Johns Hopkins
Engineering for Professionals

605.767 Applied Computer Graphics

Brian Russin

Module 10A
Intersection Principles

Module Outline

- Introduction
  - Useful concepts
  - Collision detection overview
- Collision detection with rays
- Geometry intersection
  - Triangle/triangle, triangle/box
  - BV/BV intersections
- Dynamic intersection testing
  - Sphere/plane, sphere/sphere, sphere/polygon

Hyperplanes and Half Spaces

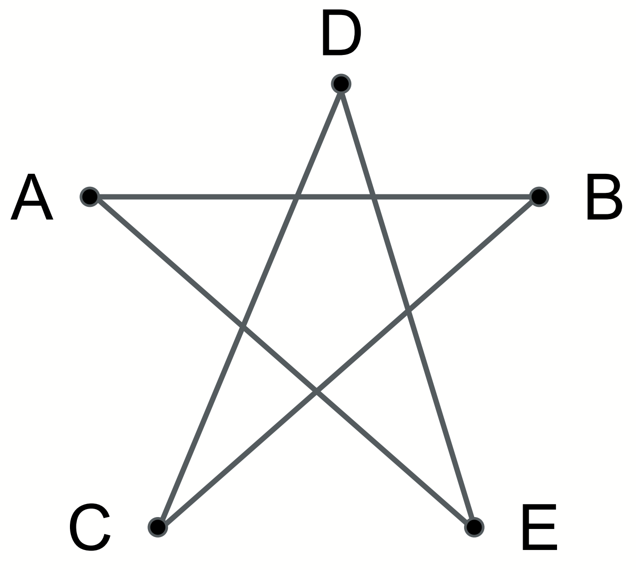
- Plane
  - 3D hyperplane
  - divides space into two halves
    - Above plane: in direction of the normal
- Hyperplane
  - Divides n-dimensional space into half spaces
  - 2D Hyperplane: line
- Best represented in point-normal form
  - e.g. $Ax + By + Cz - D = 0$ (from $Ax + By + Cz = D$)
    - [A, B, C] is the normal vector
    - D obtained by solving for any point on the plane

Convex Polytope

- Definitions
  - All points in direct line-of-sight of all other points
  - Entire polytope in one half-space of each facet’s hyperplane
- Simplifying assumption
  - Many algorithms can perform in O(n) if convex
- Convex hull
  - Useful bounding volume

Convexity Tests

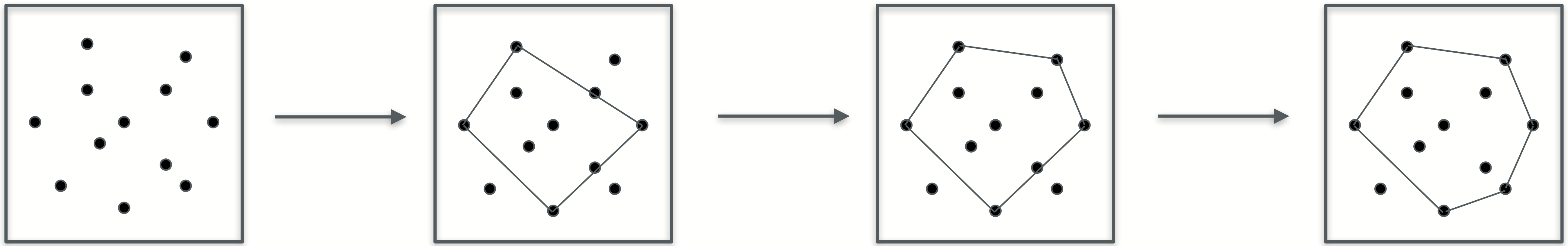
- For all edges, all other edges on the same side of its hyperplane
  - Complete test
  - Also works for 3D
  - Runs in $O(n^2)$ time
- O(n) Tests
  - Interior angle test
    - All interior angles between 0 and 180 degrees
    - Fails for pentagram
  - Change of direction
    - Only two changes in direction along any given axis while moving from vertex to vertex
    - Fails if all points are in straight line
  - Combination of Interior-angle and Change-of-direction tests accounts for each other's edge cases



Pentagram fails the interior angle test

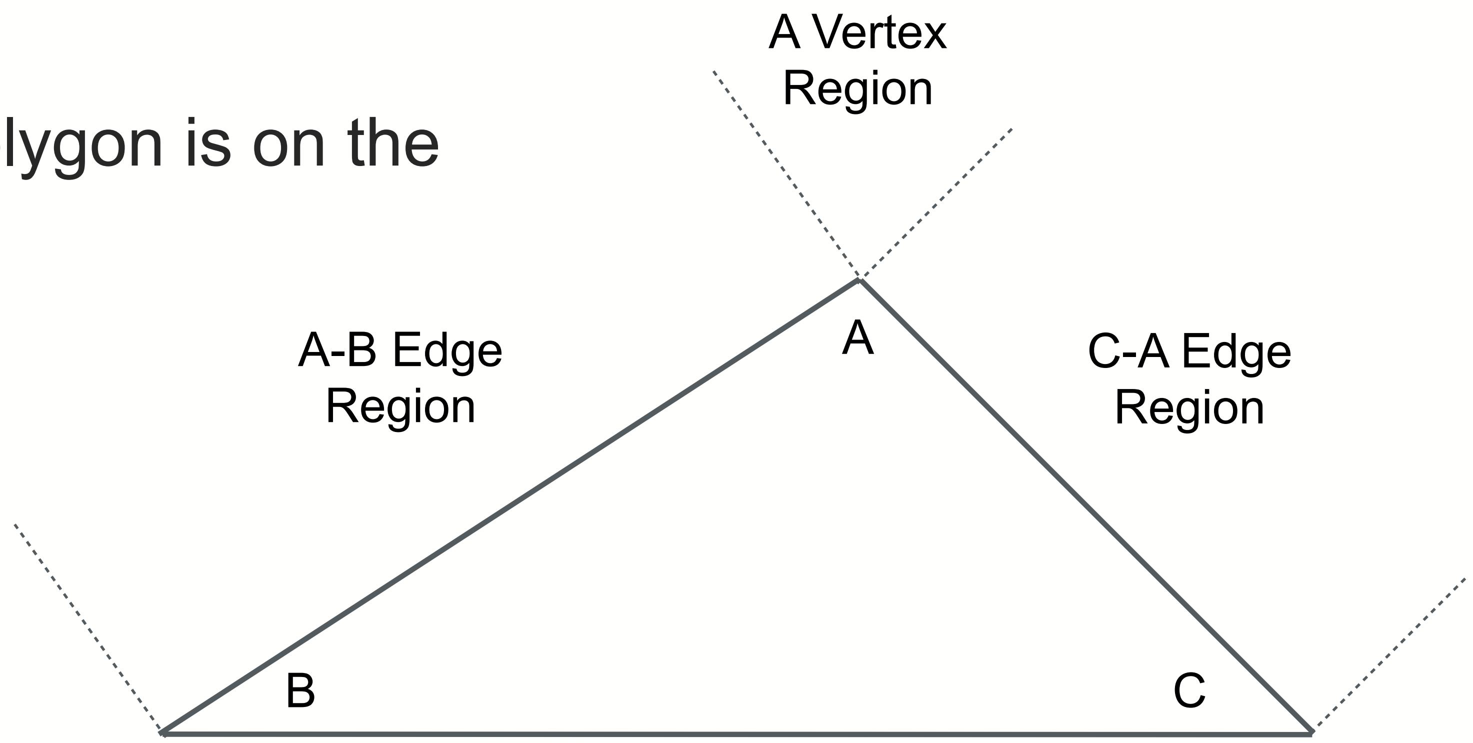
Convex Hulls

- Tight bounding volume
- Many different algorithms
- Quickhull
  - Create simple convex polygon inside the group of points
  - Iterate over all points and expand hull as needed
  - Easily adapted for 3D



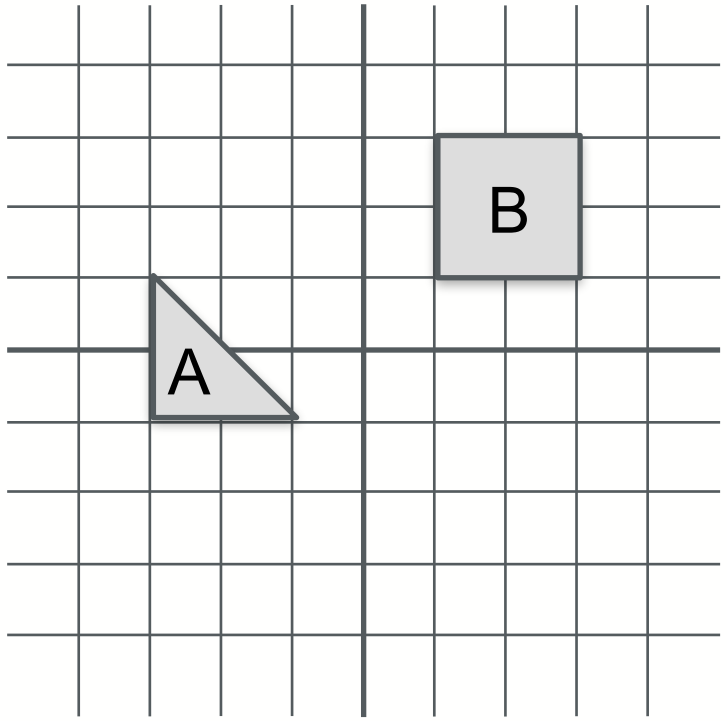
Voronoi Regions

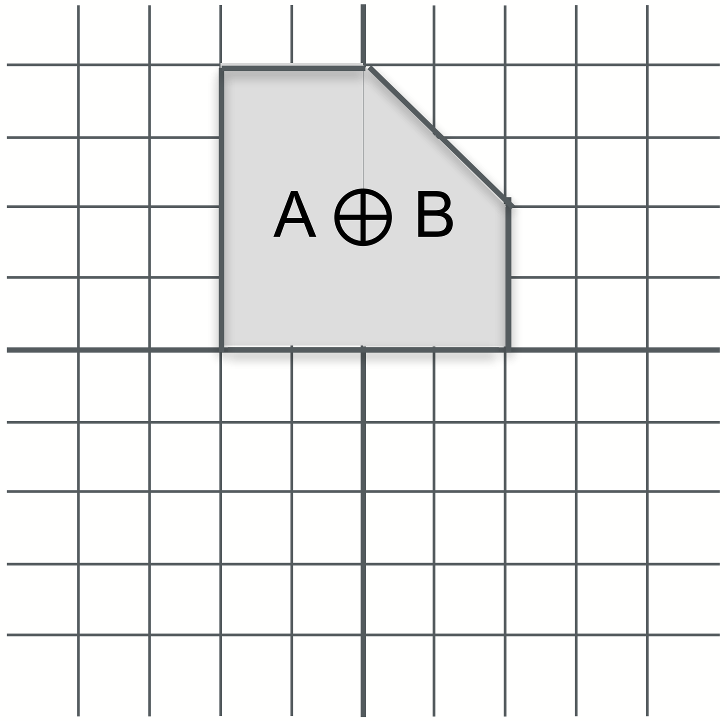
- Used in many collision tests
- Classify regions outside of a polygon/polyhedron
  - Vertex Region
    - all points whose closest point on the polygon is the vertex
  - Edge Region
    - all points whose closest point on the polygon is on the edge
  - Face Region (3D)
- Not normally computed explicitly



Minkowski Sum

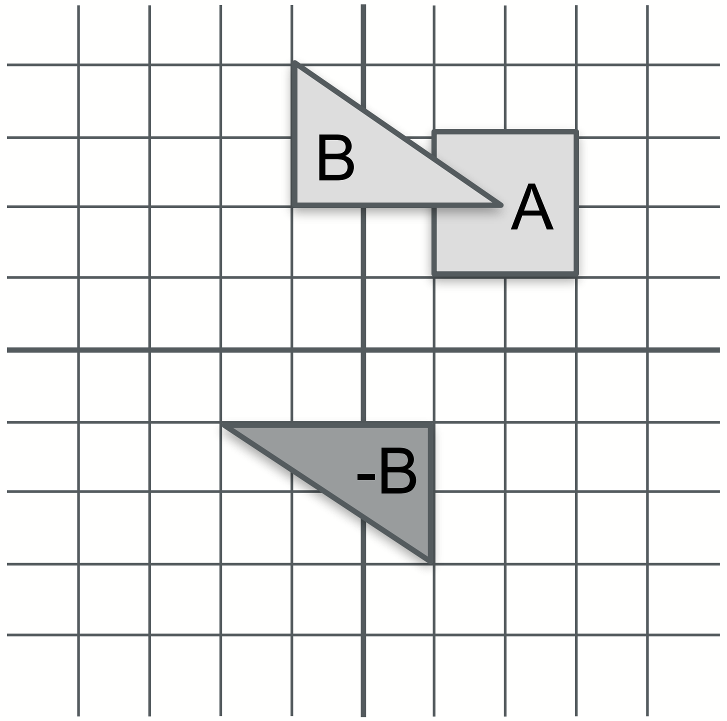
- Combining two points sets
  - $A \oplus B = \{\mathrm{a} + \mathrm{b} : \mathrm{a} \in A, \mathrm{b} \in B\}$
  - Vector sum of the all the position vectors
- Similar concept to sphere-swept volumes seen in Module 5
- Algorithms to compute Minkowski sums operate in O(n) time
  - https://cp-algorithms.com/geometry/minkowski.html
- Usually used conceptually, not explicitly

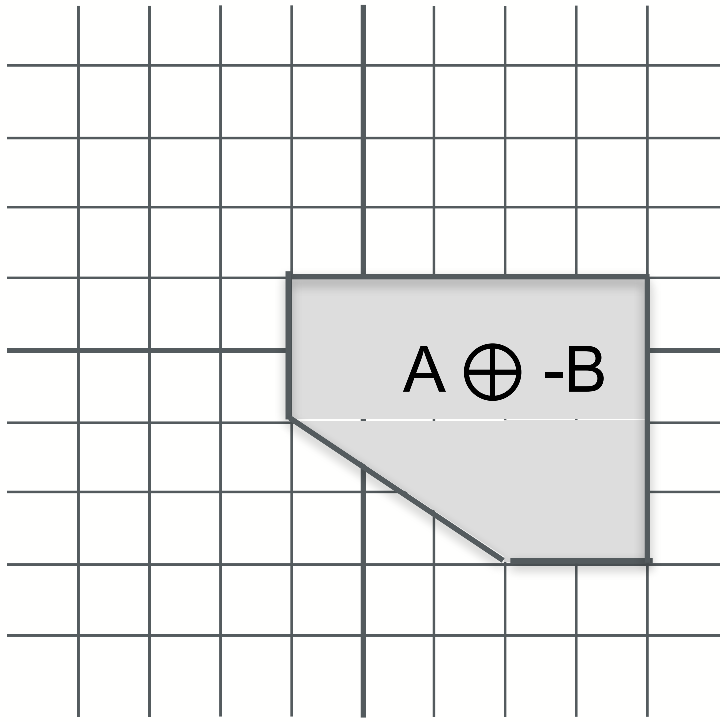




Minkowski Difference

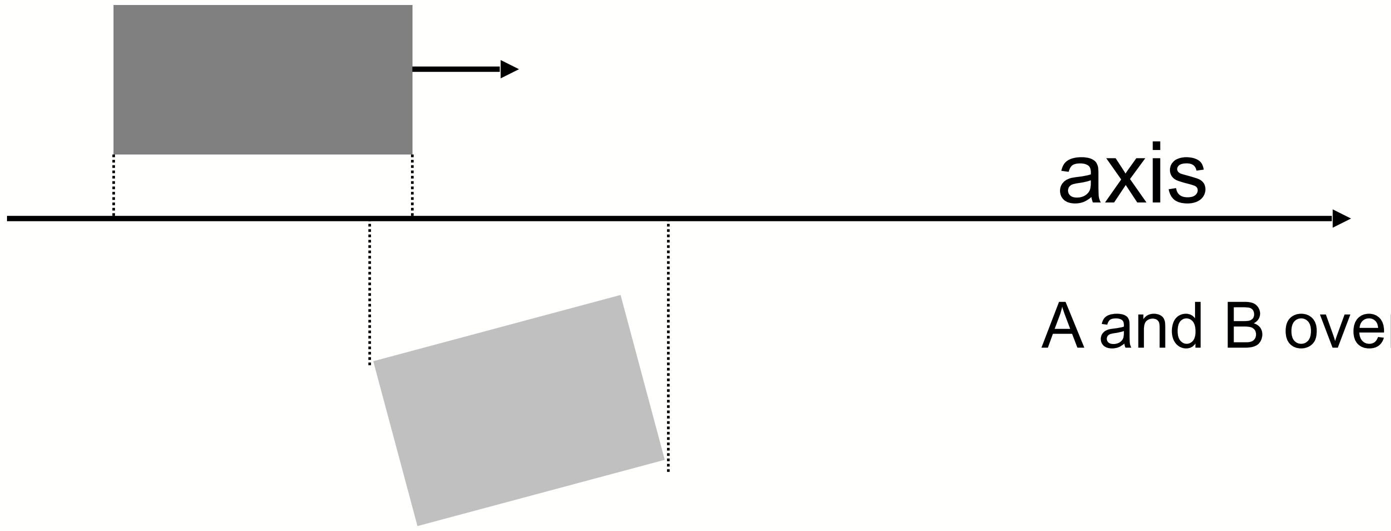
- To compute the difference
  - $A \ominus B = A \oplus -B$
  - Reflect about the origin
- There is an intersection if $A \ominus B$ contains the origin





Separating Axis Theorem

- Two convex polyhedrons, A and B, are disjoint if any of the following axes separate the objects:
  - An axis orthogonal to a face of A
  - An axis orthogonal to a face of B
  - An axis formed from the cross product of one edge from each of A and B



A and B overlap on this axis

Collision Handling

- Many applications require collision detection
  - CAD/CAM, computer animation, games, flight and vehicle simulators, robotics, virtual reality
- 3 parts to **collision handling**
  - **Collision detection**
    - Do the objects collide (boolean)
  - **Collision determination**
    - Finds actual intersections between objects
  - **Collision response**
    - Determines what actions should be taken in response to collision of 2 objects

Problem Description

- Brute force collision detection
  - Test each triangle of one object against each triangle of another
    - When objects are far from one another we expect a quick “no intersect” determination
      - Brute force is too inefficient
    - Even when objects are close there are better ways
- Scenes often contain sets of both moving and stationary objects
  - Scene with n moving objects and m stationary objects
  - Naïve solution involves:
    - nm tests of moving objects against static objects
    - n^2 tests of moving objects against each other
      - actually: n*(n-1)
- Efficient collision detection involves BV creation and testing
- Will address some further intersection computations in the next few lectures

Collision Test Guidelines

- From Haines and Moller 22.5 (16.5 in 3rd Edition)
  - Perform rejection/acceptance as early as possible
  - Exploit results from previous tests
  - Try changing the **order of rejection/acceptance cases**
    - Do not assume that what appears to be a minor change will have no effect
  - Postpone expensive calculations
    - Especially trigonometric functions, square roots, and divisions
  - **Reduce dimensions**
    - 3D objects can sometimes be cast onto a 2D plane for simpler tests
  - Use pre-calculations if performing one-to-many tests
    - e.g. testing one ray against multiple objects: normalize the ray direction before testing against the set
  - Perform simpler tests before expensive tests
    - e.g. check bounding sphere before convex hull
  - Benchmark your algorithms and test on real data
    - Also test on multiple platforms if possible
  - **Exploit results from the previous frame**
    - e.g. If a certain plane or axis separates two objects in the previous frame, it probably will separate them again this frame
  - Make your code robust; handle all edge cases
    - Also be sensitive to floating point precision errors