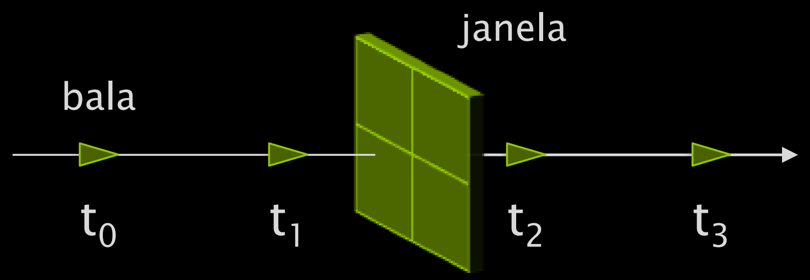
Colisão Avançada

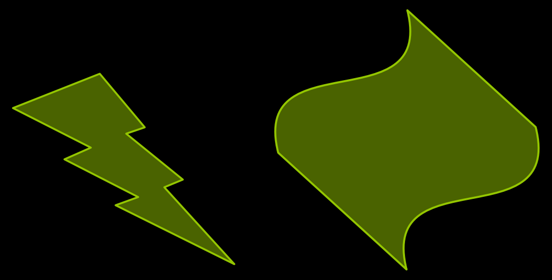
Programação de Jogos

Judson Santos Santiago

Introdução

- Detectar colisão não é uma tarefa fácil
  - Movimentação muito rápida
    Ex.: balas em um jogo de tiro
  - Formas geométricas complexas
    Ex.: personagens segurando armas
  - Custo elevado: cada objeto deve ser testado contra todos os demais objetos da cena, procedimento com custo $O(n^2)$





Formas complexas

Introdução

- Uma solução para se lidar com geometrias complexas é simplificar a geometria usando uma bounding box

- No curso trabalhamos com as geometrias:
  - Ponto
  - Círculo
  - Retângulo
  - Mista

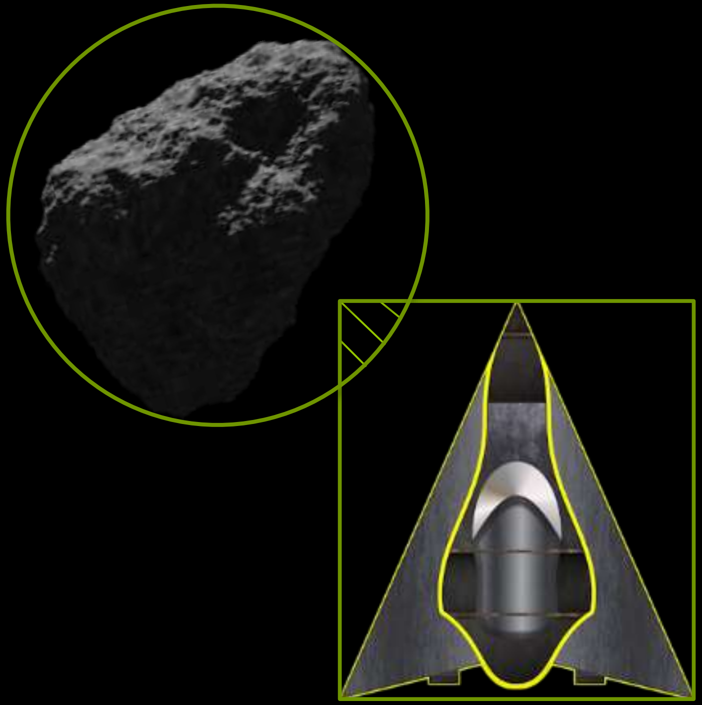
Bounding Box Circular

Bounding Box Retangular

Introdução

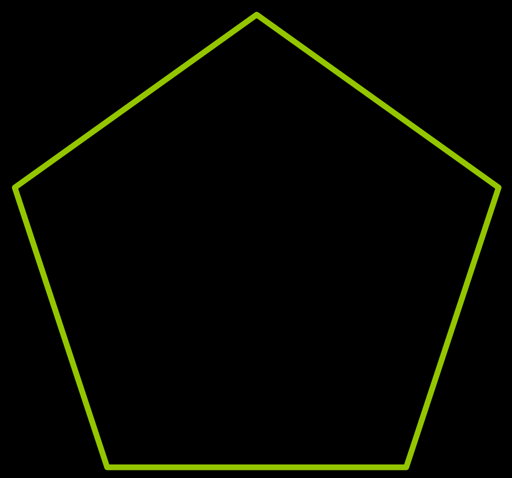
- Ainda assim, em alguns jogos é necessário trabalhar com geometrias mais fiéis aos objetos representados

Falso positivo é um dos principais problemas da utilização de bounding boxes na detecção de colisão.

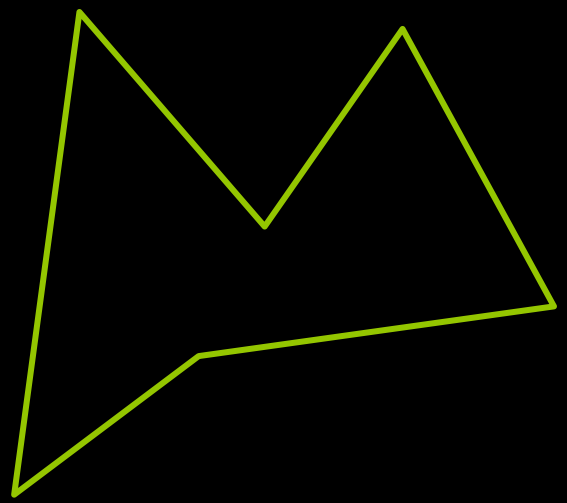


Polígonos

- Na geometria, um polígono é uma figura plana limitada por uma linha poligonal fechada



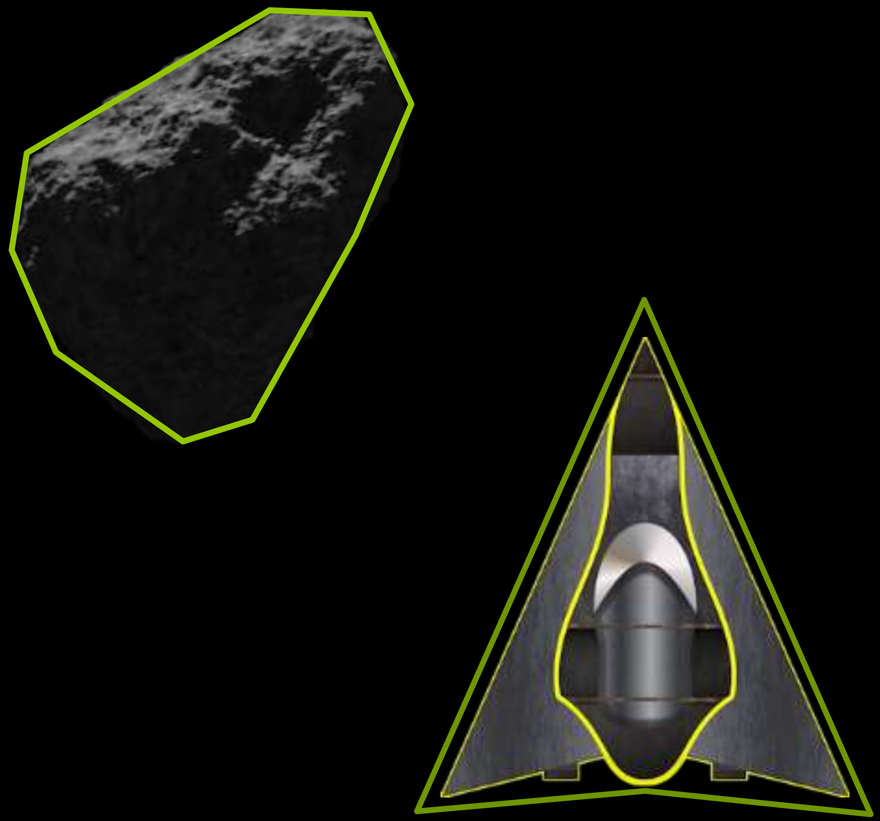
Polígono Convexo



Polígono Côncavo

Polígonos

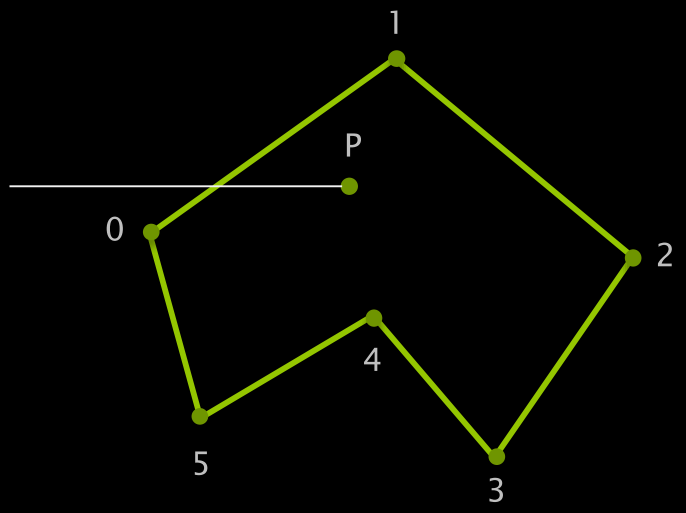
- Um polígono pode ser usado para obter uma bounding box mais fiel a forma dos objetos



Existe uma forma de detectar colisão entre polígonos e outras geometrias?

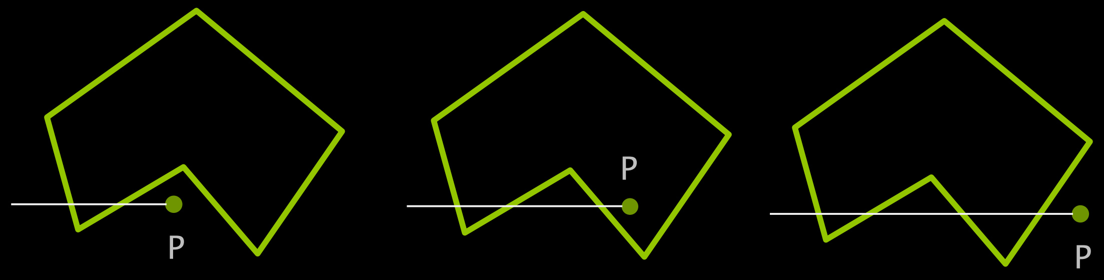
Colisão Ponto–Polígono

- O algoritmo crossing é o melhor algoritmo genérico (polígonos côncavos e convexos) existente na literatura



1 cruzamento

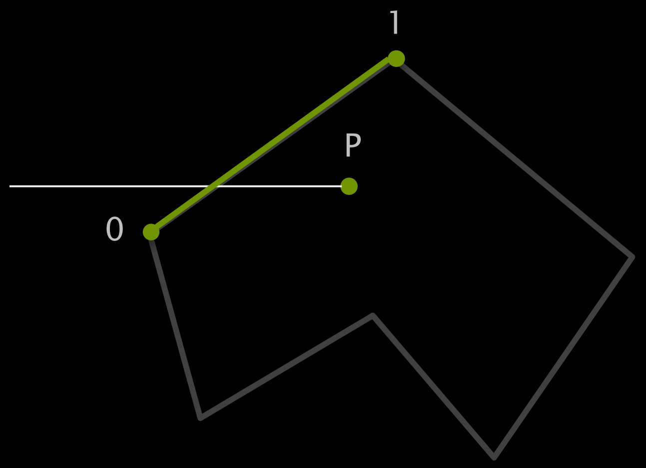
Crossing: se uma linha traçada do infinito até o ponto P cruzar um número ímpar de arestas, o ponto está dentro do polígono, caso contrário ele está fora.

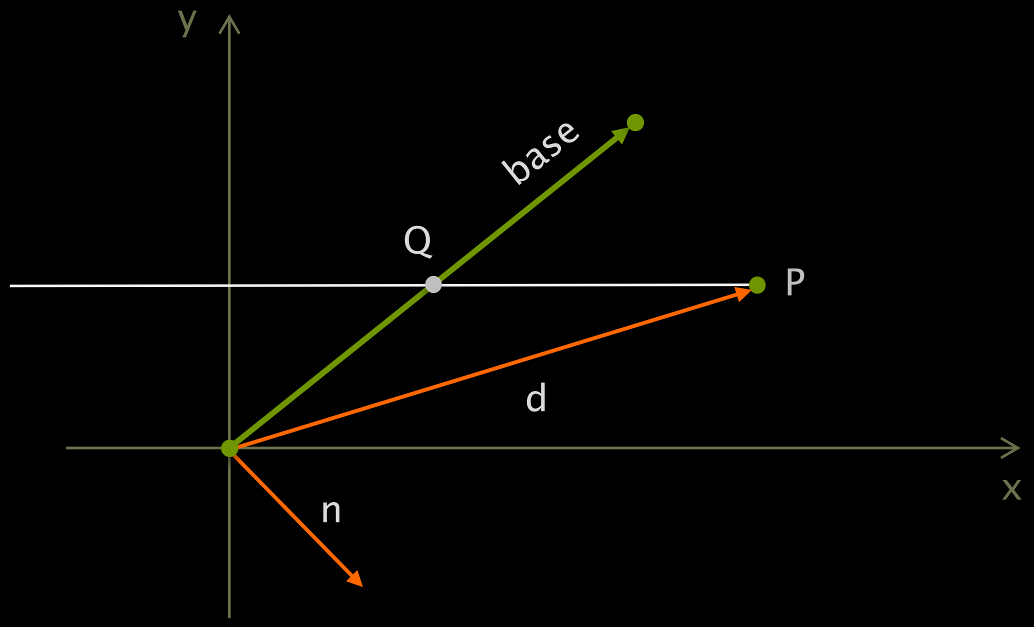


2 cruzamentos 3 cruzamentos 4 cruzamentos

Colisão Ponto–Polígono

- Para utilizar o crossing é preciso saber de que lado o ponto está em relação a cada aresta





$P = Q$ quando $\boxed{d \cdot n = 0}$

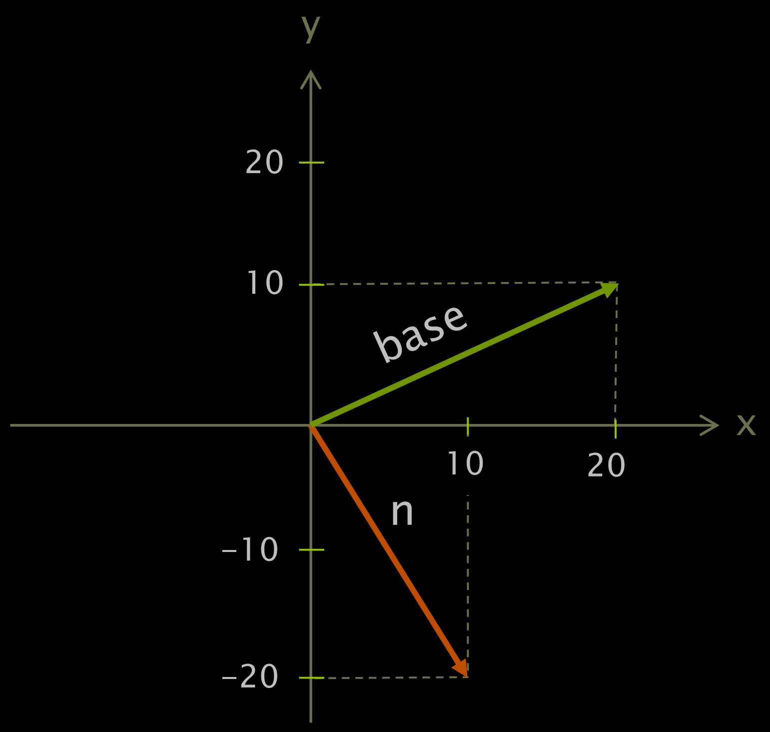
Se $\boxed{d \cdot n > 0}$ então P está do lado da normal

Colisão Ponto–Polígono

- O vetor normal é o vetor base rotacionado de –90º

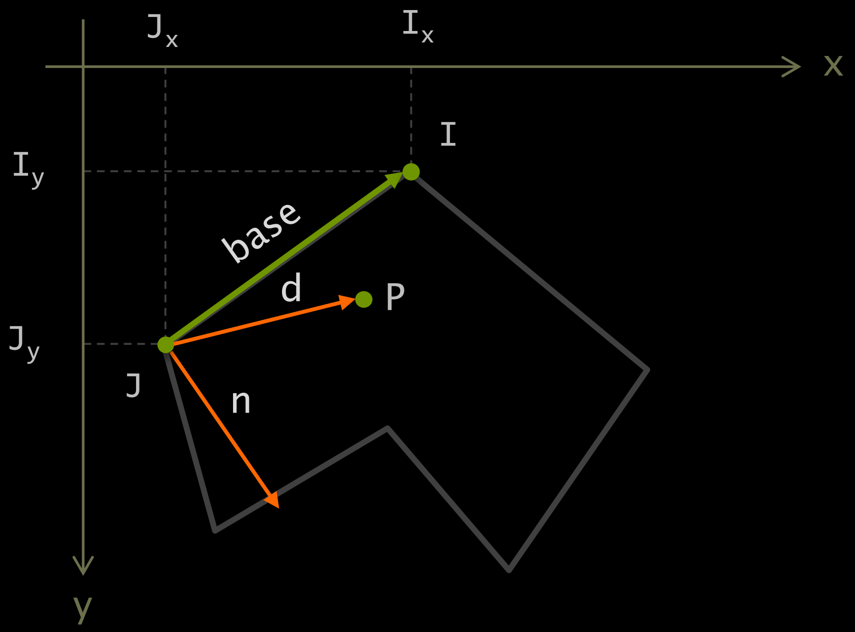
Para rotacionar
um vetor por –90º:

$$n_x = +base_y$$
$$n_y = -base_x$$

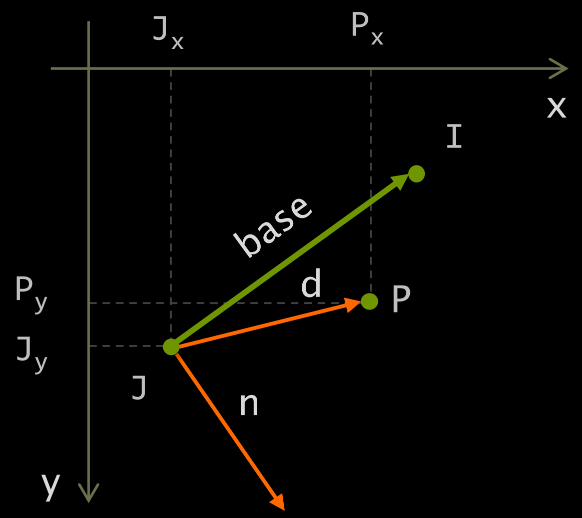


Colisão Ponto-Polígono

- Fazendo os cálculos:



$$n \begin{cases} n_x = +base_y = J_y - I_y \\ n_y = -base_x = -(I_x - J_x) = J_x - I_x \end{cases}$$

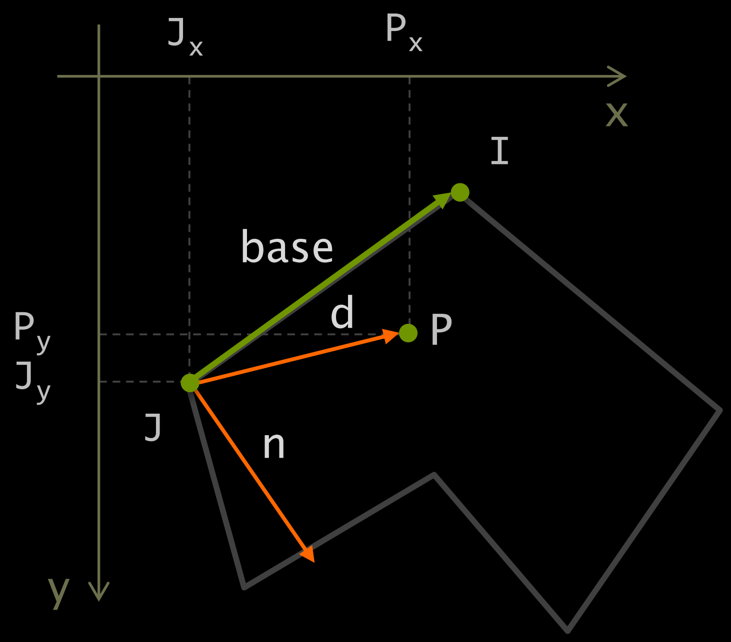


$$d \begin{cases} d_x = P_x - J_x \\ d_y = J_y - P_y \end{cases}$$

Se $\boxed{d \cdot n > 0}$ então existe cruzamento

Colisão Ponto-Polígono

- Resolvendo o produto escalar, chegamos à fórmula utilizada no algoritmo de crossing



Se $\boxed{d \cdot n > 0}$ então existe cruzamento

$$d \cdot n > 0$$

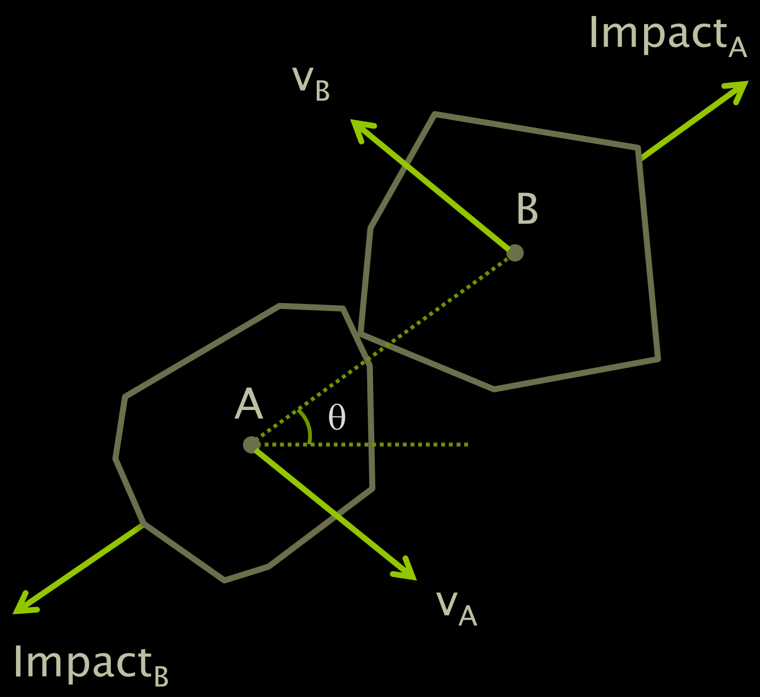
$$d_x * n_x + d_y * n_y > 0$$

$$(P_x - J_x)*(J_y - I_y) + (J_y - P_y)*(J_x - I_x) > 0$$

$$\boxed{P_x > J_x - \frac{(J_y - P_y)(J_x - I_x)}{(J_y - I_y)}}$$

Asteroides

- Resolução da colisão




```
// centros das rochas
Point pA { rockA->X(), rockA->Y() };
Point pB { rockB->X(), rockB->Y() };

// ângulo formado pela linha que interliga os centros
float angleA = Line::Angle(pA, pB);
float angleB = angleA + 180.0f;

// vetores gerados no impacto (com 25% de perda)
Vector impactA { angleA, 0.75f * rockA->speed.Magnitude() };
Vector impactB { angleB, 0.75f * rockB->speed.Magnitude() };

// adiciona vetor impacto à velocidade das rochas
rockA->speed.Add(impactB);
rockB->speed.Add(impactA);
```

Resumo

- A detecção de colisão é um problema complexo
  - Requer o uso de bounding boxes para simplificação de geometrias
  - O polígono obtém uma bounding box mais fiel aos objetos
- O cálculo da colisão com polígonos possui um custo mais elevado
  - Podemos utilizar outras geometrias para reduzir esse custo