More on convex polygons

(Not particularly interesting for the AABB special case: but needed for dealing with Triangles, non-axis-aligned rectangles, general convex polygons)

SAT can be used to determine whether convex polygons A and B overlap: *method*

- **for each line** from adjacent vertices A_J A_K of convex polygon A,
  - calculate the normal vector passing through origin, { A_{Ky}-A_{Jy} , A_{Jx}-A_{Kx} } (r n)
  - record the min and max scalar projections of *all* vertices of A onto this vector
  - record the min and max scalar projections of all vertices of B onto this vector
    - use vector dot product: the normal-through-origin and the origin-to-point
  - if A_{min}>=B_{max} or B_{min}>=A_{max}, **stop** – there is no overlap (= allows for touching)
- **do likewise** for lines from adjacent vertices of polygon B

- if you haven't stopped yet, **stop** – there is overlap

Non-convex polygons

A general polygon with concavities can be made by joining several convex polygons.

- Polygons with concavities must have more than three vertices: there is always more than one way to carve up a many-sided polygon into smaller parts
- Intersection of general polygons can be handled by breaking them into convex polygons, then determining intersections of these smaller parts

However, if collision detection is required to give more than yes/no information about collisions between two shapes, say for use in collision response, it may be necessary to distinguish between internal edges and external edges:

eg for platforms (even convex ones) with start-middle-end sections – you want to check for a bounce off an external edge but not an internal one

(if shallow approach as below, 'deepest penetration' & 'direction' both mislead)

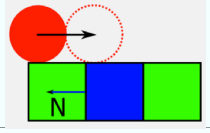
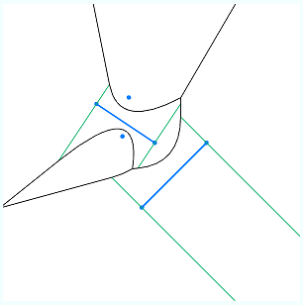


diagram from wildbunny.co.uk/blog/2011/12/14/how-to-make-a-2d-platform-game-part-2-collision-detection/

Curved shapes

Whereas bounding circles can easily be tested against each other for overlap,

- Circle-vs-AABB can be tested using the idea of *Voronoi regions*
  – regions nearest some feature of a shape: an edge, a corner



This Voronoi region example is taken from boost.org/doc/libs/1_65_0/libs/polygon/doc/voronoi_basic_tutorial.htm

Features here are single points, lines, their endpoints, Voronoi regions may extend to infinity, their boundaries have straight-line and parabolic segments

  –cf exploration, fishing rights at sea, water pumps & 1853 London Cholera Epidemic

- (vs non-AA BBs, it is best to work in the transformed coordinate system of BB)

Testing overlap of circle and AABB

If Cx Cy Cr describe a circle, BoxL BoxR BoxT BoxB describe a box (BoxT>BoxB)

Simple and efficient algorithm, not using sin or cos or sqrt.

It assumes Y increases upwards.

It returns true iff Cx,Cy is wholly inside the box or close enough for the circle to intersect the box.

```
d=0
if Cx>BoxR {s=Cx-BoxR; d+=s*s}
elif Cx<BoxL {s=Cx-BoxL; d+=s*s};
if Cy<BoxB {S=Cy-BoxB; d+=s*s}
elif Cy>BoxT {s=Cy-BoxT; d+=s*s};
return (d < Cr*Cr)
```

Voronoi regions in red are nearest to one particular edge, those in green are nearest to one corner. The boundary lines extend to infinity. A circle with centre Cx, Cx and radius Cr goes through points x, y satisfying

$(x-Cx)^2 + (y-Cy)^2 = Cr^2$

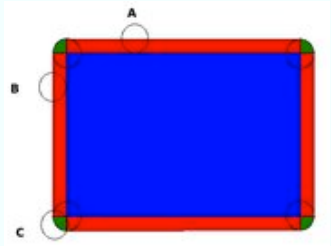


diagram from allegro.cc/forums/thread/604103

Collisions involving non-circular curved shapes

The logic to deal with semicircles and quarter-circles is easy enough to be practical, especially if axis-aligned

Dealing with semi- and quarter-circle concavities in polygons also quite feasible

Other curved shapes, ellipses etc, are best avoided altogether:

- game artwork should take implementation practicalities into account.

Such shapes, if really really required, can be dealt with via rasterization.

- geometric reasoning about them is complex enough to threaten frame rates.

Intersections of lines

Represent lines in form a.x+b.y+c=0 rather than y=slope.x+intercept: no infinities

Represent a line from P to Q parametrically: P+u*(Q-P), **0<=u<=1**

Represent a line *through* P & Q the same way: $-\infty <= u <= +\infty$

In 2D, two infinite lines through P,Q and through R,S intersect unless parallel:
if denominator not zero in solution equations

$$u1 = \frac{((S_x-R_x).(P_y-R_y) - (S_y-R_y).(P_x-R_x))}{((S_y-R_y).(Q_x-P_x) - (S_x-R_x).(Q_y-P_y))}$$

$$u2 = \frac{((Q_x-P_x).(P_y-R_y) - (Q_y-P_y).(P_x-R_x))}{((S_y-R_y).(Q_x-P_x) - (S_x-R_x).(Q_y-P_y))}$$

Two finite lines in 2D intersect if non-parallel and 0<=u1<=1 and 0<=u2<=1

Intersections of circle and line (eg ball and wall)

Simplify matters by subtracting Cx Cy from the points describing the line from P to Q

- In the circle's coordinate system, the line is described by P'+u.(Q'-P'), ie
  $X=P'_x+u.(Q'_x-P'_x)$ and $Y=P'_y+u.(Q'_y-P'_y)$
- Substitute into circle equation $X^2+Y^2=Cr^2$ & manipulate to get quadratic of form
  - $au^2 + bu + c = 0$
- hoping quadratic solution is familiar … $(-b \pm \sqrt{(b^2 -4.a.c)})/2.a$
- if now $b^2-4ac < 0$, the line does not intersect circle;
- if = 0, it is tangential;
- if > 0, line intersects circle. There are two solutions to the quadratic, u1 and u2:
  - If 0<=u1<1 there's an intersection at parameter value u1; likewise u2