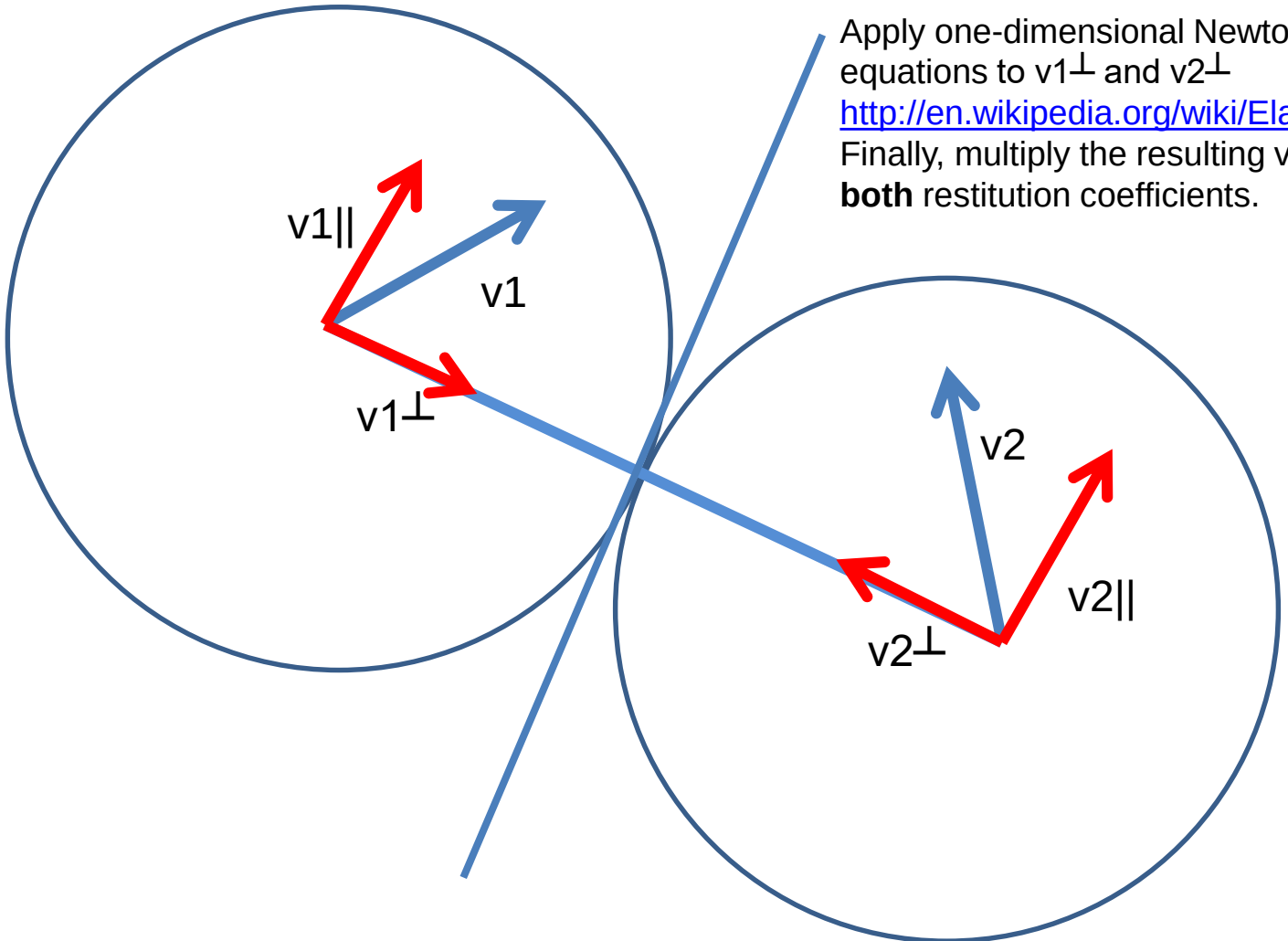
Collision Resolution: Sphere-Sphere

3. Move the objects apart



Collision Resolution: Sphere-Sphere

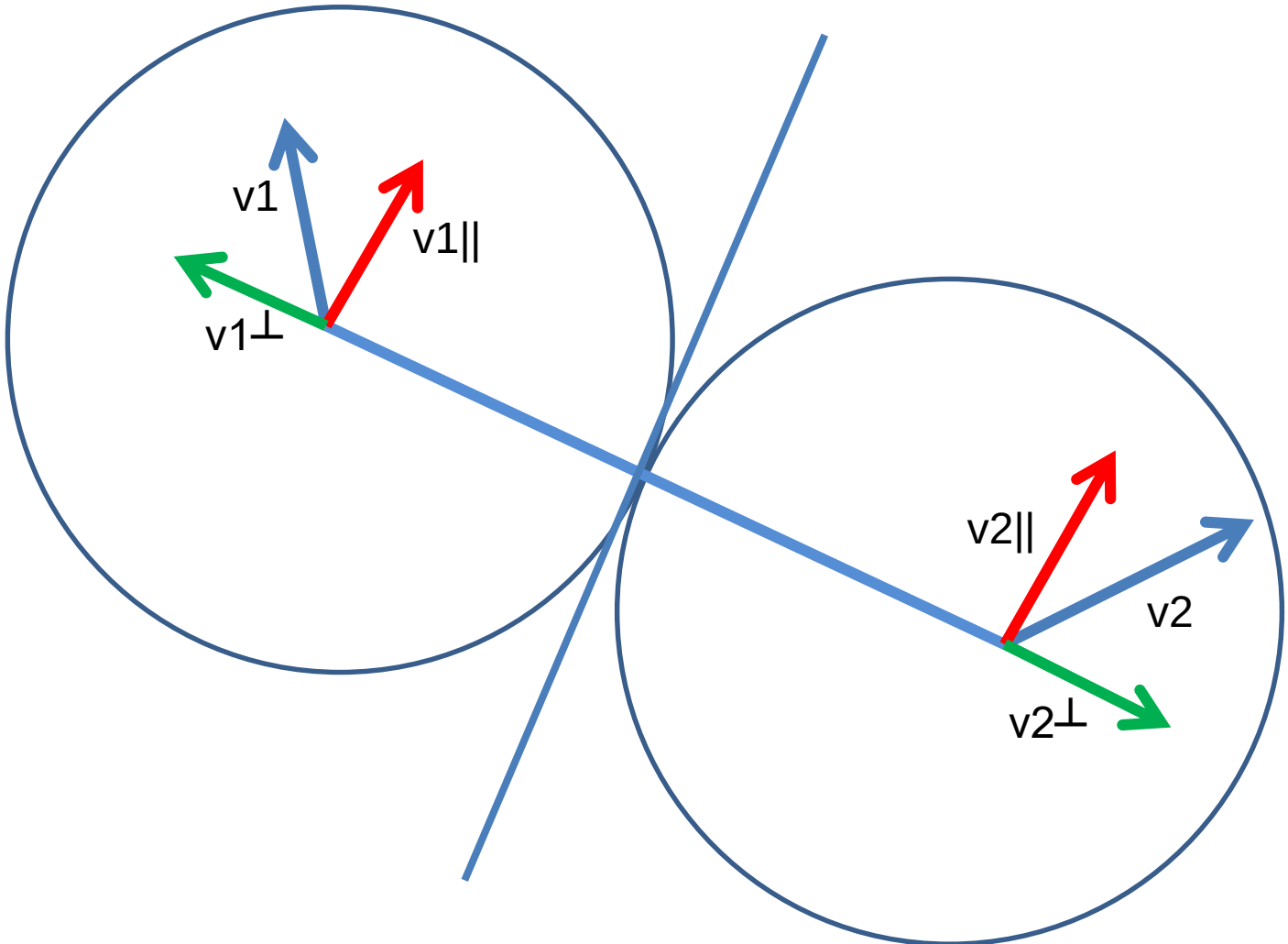
4. Compute new velocities



Apply one-dimensional Newtonian collision equations to $v1^{\perp}$ and $v2^{\perp}$
http://en.wikipedia.org/wiki/Elastic_collision
Finally, multiply the resulting velocities with **both** restitution coefficients.

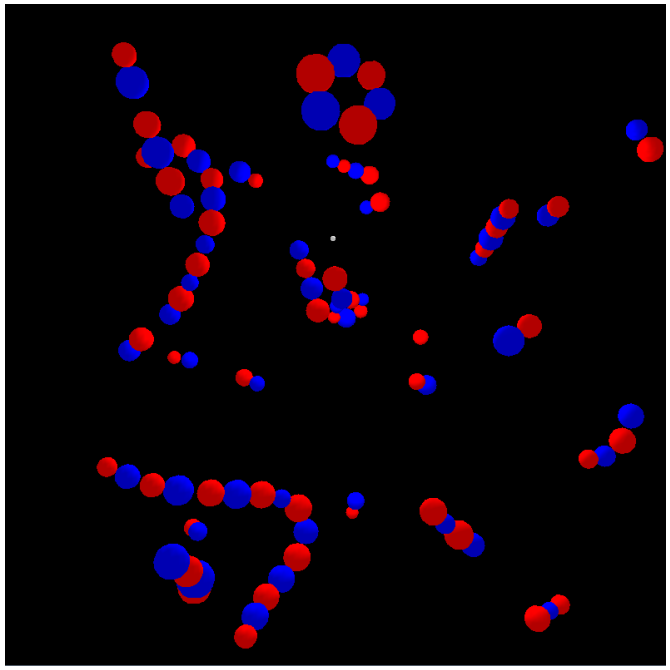
Collision Resolution: Sphere-Sphere

5. Nice job!



Particle-Particle collisions

- Add a radius to your particles so that they become a sphere, and implement elastic particle-particle collisions using broad and narrow phase collision detection. For now, broadphase can be a global $O(n^2)$ test on the AABB without further optimizations.
- The user can toggle collisions on/off. You can observe collisions in any of the previous scenes, but most remarkably in scene 2 (magnetism).



QmWorld

...

---

void simulate(float t) {
  void resetBodies()
  void applyGravity()
  void updateForces ()
  void integrate (float t)
  resolve(narrowphase(broadphase()));
}
List<QmContact> broadphase()
List<QmContact> narrowphase(List<QmContact> c)
void resolve(List<QmContact> c)

QmContact

---

body1: QmBody*
body2: QmBody*
normal: Vector3
depth: float

QmParticle

...
radius: float
restitution: float

...
QmAABB getAABB()

```
bool intersect(QmAABB a, QmAABB b) {
  return
    (a.min.x <= b.max.x && a.max.x >= b.min.x) &&
    (a.min.y <= b.max.y && a.max.y >= b.min.y) &&
    (a.min.z <= b.max.z && a.max.z >= b.min.z);
}
```

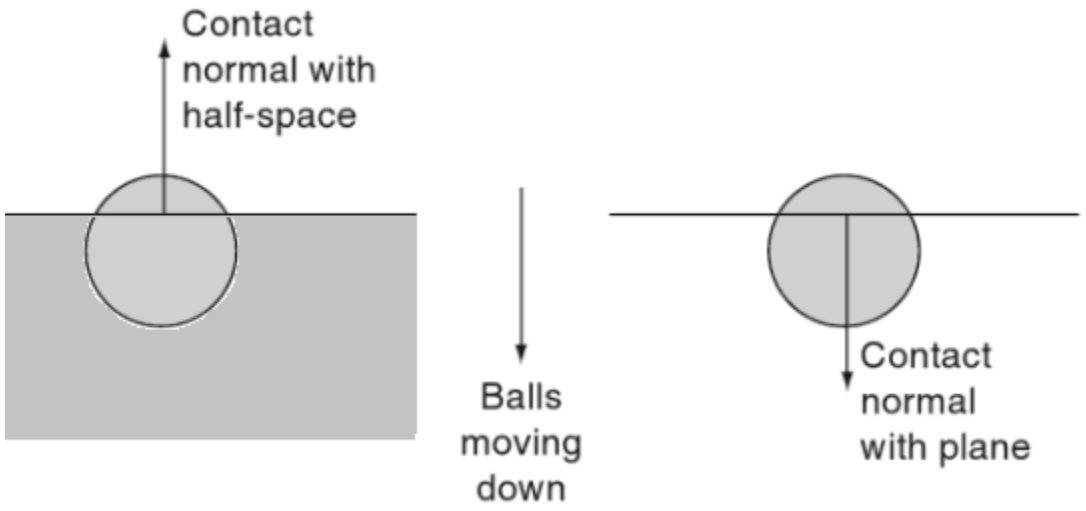
QmAABB

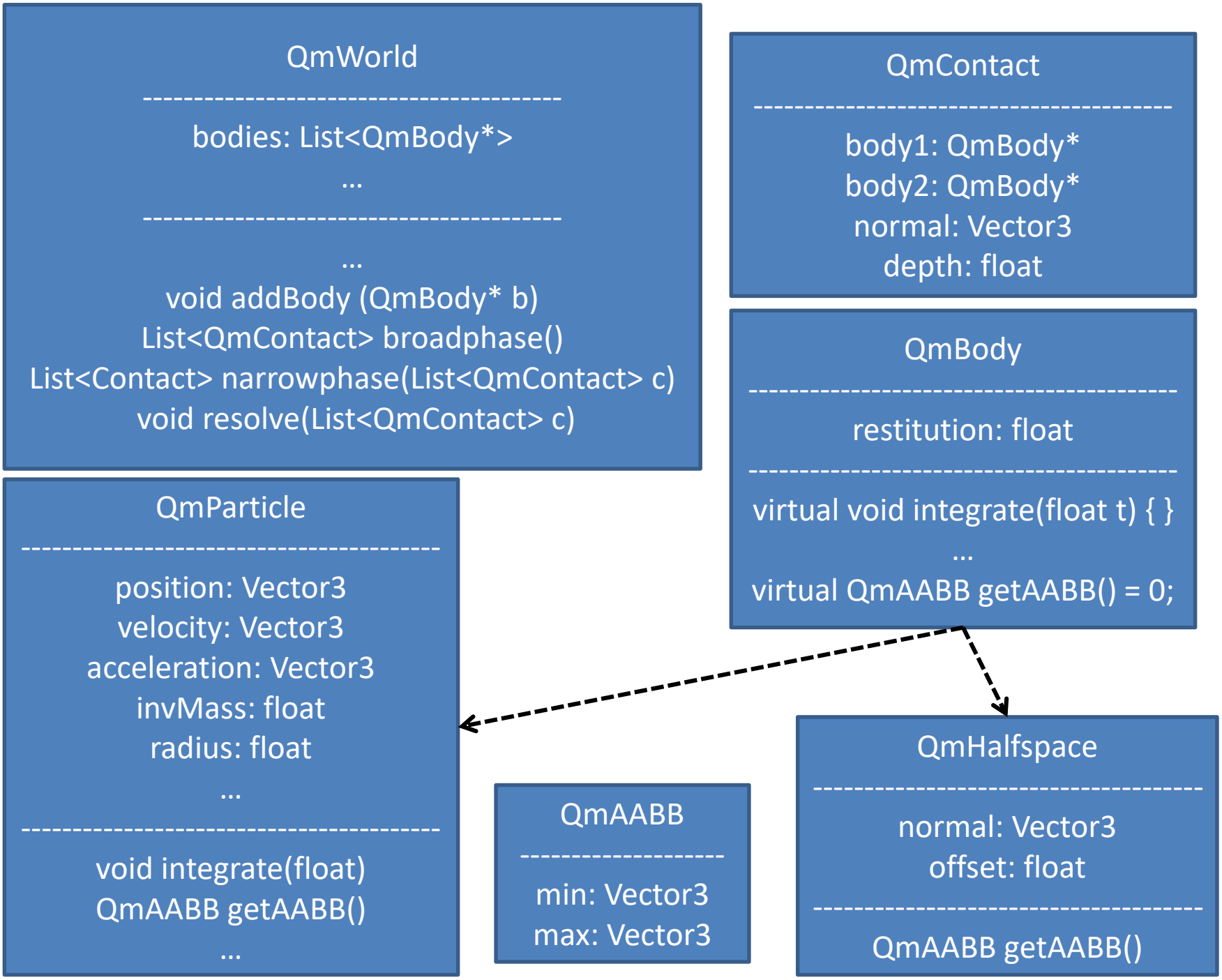
---

min: Vector3
max: Vector3

Halfspace

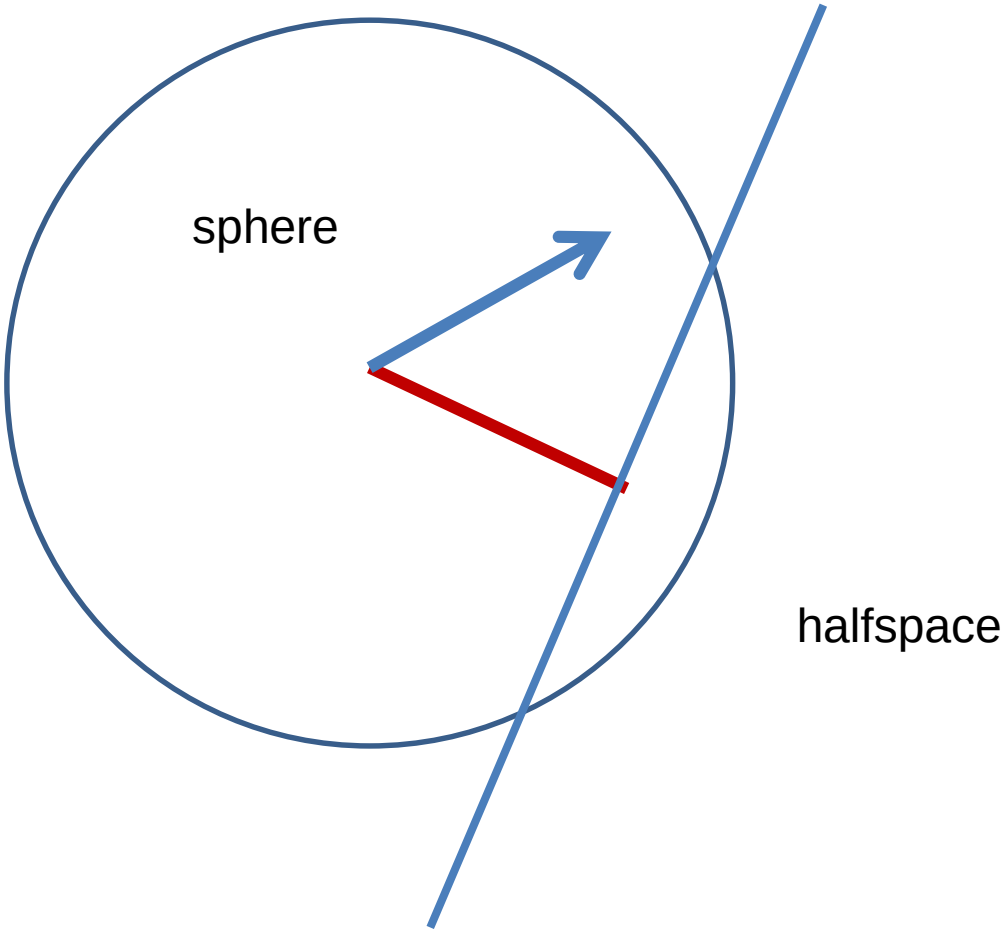
- A halfspace is a plane where the whole region on the back of the plane is solid.
- An object interpenetrating will always have its contact normal pointing in the direction of the plane normal, even if it is completely behind the plane.




QmWorld
bodies: List<QmBody*>
...
...
void addBody (QmBody* b)
List<QmContact> broadphase()
List<Contact> narrowphase(List<QmContact> c)
void resolve(List<QmContact> c)
QmContact
body1: QmBody*
body2: QmBody*
normal: Vector3
depth: float
QmBody
restitution: float
virtual void integrate(float t) { }
...
virtual QmAABB getAABB() = 0;
QmParticle
position: Vector3
velocity: Vector3
acceleration: Vector3
invMass: float
radius: float
...
void integrate(float)
QmAABB getAABB()
...
QmAABB
min: Vector3
max: Vector3
QmHalfspace
normal: Vector3
offset: float
QmAABB getAABB()

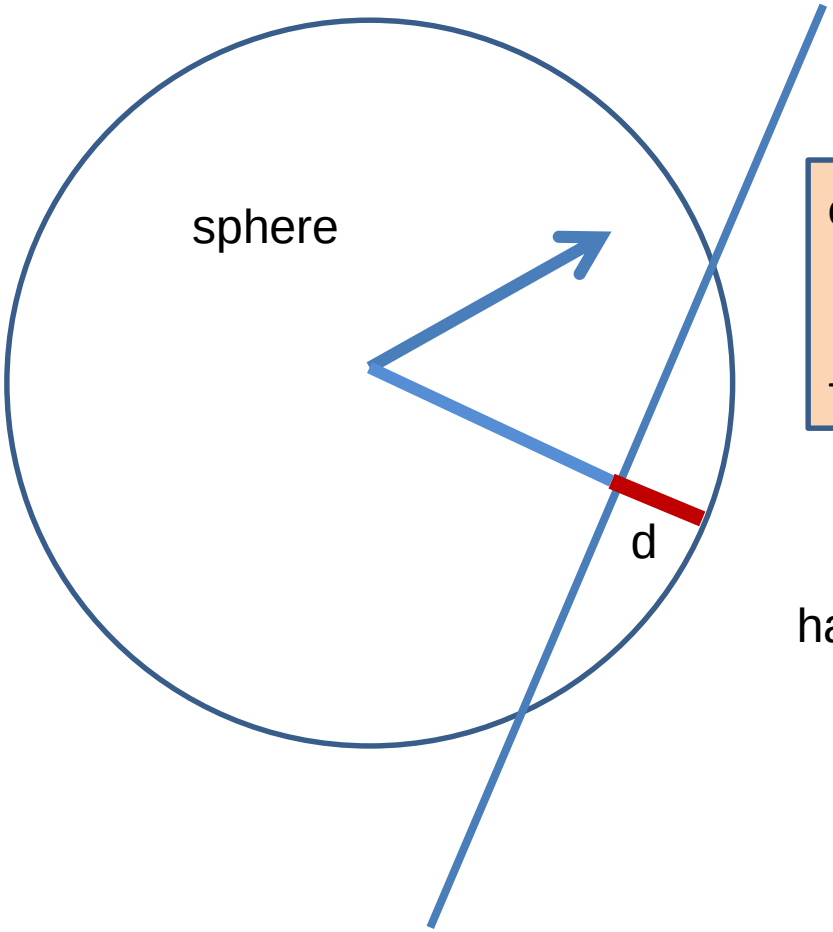
Collision Resolution: Sphere-Halfspace

1. Extract collision normal



Collision Resolution: Sphere-Halfspace

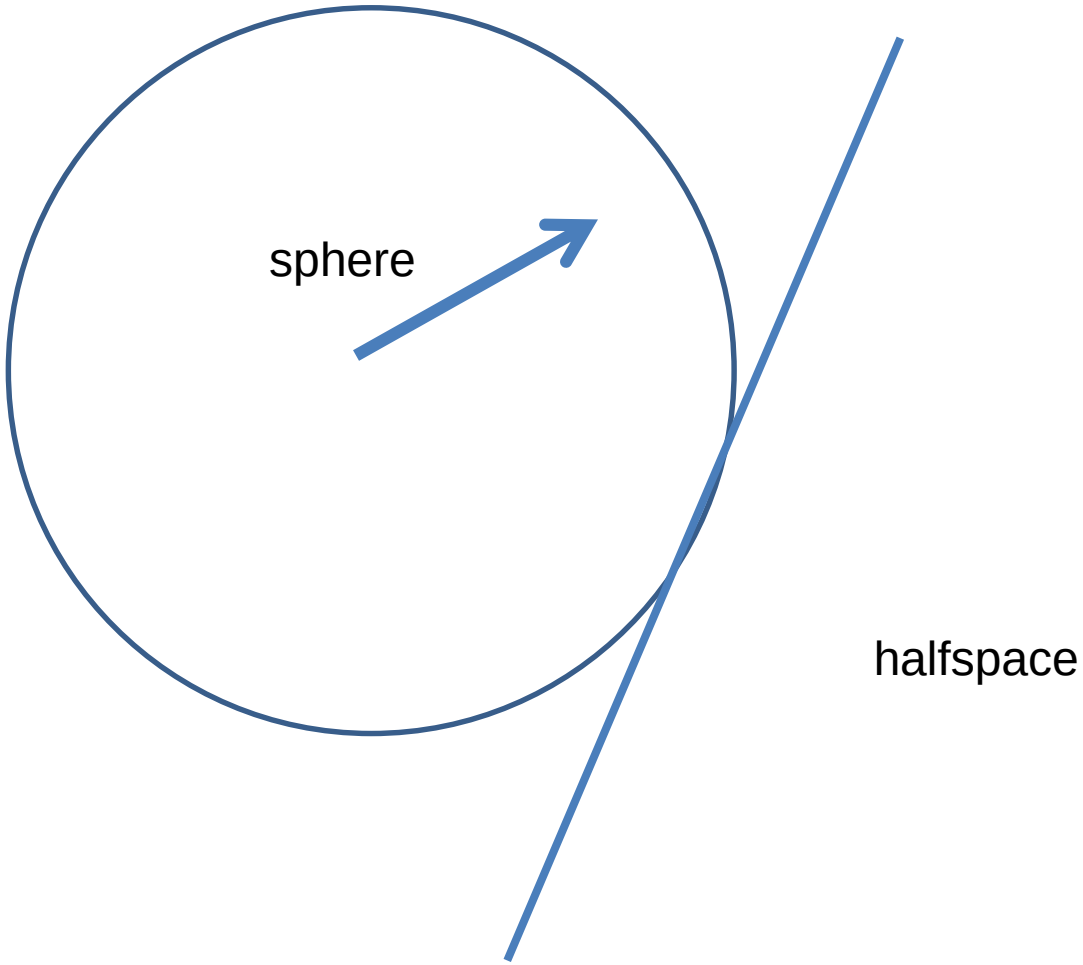
2. Extract penetration depth



d =
halfspace->offset
+ sphere->radius
- < sphere->pos, halfspace->normal >

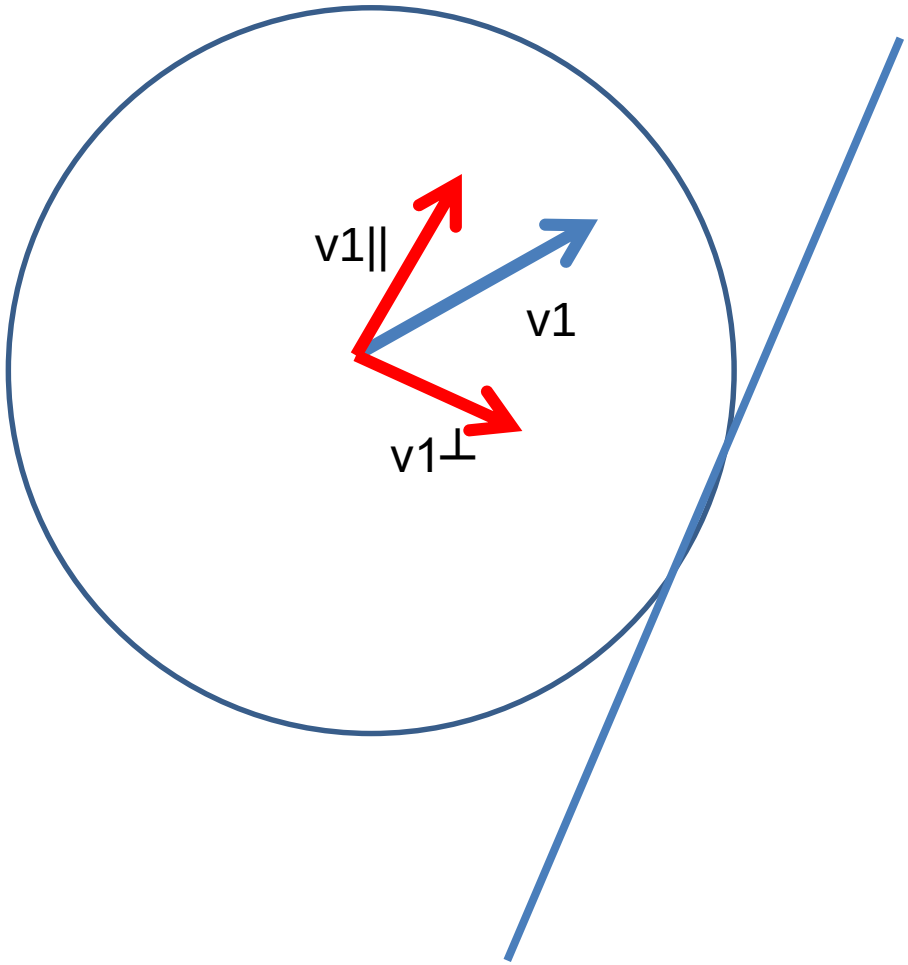
Collision Resolution: Sphere-Halfspace

3. Move the sphere



Collision Resolution: Sphere-Halfspace

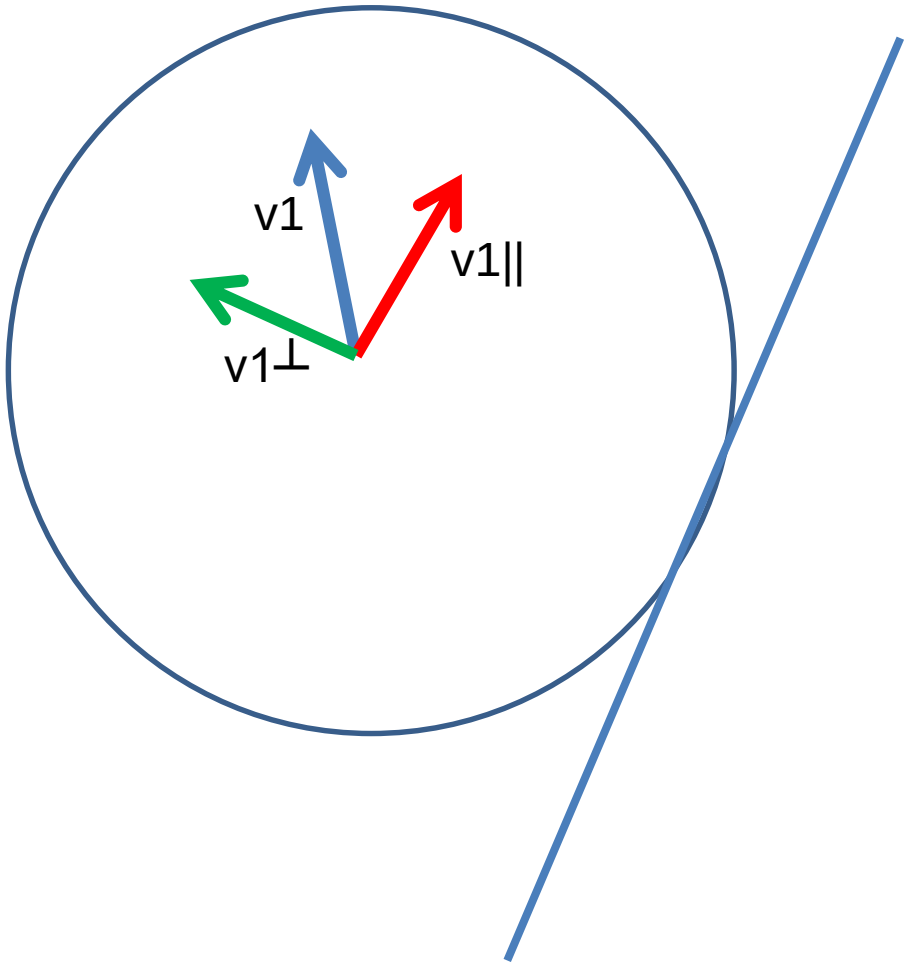
4. Compute new velocity



Invert $v1^{\perp}$
Multiply with both restitution coefficients

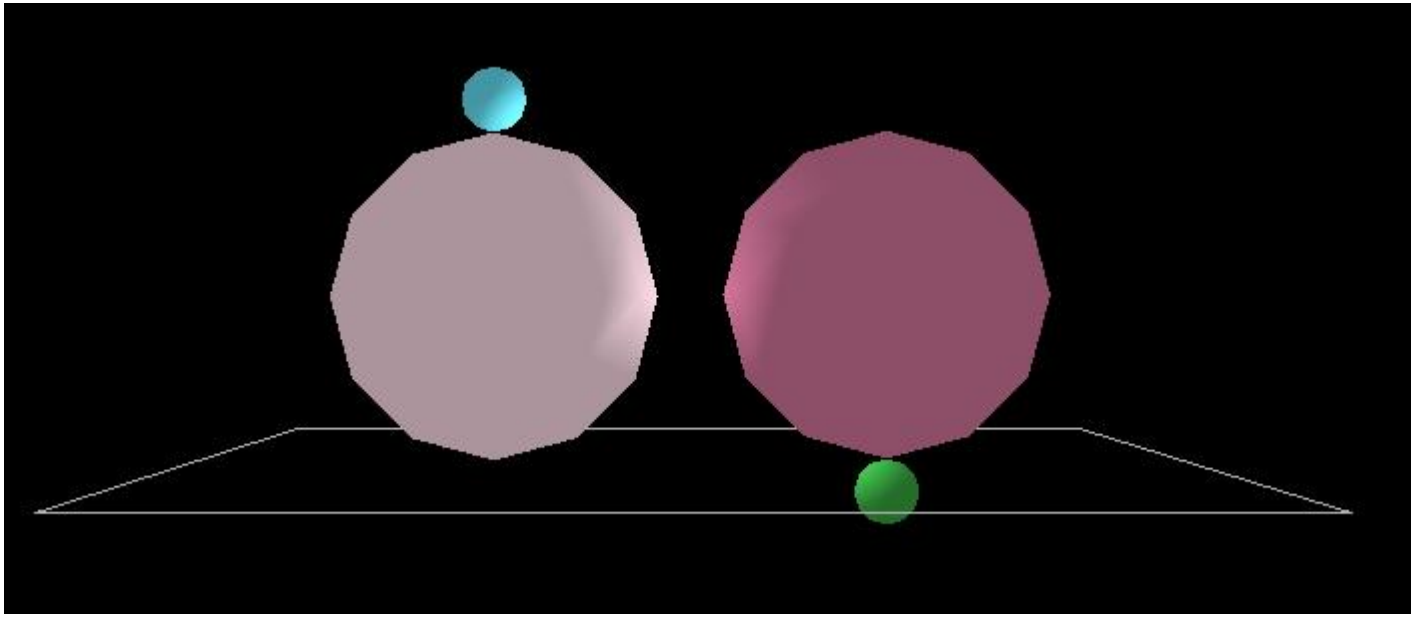
Collision Resolution: Sphere-Halfspace

5. Nice job!



Scene 4

Create a ground with a light particle resting on a heavy one, and a heavy particle resting on a light one. The latter situation poses well-known problems to real-time engines.



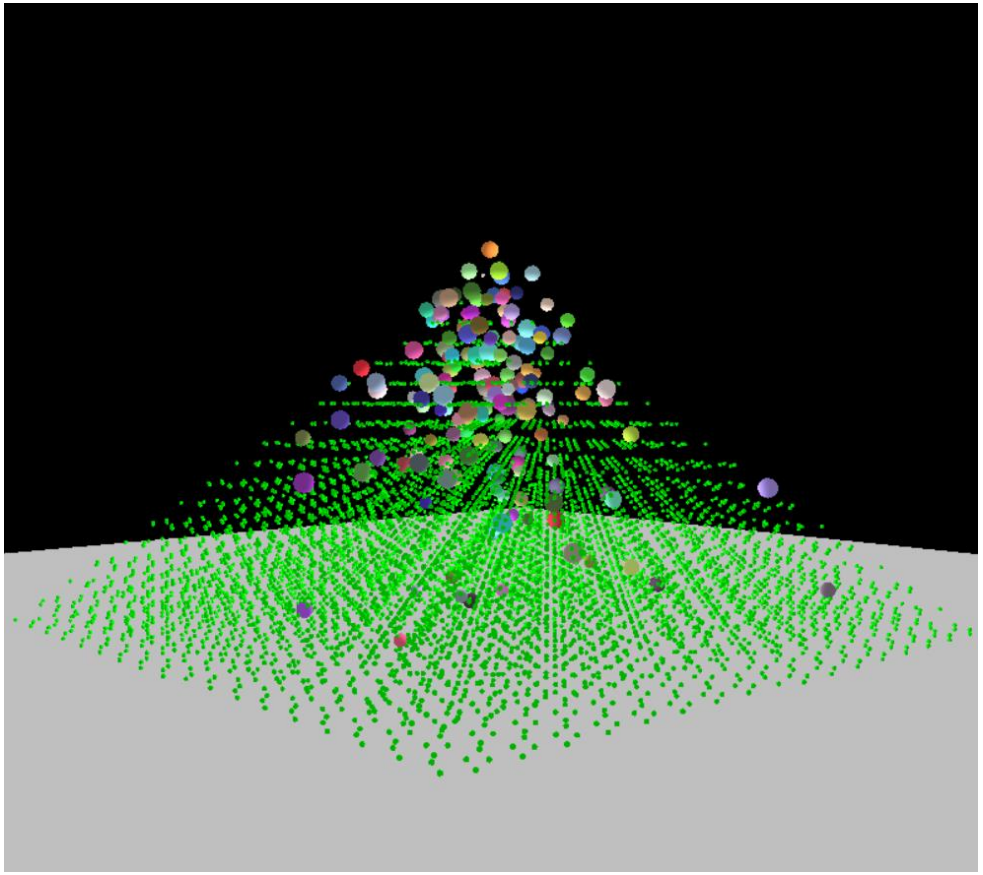
Broadphase optimization I
Static bodies

Physics engines store static bodies in a separate list.

- Static bodies are fixed and never move
- It is unnecesseray to check collisions between them

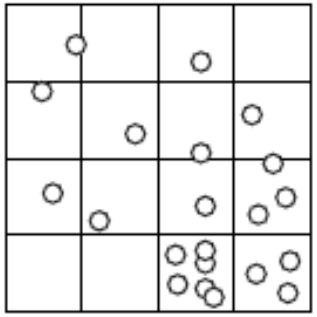
Scene 5

Create a 3D “Plinko” pyramid composed of many static particles, and some dynamic falling particles. Show that working with a separate list of static bodies significantly accelerates the simulation.

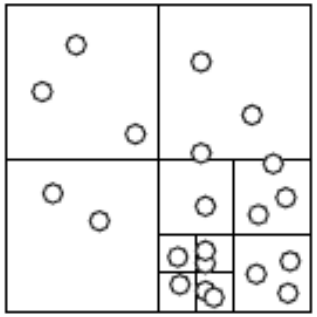


Broadphase optimization II

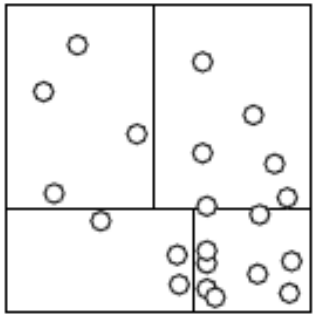
Space partitioning



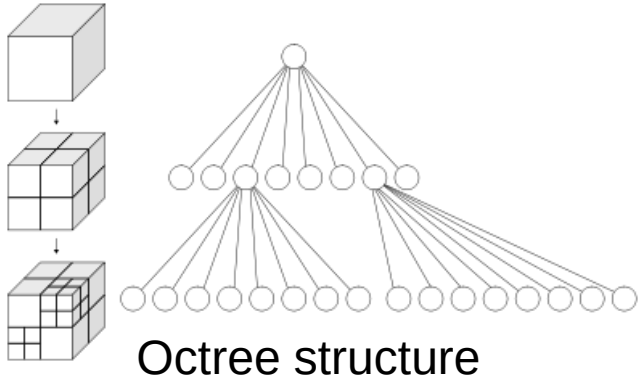
Grid



Octree



kd-tree



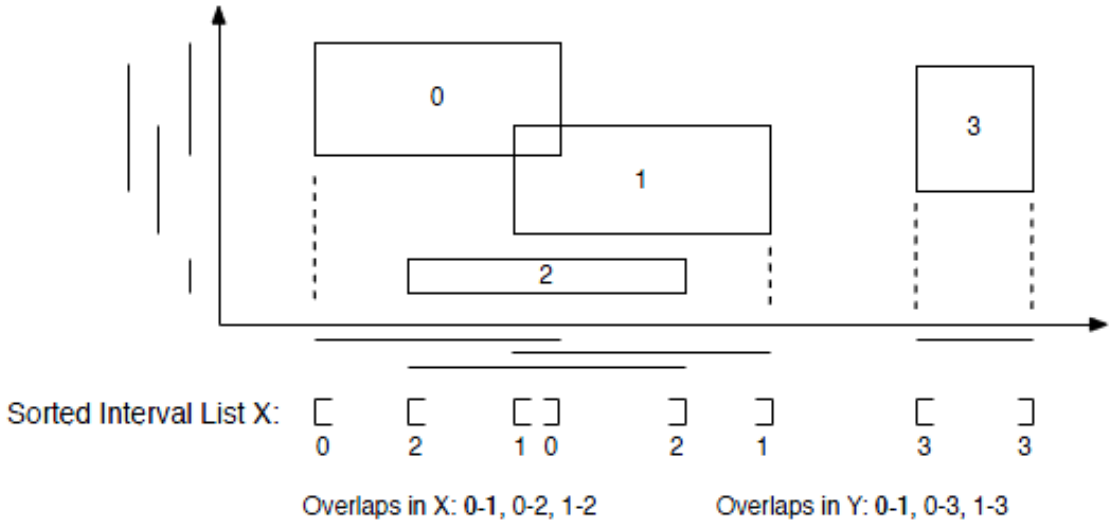
Octree structure

1. Divide environment into multiple spatial domains
2. Map objects into intersecting domains (mapping phase)
3. Perform checks for each spatial domain (intersection phase)

Subdivision techniques are most effective if the cell size is equal to the size of the bounding volumes.

Broadphase optimization III

Sweep & Prune

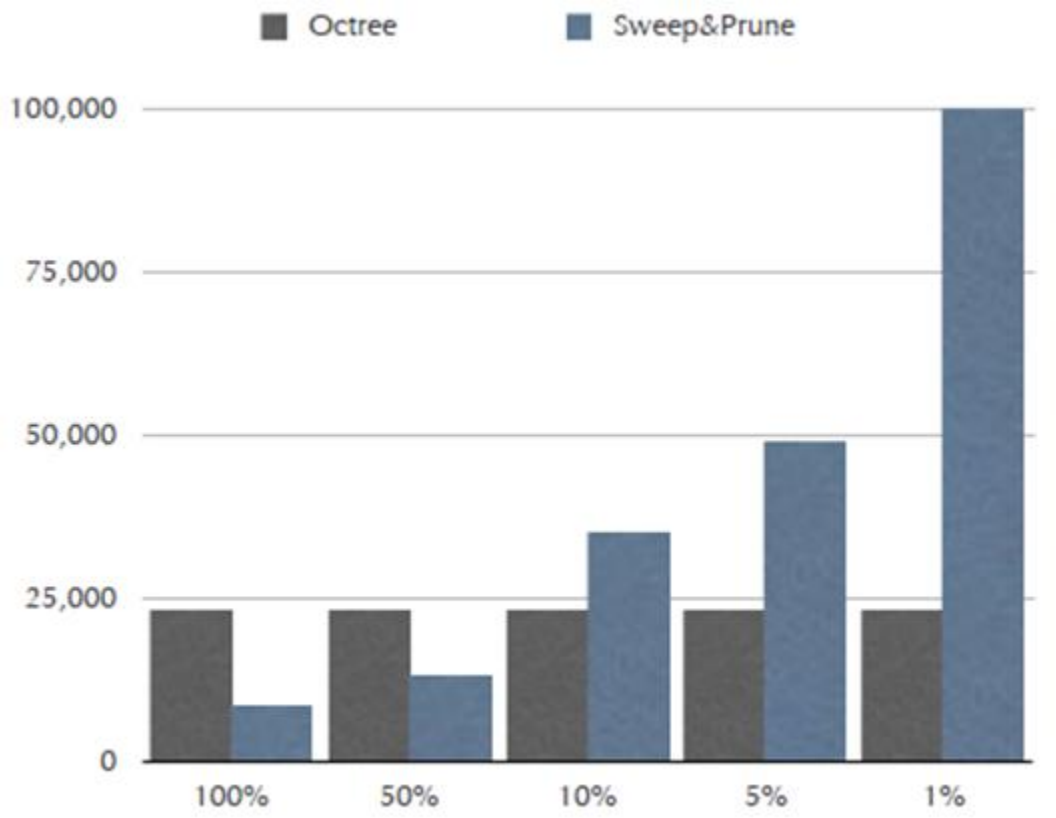


- AABB extents in each dimension are sorted into lists
- Order of minima and maxima determine overlap per dimension
- AABBs overlap if the pair overlaps in all dimensions
- Candidate Set holds set of overlapping AABB pairs
- Exploit frame-to-frame coherency: the lists are not expected to change much

Performance: ~O(n)

Space Partitioning vs. Sweep & Prune

Sweep & Prune is efficient if few objects are moving.



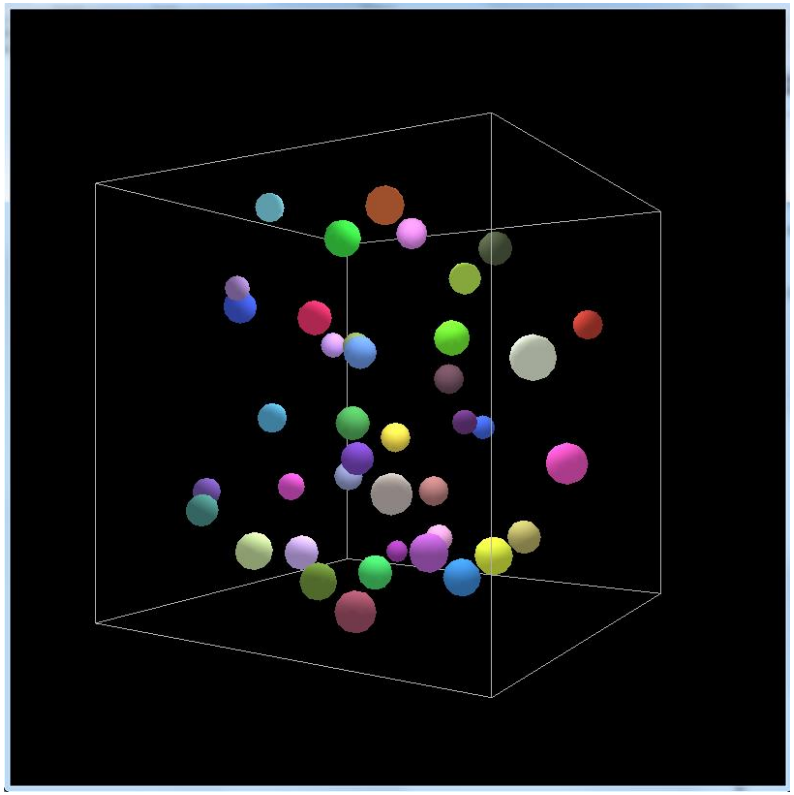
Nb of objects simulated in 1/60s depending of the nb of objects moving.

Scene 6

Create a scene with particle-particle and particle-halfspace collisions. Particles bounce in a confined space modeled by six halfspaces.

The user can rotate the « box » with the mouse.

[BONUS] Implement a broadphase sweep & prune or space partitioning algorithm.



Your work

Implement the scenes 1 – 6. In each scene, the user can toggle between

- Gravity on/off
- Framerate independence on/off
- Explicit Euler, Semi-explicit Euler and RK4 integrators.
- Collisions on/off
- Static bodies on/off
- [BONUS] Sweep & prune or space partitioning on/off