Physics based animation

Lecture 08 - Collision detection - Part 1

Grégory Leplâtre

g.leplatre@napier.ac.uk, room D32
School of Computing
Edinburgh Napier University

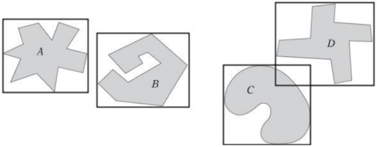
Semester 1 - 2016/2017

Plan

- Introduction - Design issues
- Mathematical tools

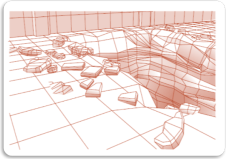
Outline

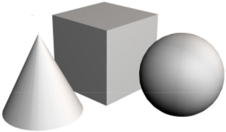
Collisions



- Provides the illusion of a physically plausible, solid world
- Can be computationally challenging:
  - Testing collisions between each pair: $(n(n-1)/2)$ tests
  - Working with complex geometry

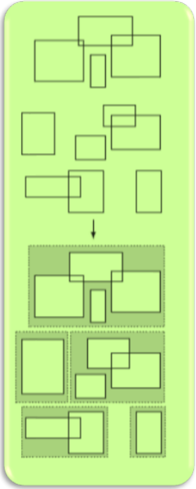
Object representation





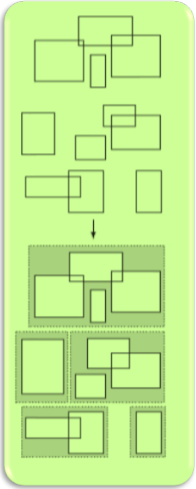
- **Explicit representation**: geometry is described by faces, edges and vertices. If not made of triangles, it can always be triangulated (it is at render time)
  - Typically too complex to be worked with directly
  - Unnecessary level of details for collision detection
- **Implicit representation**:
  - Primitives can be represented by exact mathematical model
  - or using Boolean constructions (Constructive Solid Geometry)

Types of queries



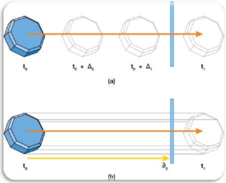
- **Broad phase**: identifying (small) subgroups of objects that may be colliding
- **Narrow phase**: pairwise collision tests to determine whether members of a subgroup are colliding

Sequential vs simultaneous motion



- Real world movements are simultaneous. Not in computer games!
- Two approaches:
  - **Simultaneous motion**: All objects motions are computed first, then collisions are computed.
  - **Sequential motion**: After each object is moved, collisions with that object are computed. This is more approximative, but usually sufficient for games where the frame rate is high enough.

Continuous vs Discrete motion



- Two approaches:
  - **Discrete Motion**: Collisions are computed between objects **as if they were static** i.e., only considering their posision at a given time.
  - **Continuous Motion**: This paradigm takes into account the continuous trajectory of the object during a time step i.e., the **volume covered** by the object. This allows precise collision times to be calculated.
- Discrete motion is a lot **cheaper** but also **less accurate.** Acceptable if the time step is small enough

Performance

- As with most aspects of Games (engine) implementation, performance is paramount.

Performance

- As with most aspects of Games (engine) implementation, performance is paramount.
- Optimisation strategies

Performance

- As with most aspects of Games (engine) implementation, performance is paramount.
- Optimisation strategies
  - **Broad-phase** approach that segregates objects based on spatial distribution

Performance

- As with most aspects of Games (engine) implementation, performance is paramount.
- Optimisation strategies
  - **Broad-phase** approach that segregates objects based on spatial distribution
  - **Bounding volume** suited to the task
    - Loose and cheap first
    - More accurate and expensive when accuracy is required

Performance

- As with most aspects of Games (engine) implementation, performance is paramount.
- Optimisation strategies
  - **Broad-phase** approach that segregates objects based on spatial distribution
  - **Bounding volume** suited to the task
    - Loose and cheap first
    - More accurate and expensive when accuracy is required
  - exploit **temporal coherency**. Most obviously, not all objects move at every frame!

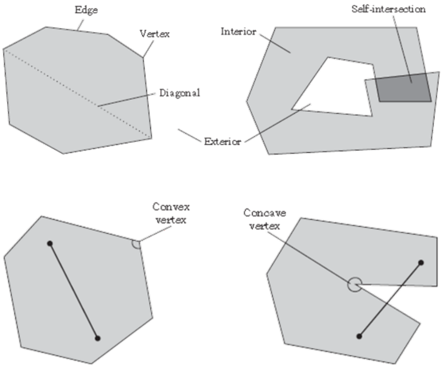
Performance

- As with most aspects of Games (engine) implementation, performance is paramount.
- Optimisation strategies
  - **Broad-phase** approach that segregates objects based on spatial distribution
  - **Bounding volume** suited to the task
    - Loose and cheap first
    - More accurate and expensive when accuracy is required
  - exploit **temporal coherency**. Most obviously, not all objects move at every frame!
  - **Architectural optimisation**, e.g., parallelisation.

Outline

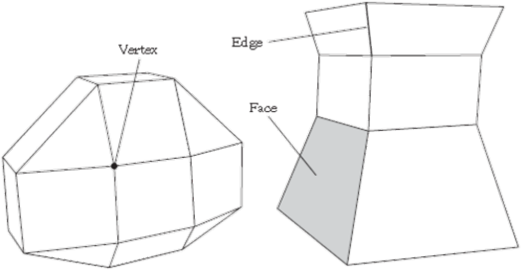
Polygons

Definition of a **polygon** and its **properties**

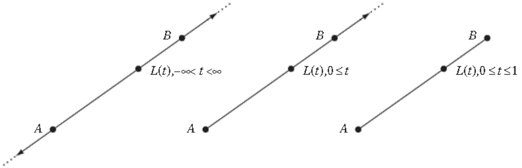


Polyhedra

Definition of a **polyhedron**, **polytope** (bounded convex polyhedron)



Lines, rays, segments

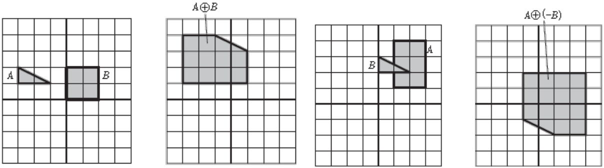


$L(t) = (1 - t)A + tB$

$L(t) = A + t\boldsymbol{v}$, where $\boldsymbol{v} = B - A$

- Line: $L(t)$, $-\infty < t < +\infty$
- Ray: $L(t)$, $L(t)$, $0 < t$
- Segment: $L(t)$, $0 < t < 1$

Minkowski Sum and difference



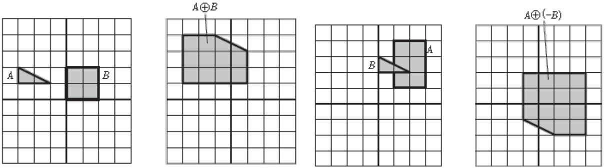
- **Sum:**

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \quad | \mathbf{a} \in A, \mathbf{b} \in B\}$$

- **Difference**

$$A \ominus B = \{\mathbf{a} - \mathbf{b} \quad | \mathbf{a} \in A, \mathbf{b} \in B\}$$

Minkowski Sum and difference



- **Sum**:

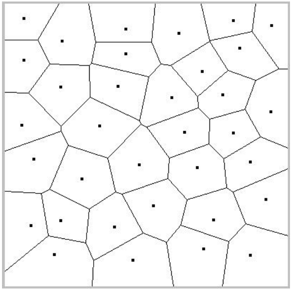
$$A \oplus B = \{\boldsymbol{a} + \boldsymbol{b} \quad | \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$$

- **Difference**

$$A \ominus B = \{\boldsymbol{a} - \boldsymbol{b} \quad | \boldsymbol{a} \in A, \boldsymbol{b} \in B\}$$

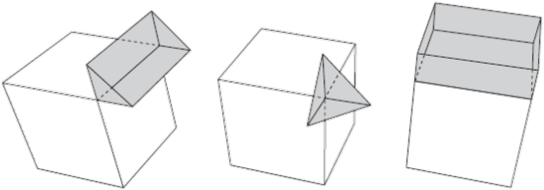
Useful observation: Two sets of points **intersect** if and only if their **Minkowski difference contains the origin.**

Voronoi regions



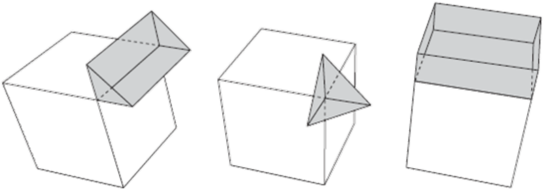
- Given a set S of points in the plane, the Voronoi region of a point P in S is defined as the set of points in the plane closer to (or as close to) P than to any other points in S.

Voronoi regions



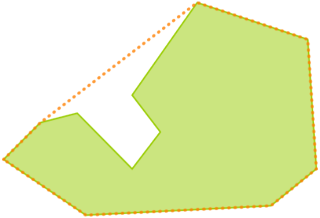
- Within a collision detection context, given a polyhedron P, let a feature of P be one of its vertices, edges, or faces. The Voronoi region of a feature of P is the **set of points** in space **closer to (or as close to) the feature than to any other feature of P.**

Voronoi regions



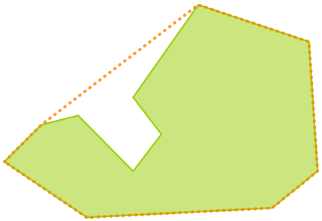
- Within a collision detection context, given a polyhedron P, let a feature of P be one of its vertices, edges, or faces. The Voronoi region of a feature of P is the **set of points** in space **closer to (or as close to) the feature than to any other feature of P**.
- **Application**: to find the closest feature to a point, find the Voronoi region that contains it.

convex hull



- Question 1: testing the convexity of a polygon/polyhedron
- Question 2: Computing a convex hull

convex hull



- Question 1: testing the convexity of a polygon/polyhedron
- Question 2: Computing a convex hull

For solutions, see:

- Andrew's algorithm
- Quickhull algorithm

Outline

Summary

- Quick introduction to design issues
- Algorithmic warm up (finish the warm up at home)

To take things further: Read **Chapters 1 and 2** of Ericson (2004).

Coming up

- Wed 9am: Tutorial 05 solutions **try and solve at home first**
- Wed 10am: Tutorial 06 **Read notes first**
- Wed 11am: Lecture on **Bounding volumes**

References

- Andrew, A. M. (1979). Another efficient algorithm for convex hulls in two dimensions. Information Processing Letters, 9(5), 216-219.
- Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. ACM Transactions on Mathematical Software (TOMS), 22(4), 469-483.
- Ericson, C. (2004). Real-time collision detection. CRC Press.