Collision detection

Jiří Chmelík, Marek Trtík

PA199

Outline

- Broad phase
  - Sweep and prune algorithm
- Narrow phase
  - Gilbert-Johnson-Keerthi (GJK) algorithm
- Caching collisions
- Computing collision time

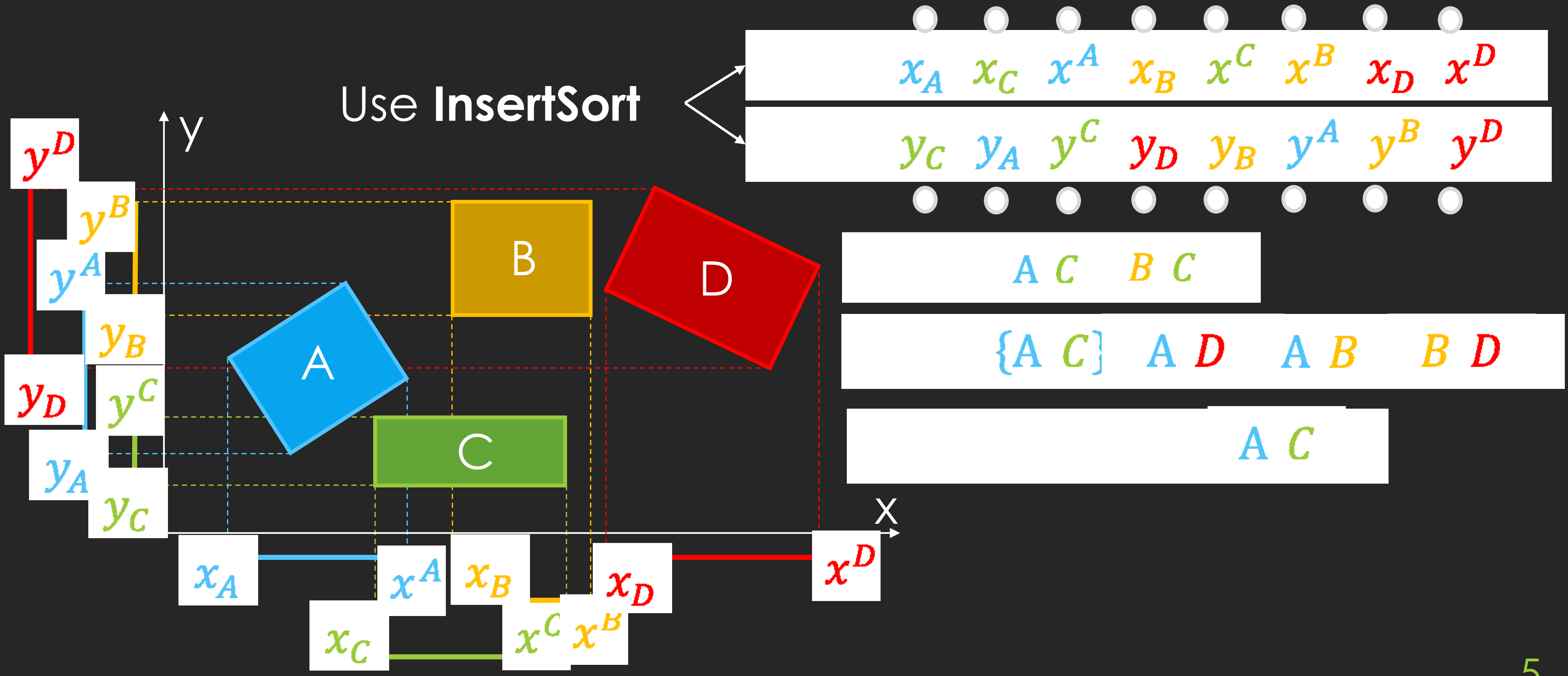
Broad phase

Broad phase

- The goal is to quickly find **pairs** of **potentially colliding** rigid bodies.
  - Used algorithm defines meaning of “potentially colliding”. Examples:
    - When AABBs of the bodies are colliding.
    - When both bodies are in the same area of space.
- We can use space partitioning data structures we already know:
  - Octree, k-D tree, BSP
- Rigid bodies change their positions and orientations during simulation.

  => The data structure must be periodically updated.
  - Utilize time coherence of frames (positions of bodies do not change much between adjacent frames) to get an efficient update algorithm.

Sweep and prune algorithm



Sweep and prune algorithm

- The presented version is easy to understand and implement.
- But it wastes time by recomputing W from scratch each timestep.
- In practice, we use an improved version:
  - We start with the arrays L_α, $\alpha \in \{x, y, z\}$, and W from the previous frame.
  - We incrementally update each L_α and W for each relocated object A.

foreach axis $\alpha \in \{x, y, z\}$ **do**
    Update a_X in L_α by the new lower bound of A along the axis α.
    while $\exists\, a_Y^X$ right before a_X in L_α s.t. $a_Y^X > a_X$ **do**
        Swap a_X with a_Y^X in L_α.
        if X **is None then** insert (A, B) to W.

Sweep and prune algorithm

- Update a^A in L_x by the new upper bound of A along the axis x.

  while $\exists a_x^Y$ right after a^A in L_x s.t. $a^A > a_x^Y$ **do**
  Swap a^A with a_x^Y in L_x.
  if Y **is None then** Insert (A, B) to W.

  Moving a^A to the right

  while $\exists a_x^X$ right after a_A in L_x s.t. $a_A > a_x^X$ **do**
  Swap a_A with a_x^X in L_x.
  if X **is None then** Erase (A, B) from W.

  Moving a_A to the right

  while $\exists a_x^Y$ right before a^A in L_x s.t. $a_x^Y > a^A$ **do**
  Swap a^A with a_x^Y in L_x.
  if Y **is None then** Erase (A, B) from W.

  Moving a^A to the left

Sweep and prune algorithm

- Possible memory representation of the lists L_α, $\alpha \in \{x, y, z\}$:

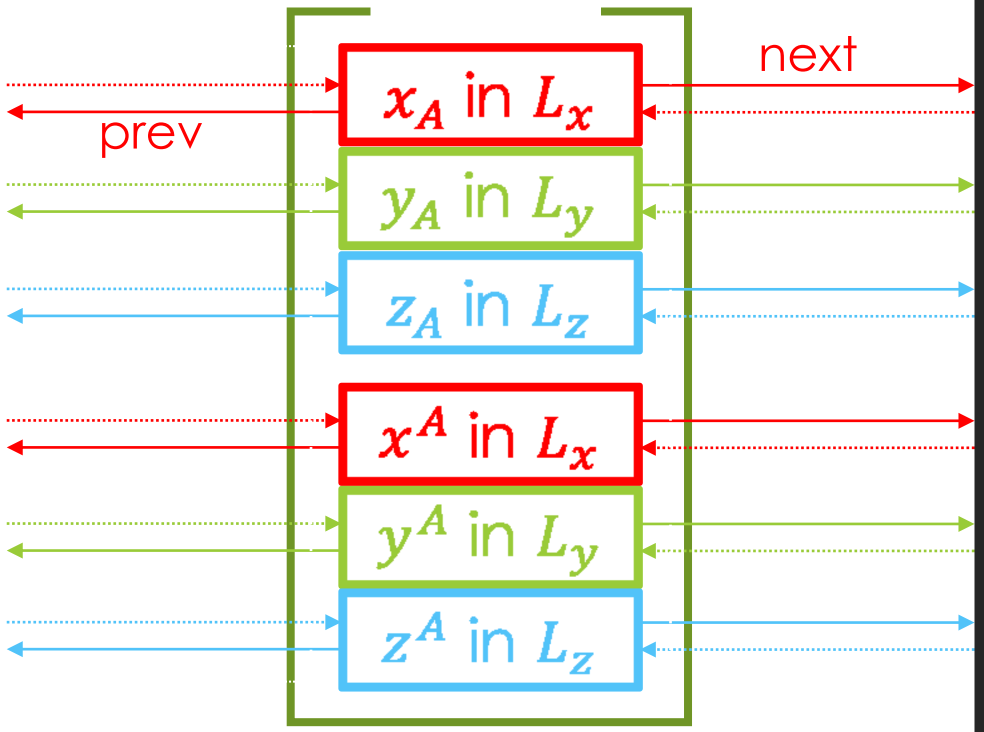
```
struct Link {
    Link *next, *prev;
    float coord;
    char lohi : 1;
};
```

"red"

If lohi == 0, then "coord" is α_A and α^A otherwise.

Represents either α_A or α^A.

```
using AABB = Link[2][3];
```



Sweep and prune algorithm

- If "p" is a pointer to a "Link" of the list $L_\alpha, \alpha \in \{0,1,2\}$ (i.e., $\{x,y,z\}$), then we can convert it to a pointer to AABB using the pointer arithmetic:

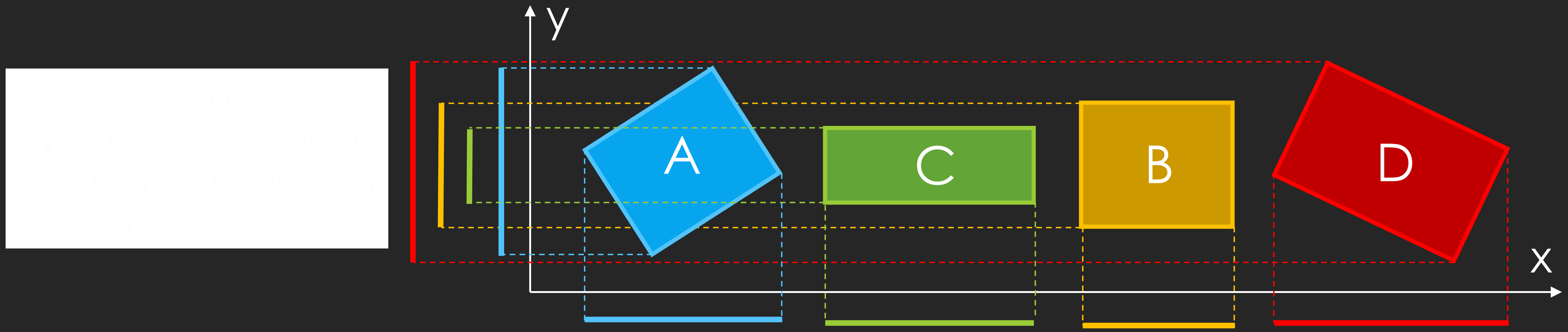
  (AABB*)(p − (α + 3 * (int)p.lohi) * sizeof(Link))

- Represent the set W as a dictionary of pairs of object IDs.
  - Sort the pair s.t. the lower ID comes first and the other the second.

- Initialize the data structure to contain a single auxiliary AABB s.t.:
  - Values in the links are: $x_A = y_A = z_A = -\infty$ and $x^A = y^A = z^A = +\infty$.
  - All 2*3 links are properly interconnected in the lists L_x, L_y, L_z.
  - This auxiliary AABB avoids the "nullptr" checks in the algorithms.

Sweep and prune algorithm

- Performance of the algorithm is sensitive to alignment of objects along coordinate axes:

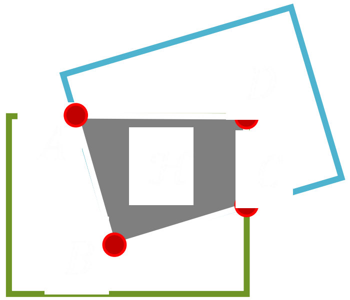


- A relocation of an object leads to a lot of swaps thought the “cluster” in the array.

Narrow phase

Narrow phase

- The goal is for each pair of potentially colliding shapes to:
  - Decide whether the shapes really collide or not.
  - Compute a finite model of the (infinite) set of all collision points.
- Example: Find finite and minimal number of points in $\mathcal{H}$ whose convex hull contains $\mathcal{H}$.



- Requirement: The effect of collision forces computed at points of the model must be equal to collision forces computed at all points in $\mathcal{H}$.

Gilbert-Johnson-Keerthi (GJK) algorithm

- Decides whether two **convex** shapes have **empty intersection** or not.

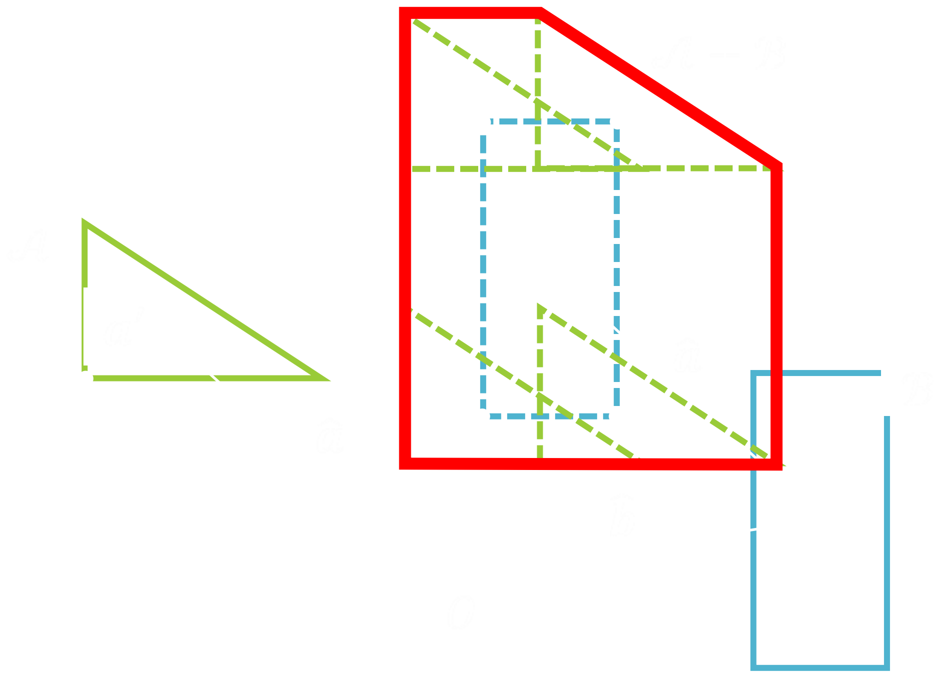
  - We can approximate a concave shape by a **set** of convex shapes.
  - For the empty intersection we can obtain a pair of the **closest points**.

- We must first build a terminology:
  - Minkowski sum and difference
  - Simplex
  - Support function

GJK: Minkowski sum

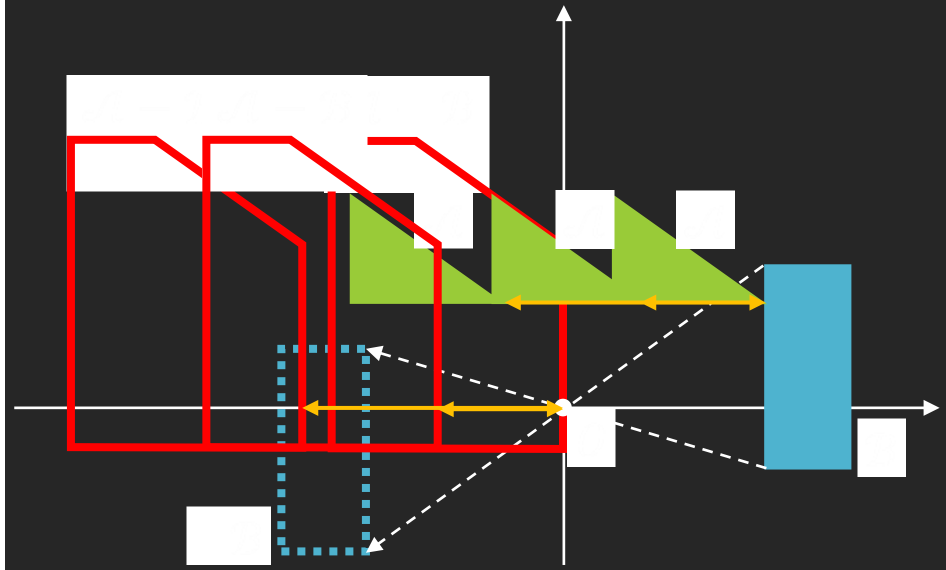
- Minkowski sum: $\mathcal{A} + \mathcal{B} = \{a + b \mid a \in \mathcal{A} \land b \in \mathcal{B}\}$.
- How to draw Minkowski sum?
  - Choose some points $\hat{a} \in \mathcal{A} \land \hat{b} \in \mathcal{B}$.
  - Then, $\forall a \in \mathcal{A}\ \exists a'$ s.t. $a = \hat{a} + a'$.
  - Therefore, for each $a \in \mathcal{A} \land b \in \mathcal{B}$

    $$a + b = \hat{a} + \hat{b} + a' + b'$$
  - So, we draw $\mathcal{A} + \mathcal{B}$ around $\hat{a} + \hat{b}$.
    - Draw $\mathcal{B}$ around $\hat{a} + \hat{b}$.
    - Draw $\mathcal{A}$ around $\mathcal{B}$'s perimeter.
    - $\mathcal{A} + \mathcal{B}$ is the convex hull.



GJK: Minkowski difference

- Minkowski difference: $\mathcal{A} - \mathcal{B} = \mathcal{A} + (-\mathcal{B})$, where $-\mathcal{B} = \{-b; b \in \mathcal{B}\}$
- Lemma: The shortest distance between $\mathcal{A}$ and $\mathcal{B}$ is equal to the distance of $\mathcal{A} - \mathcal{B}$ to the origin.
  Proof: It is a length of the shortest $\hat{a} - \hat{b}$, s.t. $\hat{a} \in \mathcal{A} \wedge \hat{b} \in \mathcal{B}$. But $\hat{a} - \hat{b} \in \mathcal{A} - \mathcal{B}$.
- Consequence: Shapes $\mathcal{A}$ and $\mathcal{B}$ collide if and only if $\mathcal{A} - \mathcal{B}$ contains the origin.



GJK: Minkowski difference

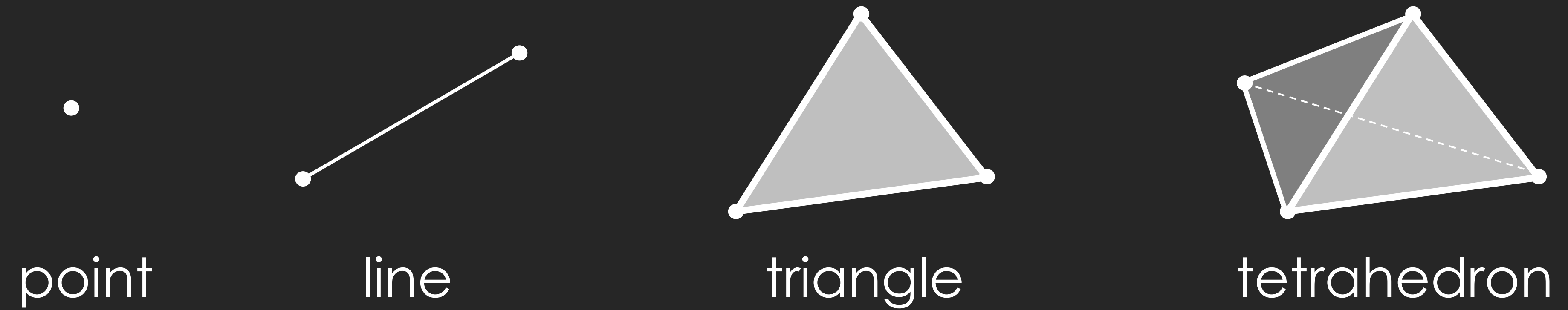
- Lemma: If shapes $\mathcal{A}$ and $\mathcal{B}$ are convex, then $\mathcal{A}-\mathcal{B}$ is also convex.
  Proof: For each $u, v \in \mathcal{A}-\mathcal{B}$ there exist $a_u, a_v \in \mathcal{A}$ and $b_u, b_v \in \mathcal{B}$ s.t. $u = a_u - b_u$ and $v = a_v - b_v$. Then, for $t \in (0,1)$, we get

$$u + t(v - u) = (a_u - b_u) + t\big((a_v - b_v) - (a_u - b_u)\big) =$$
$$a_u - b_u + ta_v - tb_v - ta_u + tb_u =$$
$$a_u + t(a_v - a_u) - \big(b_u + t(b_v - b_u)\big).$$

  $\mathcal{A}$ and $\mathcal{B}$ are convex => $a_u + t(a_v - a_u) \in \mathcal{A}$, $b_u + t(b_v - b_u) \in \mathcal{B}$ => $a_u + t(a_v - a_u) - \big(b_u + t(b_v - b_u)\big) \in \mathcal{A}-\mathcal{B}$ => $\mathcal{A}-\mathcal{B}$ is convex.
- Consequence: If the origin lies in the convex hull of points $x_1, \dots, x_n \in \mathcal{A}-\mathcal{B}$, then convex shapes $\mathcal{A}$ and $\mathcal{B}$ have non-empty intersection.

GJK: Simplex

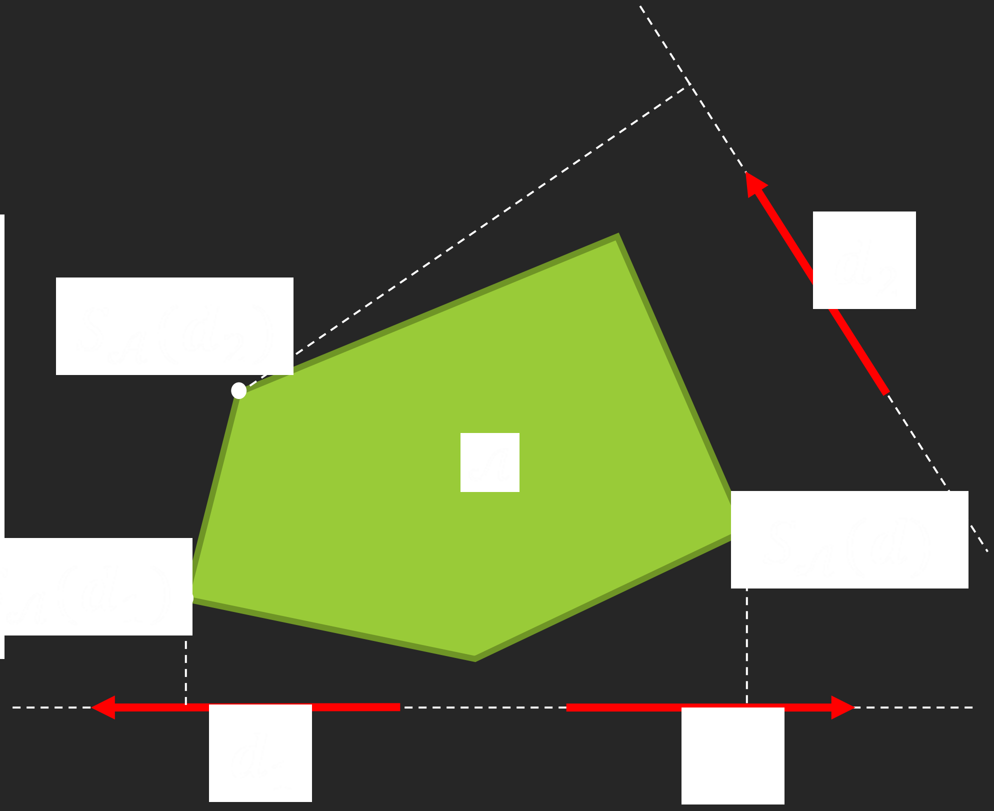
- A **simplex** is a convex hull of an affinely independent points.



- GJK searches for a simplex s.t. origin lies inside or prove that no such simplex exists.
- Note: In 2D case we only need point, line and triangle.

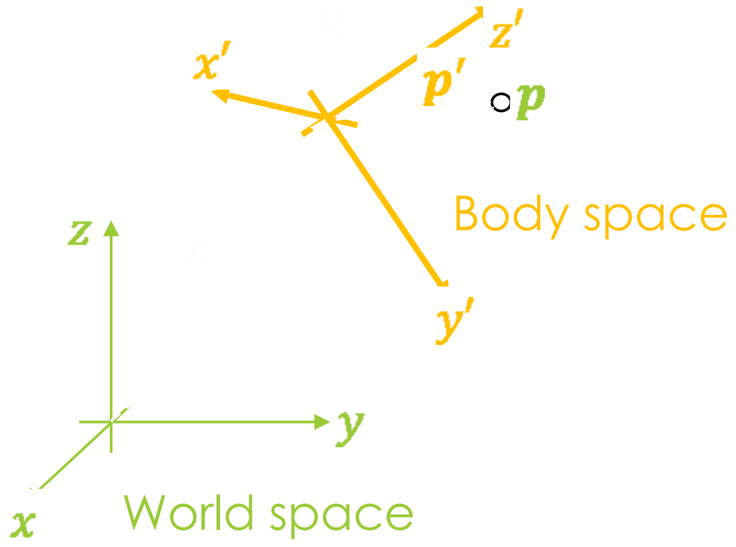
GJK: Support function

- Given a shape $\mathcal{A}$ and a non-zero vector d, a support function $S_\mathcal{A}$ returns a point $S_\mathcal{A}(d) \in \mathcal{A}$ s.t. $S_\mathcal{A}(d) \cdot d = \max\{x \cdot d; x \in \mathcal{A}\}$.



GJK: Support function

- A shape $\mathcal{A}$ can be defined in a local system – Body/model space.



$$\mathbf{p} = R\,\mathbf{p}' + \mathbf{x}$$

$$\mathbf{p}' = R^{-1}(\mathbf{p} - \mathbf{x})$$

- Therefore, this must be reflected in the computation of $S_{\mathcal{A}}(\mathbf{d})$.

GJK: Support function

- When a convex shape $\mathcal{A}$ is defined in body space (R, x), then we denote $R\mathcal{A} + x$ the corresponding convex shape in the world space.
  - More precisely: $R\mathcal{A} + x = \{Rp' + x;\ p' \in \mathcal{A}\}$.
- Lemma: $S_{R\mathcal{A}+x}(d) = R S_{\mathcal{A}}(R^\top d) + x$, for each world-space vector $d \neq 0$.

Proof: First, we show that $\forall p' \in \mathcal{A}$ the following equality (*) holds true

$$\begin{aligned}(Rp' + x) \cdot d &= (Rp') \cdot d + x \cdot d \\ &= d^\top (Rp') + x \cdot d \\ &= (d^\top R) p' + x \cdot d \\ &= (R^\top d)^\top p' + x \cdot d \\ &= p' \cdot (R^\top d) + x \cdot d.\end{aligned}$$

GJK: Support function

- Now, $S_{R\mathcal{A}+x}(d)\cdot d = \max\{p\cdot d; p\in R\mathcal{A}+x\}$

$$= \max\{(Rp'+x)\cdot d; p'\in\mathcal{A}\}$$

$$= \max\{p'\cdot(R^{\top}d)+x\cdot d; p'\in\mathcal{A}\} \quad \text{according to (*)}$$

$$= \max\{p'\cdot(R^{\top}d); p'\in\mathcal{A}\}+x\cdot d$$

$$= S_{\mathcal{A}}(R^{\top}d)\cdot R^{\top}d+x\cdot d$$

$$= (RS_{\mathcal{A}}(R^{\top}d)+x)\cdot d \quad \text{according to (*)}$$

Therefore, $S_{R\mathcal{A}+x}(d) = RS_{\mathcal{A}}(R^{\top}d)+x$.

GJK: Support function examples

- $\mathcal{A}$ is a sphere at the origin with the radius r:

$$S_{\mathcal{A}}(\vec{d}) = r\frac{\vec{d}}{|\vec{d}|}$$

- $\mathcal{A}$ is an axis aligned bounding box (AABB) at the origin with sizes $2s_x, 2s_y, 2s_z$ along corresponding coordinate axes:

$$S_{\mathcal{A}}(\vec{d}) = \left(\operatorname{sgn}(d_x)\, s_x, \operatorname{sgn}(d_y)\, s_y, \operatorname{sgn}(d_z)\, s_z\right)^T$$

$$\text{where } \operatorname{sgn}(\alpha) = \begin{cases} -1 & \text{if } \alpha < 0 \\ 1 & \text{otherwise} \end{cases}$$

GJK: Support function examples

- $\mathcal{A}$ is a cylinder at the origin with the central axis aligned with the z coordinate axis, with the radius r and with the top and bottom base at z-coordinate h and $-h$, respectively:

$$S_{\mathcal{A}}(\vec{d}) = \begin{cases} \left(\frac{r}{\sigma} d_x, \frac{r}{\sigma} d_y, \operatorname{sgn}(d_z)\, h\right)^T & \text{if } \sigma > 0 \\ (0, 0, \operatorname{sgn}(d_z)\, h)^T & \text{otherwise} \end{cases}$$

where $\sigma = \sqrt{d_x^2 + d_y^2}$, and $\operatorname{sgn}(x)$ was defined earlier.

- $\mathcal{A}$ is any convex polytope (e.g., point, line, triangle, convex polygon, tetrahedron, box, …) with vertices $V = \{v_1, \ldots, v_n\}$:

$$S_{\mathcal{A}}(\vec{d}) = v_i \quad \text{s.t.} \quad v_i \cdot \vec{d} = \max\{v_j \cdot \vec{d} : v_j \in V\}$$

GJK: Support function

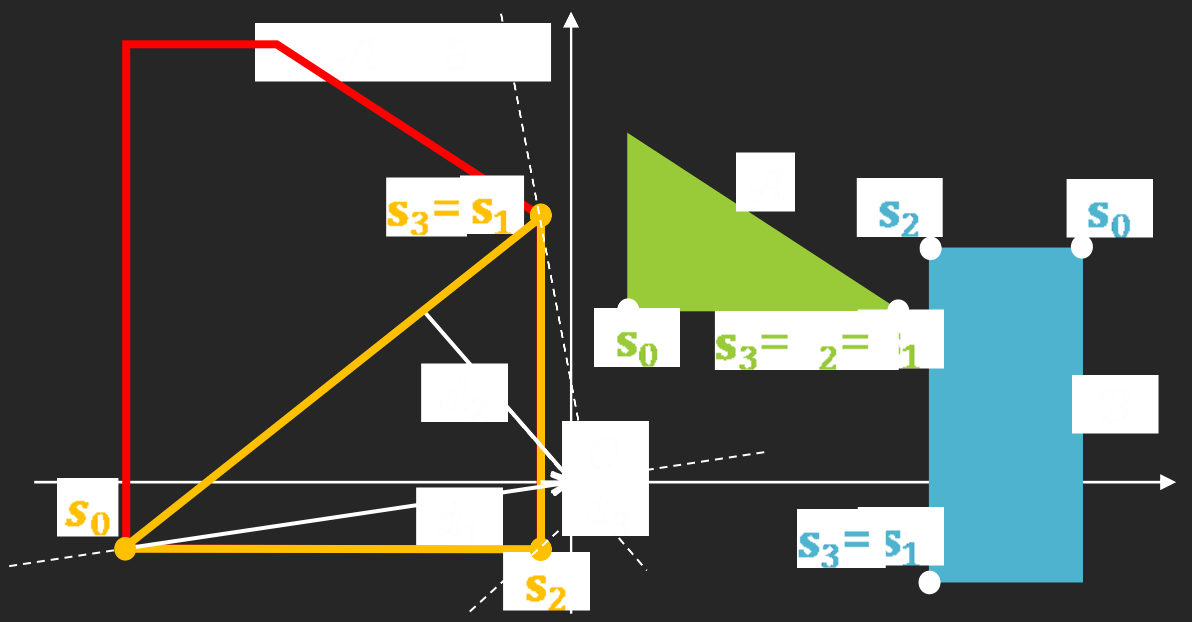
-
- Lemma: $S_{A-B}(\vec{d}) = S_A(\vec{d}) - S_B(-\vec{d})$

  Proof: $S_{A-B}(\vec{d}) \cdot \vec{d} = \max\{(a-b) \cdot \vec{d} : a \in A, b \in B\}$

  $$= \max\{a \cdot \vec{d} : a \in A\} - \min\{b \cdot \vec{d} : b \in B\}$$
  $$= S_A(\vec{d}) \cdot \vec{d} + \max\{-b \cdot \vec{d} : b \in B\}$$
  $$= S_A(\vec{d}) \cdot \vec{d} + S_B(-\vec{d}) \cdot (-\vec{d})$$
  $$= (S_A(\vec{d}) - S_B(-\vec{d})) \cdot \vec{d}$$

- We therefore do not have to construct $A - B$ and S_{A-B}. We work with the given shapes A and B and their support functions.

GJK: The algorithm – intuition (2D case)



s_0 s_0

$S = \{ s_0\ s_1, s_2$

s_0 s_0 s_0

s_1 s_1 s_1

s_1

s_2 s_2 s_2

s_2

s_3 s_3 s_3

s_3 $<$ NO INTERSECTION!

GJK: The algorithm – intuition (2D case)

- Since A and B have empty intersection, we can compute a pair of closest points:
  - First, we find the closest point X of the simplex $S = \{\mathbf{s}_1, \mathbf{s}_2\}$ to the origin. That is $X = \mathbf{s}_1 + t(\mathbf{s}_2 - \mathbf{s}_1)$ for some $t \in (0,1)$ and $\|X - 0\|^2 = \|\mathbf{s}_1 + t(\mathbf{s}_2 - \mathbf{s}_1)\|^2$ is minimal. So, solve the equation:

$$\frac{d}{dt}\|\mathbf{s}_1 + t(\mathbf{s}_2 - \mathbf{s}_1)\|^2 = 0$$

$$\frac{d}{dt}\left(\mathbf{s}_1 \cdot \mathbf{s}_1 + 2t\,\mathbf{s}_1 \cdot (\mathbf{s}_2 - \mathbf{s}_1) + t^2 (\mathbf{s}_2 - \mathbf{s}_1)^2\right) = 0$$

$$2\,\mathbf{s}_1 \cdot (\mathbf{s}_2 - \mathbf{s}_1) + 2t(\mathbf{s}_2 - \mathbf{s}_1)^2 = 0$$

$$t = -\frac{\mathbf{s}_1 \cdot (\mathbf{s}_2 - \mathbf{s}_1)}{(\mathbf{s}_2 - \mathbf{s}_1)^2}$$

Also clamp to (0,1)

GJK: The algorithm – intuition (2D case)

- Then, find the corresponding points in the shapes $\mathcal{A}$ and $\mathcal{B}$

$$s_1 + t(s_2 - s_1) = (s_1 - s_1) + t((s_2 - s_2) - (s_1 - s_1))$$
$$= \underbrace{(s_1 + t(s_2 - s_1))}_{\in \mathcal{A}} - \underbrace{(s_1 + t(s_2 - s_1))}$$

GJK: The algorithm

- Choose some $p \in \mathcal{A} - \mathcal{B}$ // Usually, $\boldsymbol{p}$ comes from the previous frame.

$S = \emptyset$ // We start with the empty simplex.

$s = S_{\mathcal{A}-\mathcal{B}}(-p)$ // NOTE: Our direction vector $\boldsymbol{d}$ to the origin is just $-\boldsymbol{p}$.

while $|p|^2 - p \cdot s > \varepsilon^2$ **do** // Proving termination condition: see [4].

  // $\boldsymbol{p}$ is still far from the closest point of $\mathcal{A} - \mathcal{B}$ to the origin.

  $p = \boxed{\text{closest\_to\_origin}}\boxed{(\text{convex\_hull}(S \cup \{s\}))}$

Point, line, triangle, or tetrahedron.

Can be computed quickly for shapes

  $S = \text{smallest } X \subseteq S \cup \{s\} \text{ s.t. } p \in \text{convex\_hull}(X)$ // Reduce the simplex.

  $s = S_{\mathcal{A}-\mathcal{B}}(-p)$

return $\boxed{\{a \in \mathcal{A}, b \in \mathcal{B} \mid p = a - b\}}$ // $|\boldsymbol{p}|$ is the closest distance.

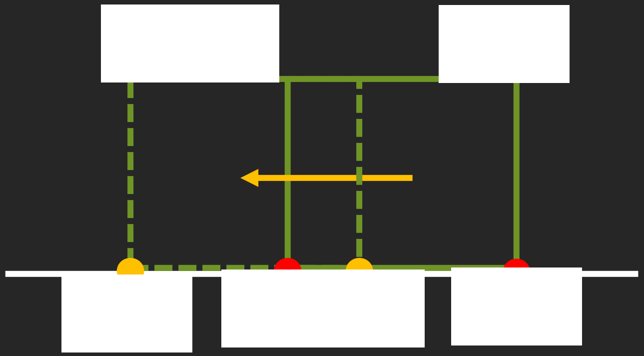
Caching collisions

Caching collisions

- Efficiency of the PGS algorithm for a constraint system depends on the initial value λ^0.
- It's likely that λ computed for a collision constraint at current frame would be "almost valid" for the next frame (if the collision persists).
- Therefore, caching λ values for collision (and other types of) constraints amongst frames can bring considerable speed boost.
- How to match collisions computed in different frames?

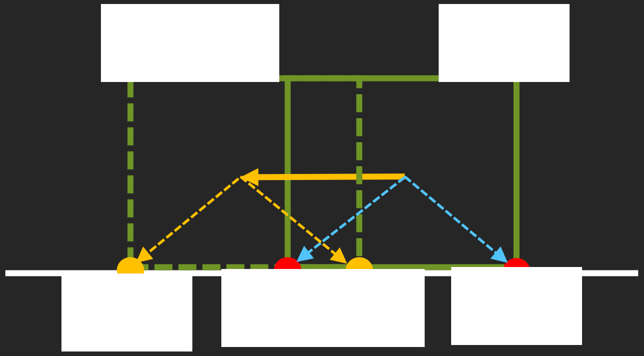
Caching collisions

- There are several possibilities:
  - **Distance** between collision points in **world space**:

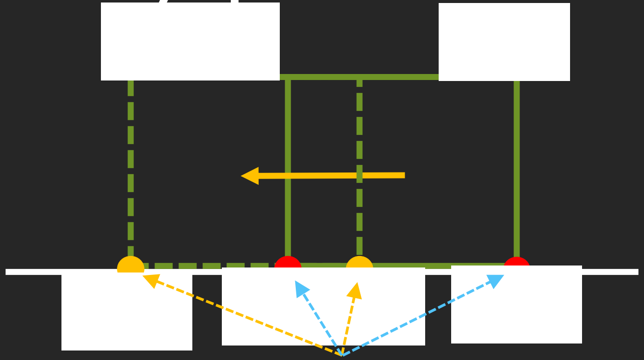


Imprecise.

  - **Distance** between collision points in **body space**:



Precise in space of the box



Imprecise in space of **ground**

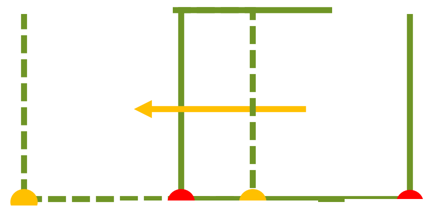
Caching collisions

- Identify collisions by geometrical properties of collision shapes:

```
enum GTYPE { VERTEX, EDGE, FACE };
struct CollisionID {
    int body_index_1;             // The index of R_i: i
    GTYPE feature_type_1;         // The type of colliding geometry in R_i
    int feature_index_1;          // Index of the colliding geometry in R_i
    int body_index_2;             // The index of R_j: j
    GTYPE feature_type_2;         // The type of colliding geometry in R_i
    int feature_index_2;          // Index of the colliding geometry in R_i
};
```

Define also comparison and hashing of CollisionID instances

Caching collisions

- 

We get precise mapping:

Recommended approach

- The cache should be a map from CollisionID instance to values Δt:

```
using collision_cache = std::unordered_map<CollisionID, float>;
```

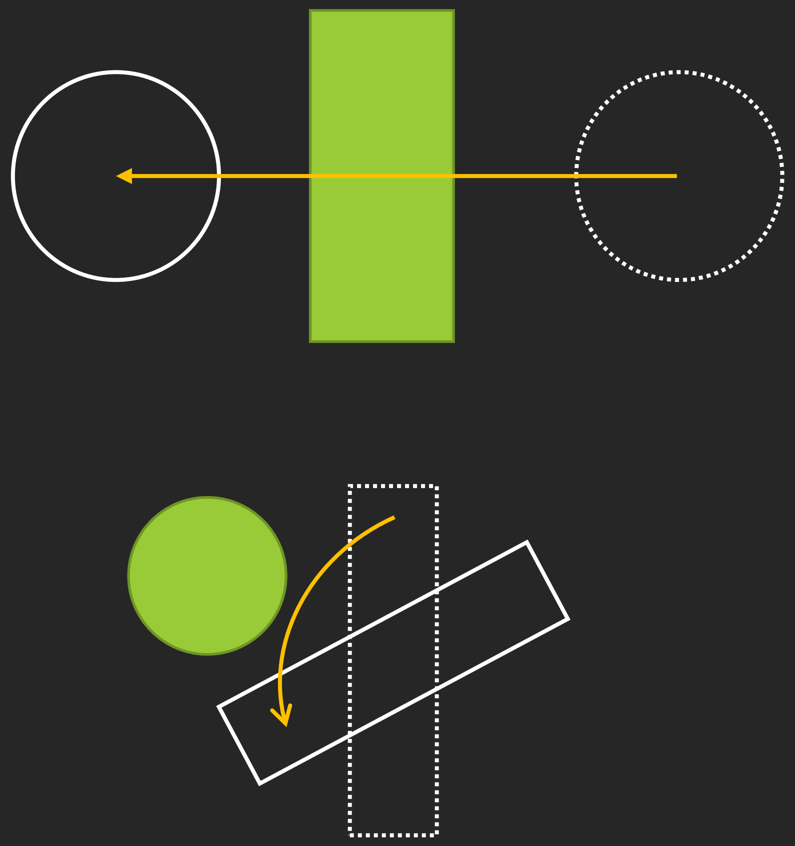
- And how to use the cache?

Caching collisions

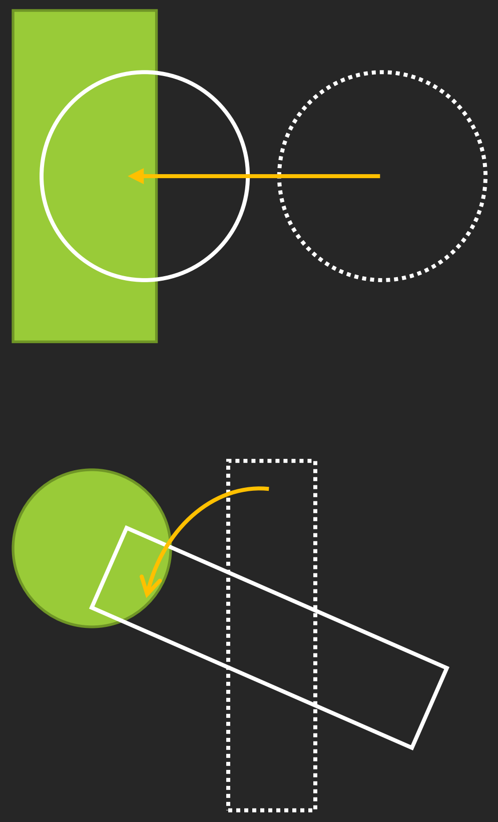
- Before solving the constraint system initialize λ^0 s.t.:
  - For each computed collision c and the corresponding element λ_i^0:
    - Build the CollisionID instance id from c.
    - If id is present in the cache, then set λ_i^0 to the value λ in the cache.
    - Otherwise, set λ_i^0 to 0.
- Once new solution λ is computed updated the cache as follows:
  - Clear the cache.
  - For each collision c and the corresponding computed value λ_i:
    - Build the CollisionID instance id from c.
    - Insert the mapping $id \to \lambda_i$ to the cache.

Computing collision time

Tunnelling and penetration



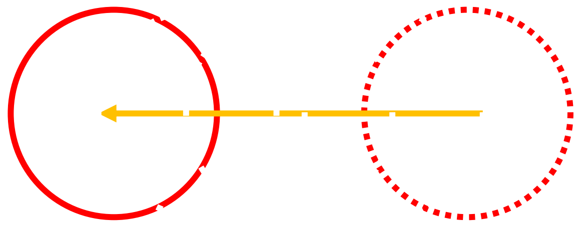
Tunnelling

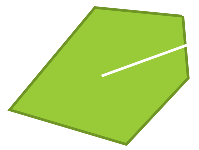


Penetration

Dealing with tunnelling and penetration

- The simplest approach is to subdivide the game time step dt into several small "internal" time steps.
- For broad phase:
  - Approximate collision shapes of bodies by "moving spheres":

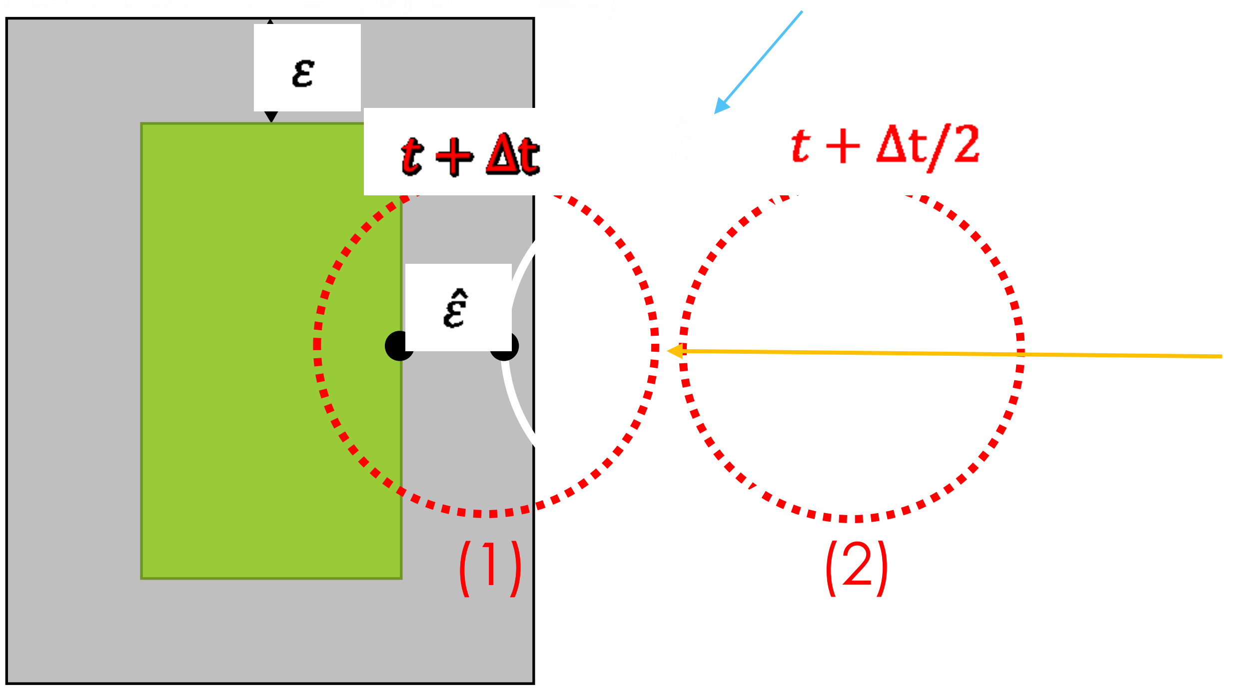




- Use the adaptive time step:
  - For each pair of potentially colliding shapes compute the nearest collision time.
  - Move the bodies only to the minimum of all nearest collision times.

Computing collision time

- Binary search for $\hat{t} \in (t, t + \Delta t)$



- There are 4D collision algorithms – they consider translations and rotations of tested objects.

References

[1] *Erin Catto;* Iterative Dynamics with Temporal Coherence; Crystal Dynamics, Menlo Park, California, 2005
[2] E. G. Gilbert, D. W. Johnson and S. S. Keerthi; A fast procedure for computing the distance between complex objects in three-dimensional space; Journal on Robotics and Automation, vol. 4, no. 2, pp. 193-203, April 1988
[3] *G. Bergen*; A Fast and Robust GJK Implementation for Collision Detection of Convex Objects; Eindhoven University of Technology. 1999
[4] G.v.d. Bergen; Collision detection in interactive 3D environments; ISBN: 1-55860-801-X, Elsevier, 2004.