

Geometria Computacional: Principais Algoritmos Usados

Luiz Fernando Martha – Depto. de Eng. Civil – PUC-Rio

e

André Maués Brabo Pereira – Depto. de Eng. Civil – UFF

Adaptado para a disciplina:

CIV2802 – Sistemas Gráficos para Engenharia

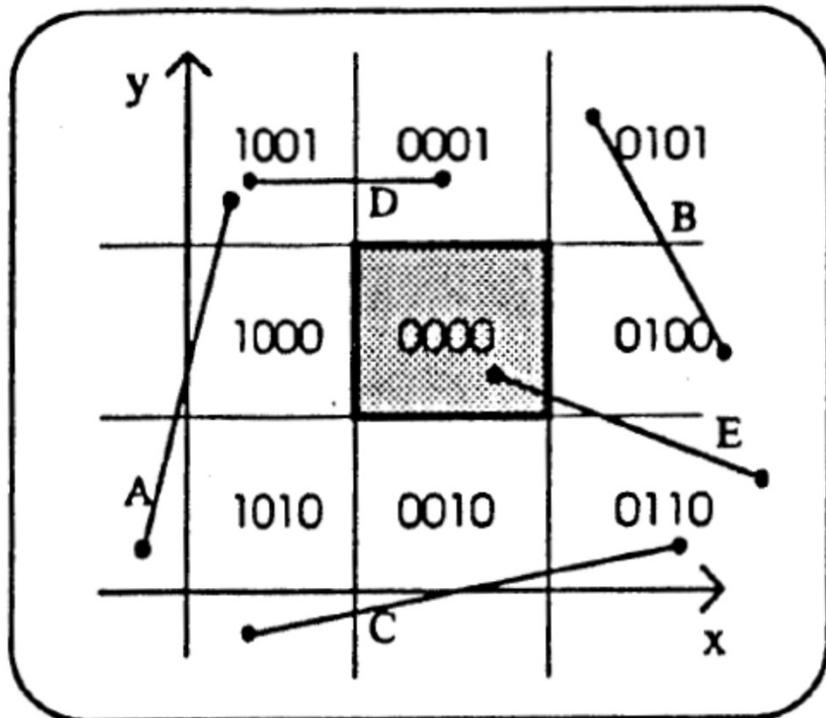
Conteúdo

- Pick de segmento de reta
- Ponto mais próximo em um segmento de reta
- Tesselagem de polígonos
- Interseção de segmentos de reta
- Verificação de inclusão de ponto em polígono

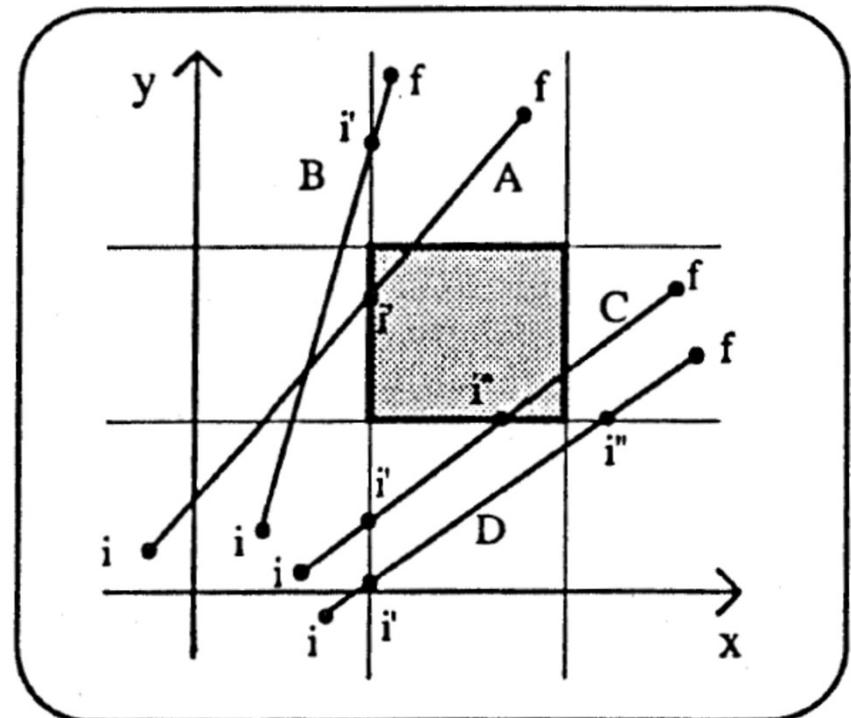
Pick de Segmento de Reta

Seleção (Pick) de Segmento de Reta (baseada no algoritmo de cerceamento de linhas)

Regiões com o mesmo código e posições triviais
(A, B, C, D):



Algoritmo recursivo para recair nos casos triviais:



```

# -----
# pickLine: check to see if given point (x,y) touches given line
# (x0,y0)-(x1,y1) based on given tolerance. Uses auxiliary method
# pickCode.
def pickCode(x, y, xmin, xmax, ymin, ymax):
    cod = [x < xmin, x > xmax, y < ymin, y > ymax]
    return cod

def pickLine(x0, y0, x1, y1, x, y, tol):
    # Define attraction window.
    xmin = x - tol
    xmax = x + tol
    ymin = y - tol
    ymax = y + tol

    cod1 = Curve.pickCode(x1, y1, xmin, xmax, ymin, ymax)
    while True:
        cod0 = Curve.pickCode(x0, y0, xmin, xmax, ymin, ymax)

        count = 0
        for i in range(0, 4):
            if cod0[i] and cod1[i]: # Test no-trivial pick
                break
            else:
                count += 1
        if count != 4:
            break

        # Move point 0 to window limit.
        if cod0[0]:
            y0 += (xmin - x0) * (y1 - y0) / (x1 - x0)
            x0 = xmin
        elif cod0[1]:
            y0 += (xmax - x0) * (y1 - y0) / (x1 - x0)
            x0 = xmax
        elif cod0[2]:
            x0 += (ymin - y0) * (x1 - x0) / (y1 - y0)
            y0 = ymin
        elif cod0[3]:
            x0 += (ymax - y0) * (x1 - x0) / (y1 - y0)
            y0 = ymax
        else:
            return True

    return False

```

Ponto mais Próximo em um Segmento de Reta

PONTO MAIS PRÓXIMO EM UM SEGMENTO DE RETA UTILIZANDO PRODUTO INTERNO

Projeção do ponto C na reta AB:

$$C' = A + t_{C'}(B - A)$$

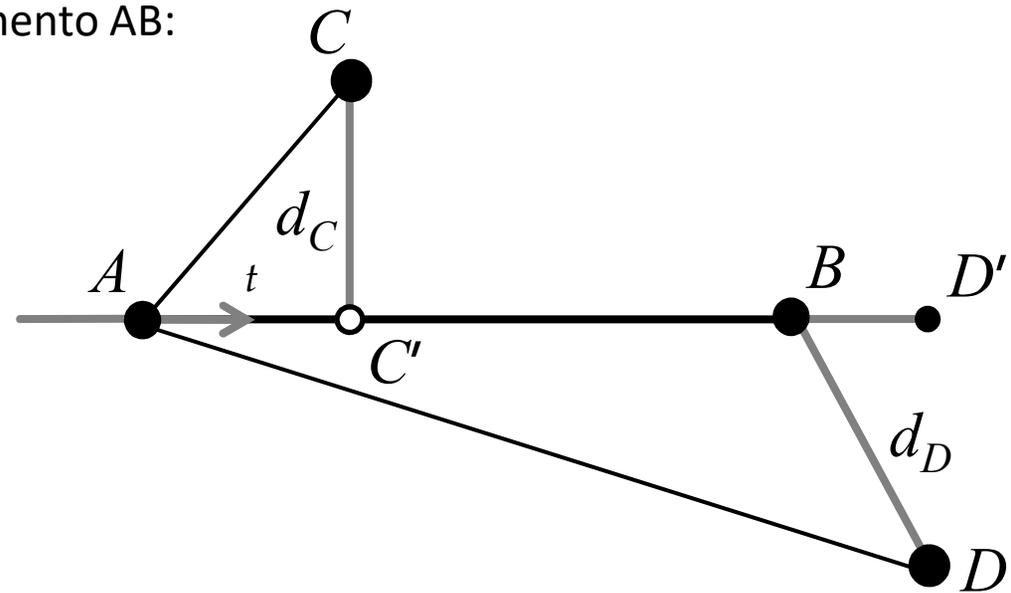
Ponto mais próximo de C no segmento AB:

$$P = C'$$

Valor paramétrico do ponto C' no segmento AB:

$$t_{C'} = \frac{\overrightarrow{AB} \circ \overrightarrow{AC}}{|\overrightarrow{AB}|^2} \quad (\text{produto interno})$$

$$0 < t_{C'} < 1$$



Projeção do ponto D na reta AB:

$$D' = A + t_{D'}(D - C)$$

Valor paramétrico do ponto D' no segmento AB:

$$t_{D'} = \frac{\overrightarrow{AB} \circ \overrightarrow{AD}}{|\overrightarrow{AB}|^2}$$

$$t_{D'} > 1$$

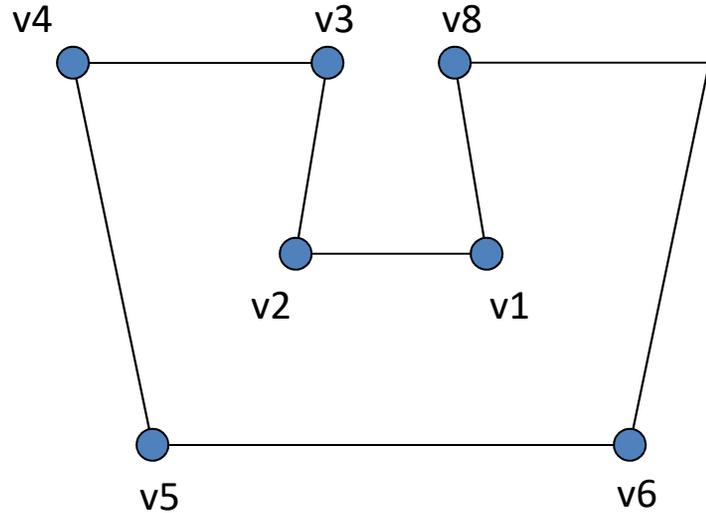
Ponto mais próximo de D no segmento AB:

$$P = B$$

Algoritmo para Tesselagem de Polígonos

Como **tecer** uma face não convexa?

Solução de SKIENA & REVILLA, 2002, *Programming Challenges*, p.319



Encontrar ORELHA do polígono até remanescer um único triângulo.

POL = 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8

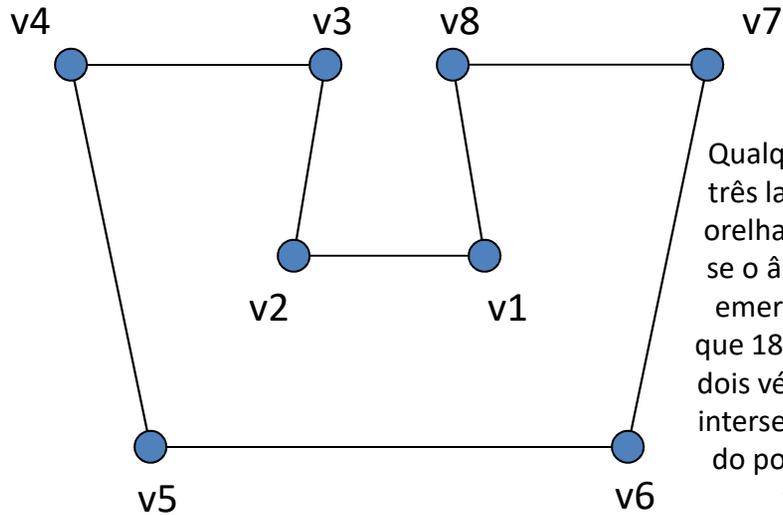
Defina duas listas com os vértices anteriores e posteriores:

L = 8 / 1 / 2 / 3 / 4 / 5 / 6 / 7

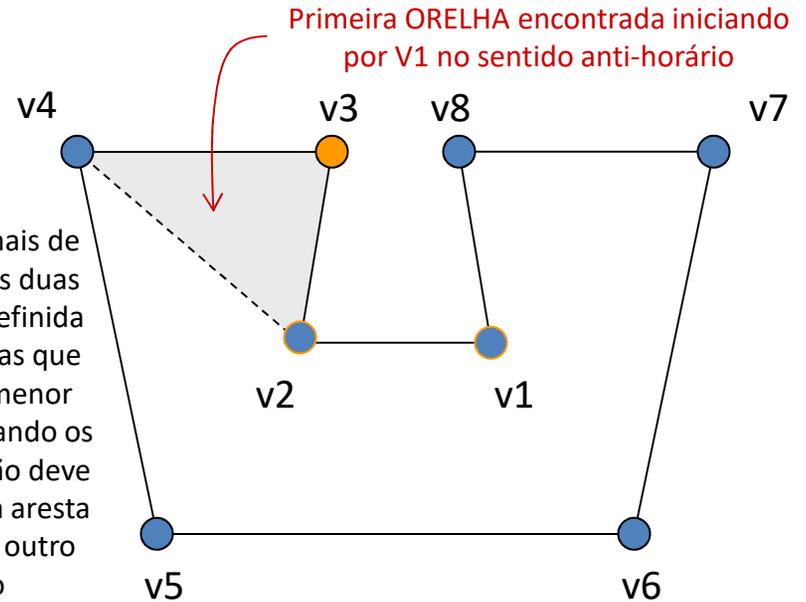
R = 2 / 3 / 4 / 5 / 6 / 7 / 8 / 1

Como tecer uma face não convexa?

Solução de SKIENA & REVILLA, 2002, *Programming Challenges*, p.319



Qualquer polígono com mais de três lados tem pelo menos duas orelhas. Uma ORELHA é definida se o ângulo entre as arestas que emergem do vértice for menor que 180° e a corda conectando os dois vértices adjacentes não deve intersectar nenhuma outra aresta do polígono (i.e. nenhum outro vértice deve ficar no triângulo/orelha).



Encontrar ORELHA do polígono até remanescer um único triângulo.

POL = 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8

Defina duas listas com os vértices anteriores e posteriores:

L = 8 / 1 / 2 / 3 / 4 / 5 / 6 / 7

R = 2 / 3 / 4 / 5 / 6 / 7 / 8 / 1

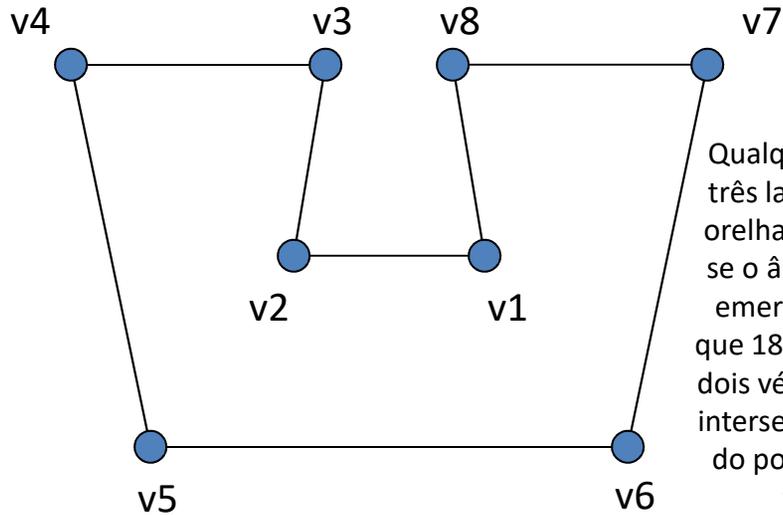
Atualize a lista após o primeiro triângulo encontrado:

L = 8 / 1 / 2 / 2 / 4 / 5 / 6 / 7

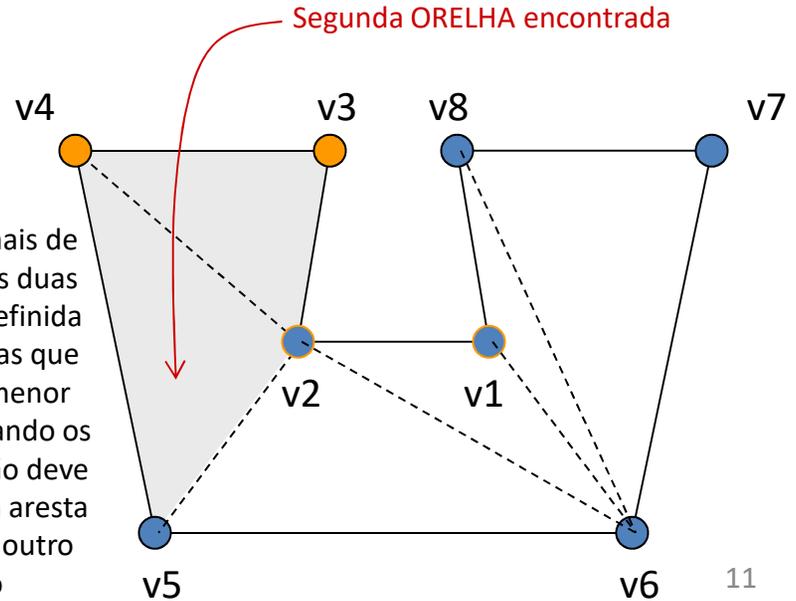
R = 2 / 4 / 4 / 5 / 6 / 7 / 8 / 1

Como tecer uma face não convexa?

Solução de SKIENA & REVILLA, 2002, *Programming Challenges*, p.319



Qualquer polígono com mais de três lados tem pelo menos duas orelhas. Uma ORELHA é definida se o ângulo entre as arestas que emergem do vértice for menor que 180° e a corda conectando os dois vértices adjacentes não deve intersectar nenhuma outra aresta do polígono (i.e. nenhum outro vértice deve ficar no triângulo/orelha).



Encontrar ORELHA do polígono até remanescer um único triângulo.

POL = 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8

Defina duas listas com os vértices anteriores e posteriores:

L = 8 / 1 / 2 / 3 / 4 / 5 / 6 / 7

R = 2 / 3 / 4 / 5 / 6 / 7 / 8 / 1

Atualize a lista após o primeiro triângulo encontrado:

L = 8 / 1 / 2 / 2 / 4 / 5 / 6 / 7

R = 2 / 4 / 4 / 5 / 6 / 7 / 8 / 1

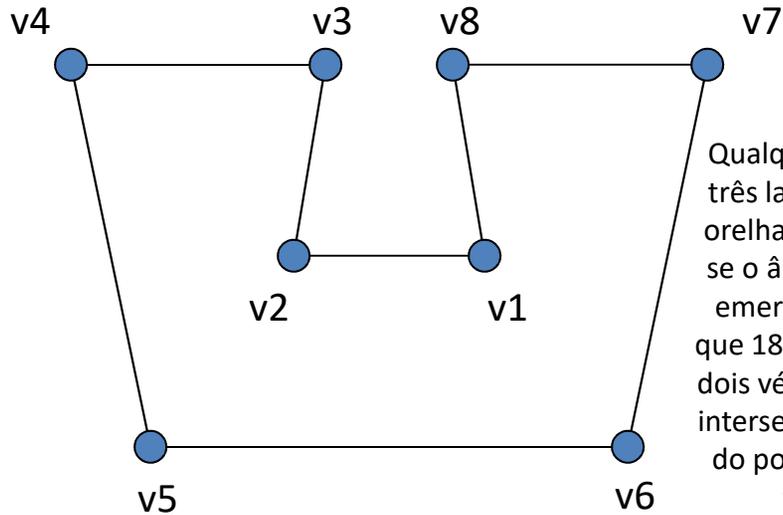
Atualize a lista após o segundo triângulo encontrado:

L = 8 / 1 / 2 / 2 / 2 / 5 / 6 / 7

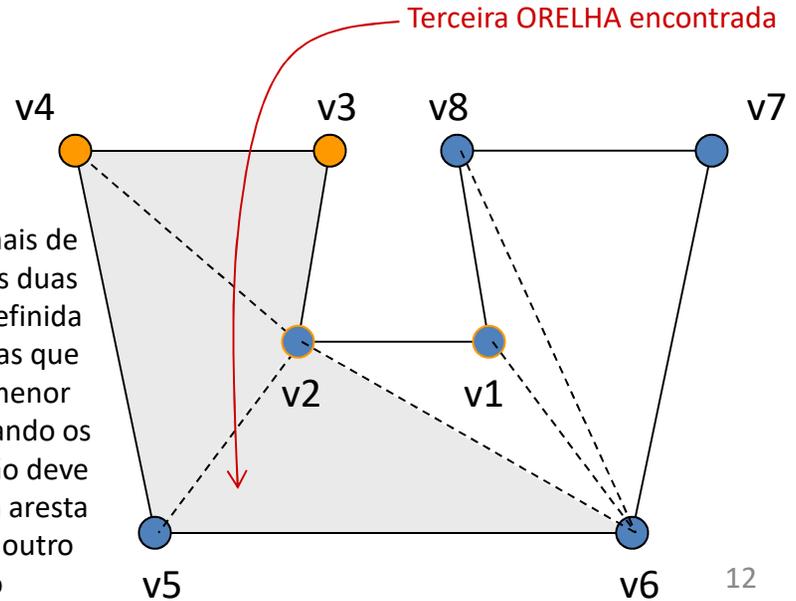
R = 2 / 5 / 4 / 5 / 6 / 7 / 8 / 1

Como tecer uma face não convexa?

Solução de SKIENA & REVILLA, 2002, *Programming Challenges*, p.319



Qualquer polígono com mais de três lados tem pelo menos duas orelhas. Uma ORELHA é definida se o ângulo entre as arestas que emergem do vértice for menor que 180° e a corda conectando os dois vértices adjacentes não deve intersectar nenhuma outra aresta do polígono (i.e. nenhum outro vértice deve ficar no triângulo/orelha).



Encontrar ORELHA do polígono até remanescer um único triângulo.

POL = 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8

Defina duas listas com os vértices anteriores e posteriores:

L = 8 / 1 / 2 / 3 / 4 / 5 / 6 / 7

R = 2 / 3 / 4 / 5 / 6 / 7 / 8 / 1

Atualize a lista após o primeiro triângulo encontrado:

L = 8 / 1 / 2 / 2 / 4 / 5 / 6 / 7

R = 2 / 4 / 4 / 5 / 6 / 7 / 8 / 1

Atualize a lista após o segundo triângulo encontrado:

L = 8 / 1 / 2 / 2 / 2 / 5 / 6 / 7

R = 2 / 5 / 4 / 5 / 6 / 7 / 8 / 1

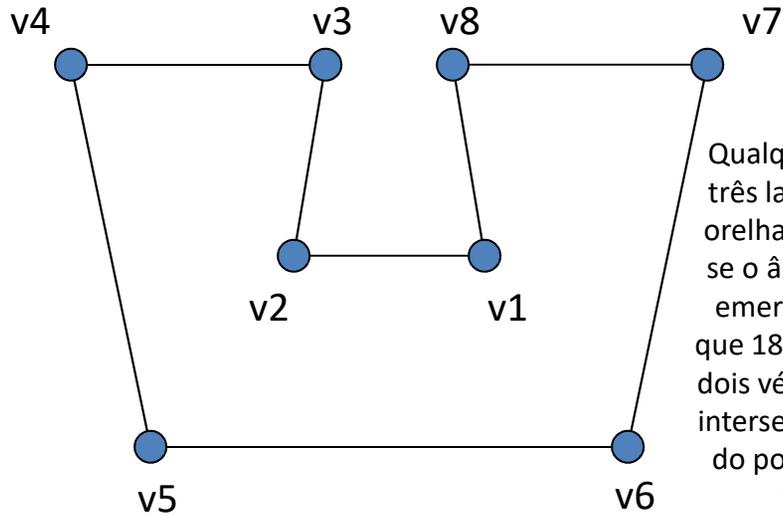
Atualize a lista após o terceiro triângulo encontrado:

L = 8 / 1 / 2 / 2 / 2 / 6 / 7

R = 2 / 6 / 4 / 5 / 6 / 7 / 8 / 1

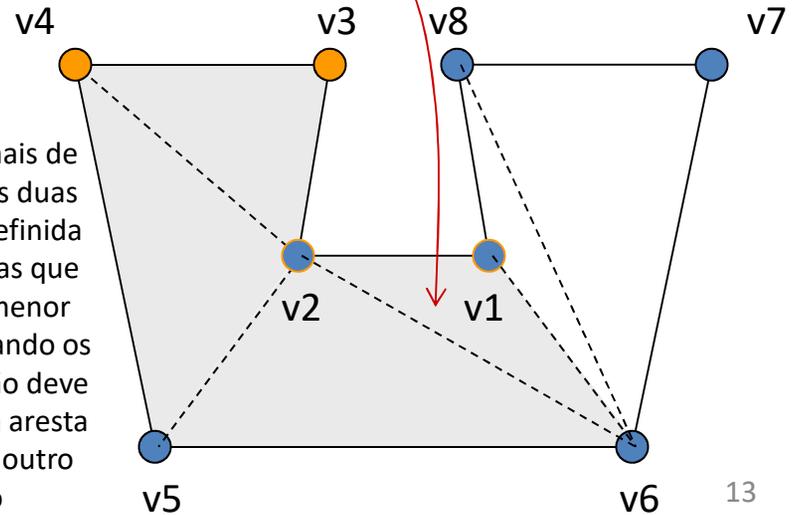
Como tecer uma face não convexa?

Solução de SKIENA & REVILLA, 2002, *Programming Challenges*, p.319



Qualquer polígono com mais de três lados tem pelo menos duas orelhas. Uma ORELHA é definida se o ângulo entre as arestas que emergem do vértice for menor que 180° e a corda conectando os dois vértices adjacentes não deve intersectar nenhuma outra aresta do polígono (i.e. nenhum outro vértice deve ficar no triângulo/orelha).

Quarta ORELHA encontrada



Encontrar ORELHA do polígono até remanescer um único triângulo.

POL = 1 / 2 / 3 / 4 / 5 / 6 / 7 / 8

Defina duas listas com os vértices anteriores e posteriores:

L = 8 / 1 / 2 / 3 / 4 / 5 / 6 / 7

R = 2 / 3 / 4 / 5 / 6 / 7 / 8 / 1

Atualize a lista após o primeiro triângulo encontrado:

L = 8 / 1 / 2 / 2 / 4 / 5 / 6 / 7

R = 2 / 4 / 4 / 5 / 6 / 7 / 8 / 1

Atualize a lista após o segundo triângulo encontrado:

L = 8 / 1 / 2 / 2 / 2 / 5 / 6 / 7

R = 2 / 5 / 4 / 5 / 6 / 7 / 8 / 1

Atualize a lista após o terceiro triângulo encontrado:

L = 8 / 1 / 2 / 2 / 2 / 2 / 6 / 7

R = 2 / 6 / 4 / 5 / 6 / 7 / 8 / 1

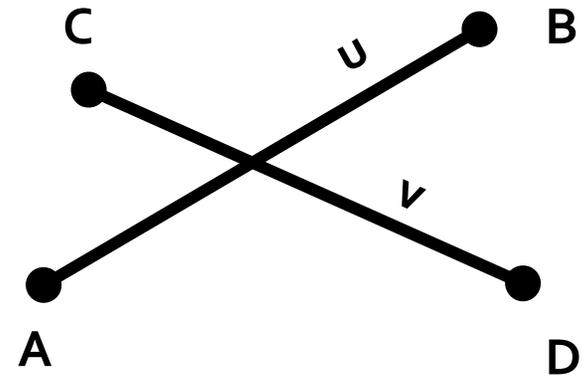
Atualize a lista após o quarto triângulo encontrado:

L = 8 / 1 / 2 / 2 / 2 / 1 / 6 / 7

R = 6 / 6 / 4 / 5 / 6 / 7 / 8 / 1

Algoritmo para Interseção de Segmentos de Reta

COMO TRATAR A INTERSEÇÃO DE SEGMENTOS DE RETA DE FORMA ROBUSTA E EFICIENTE?

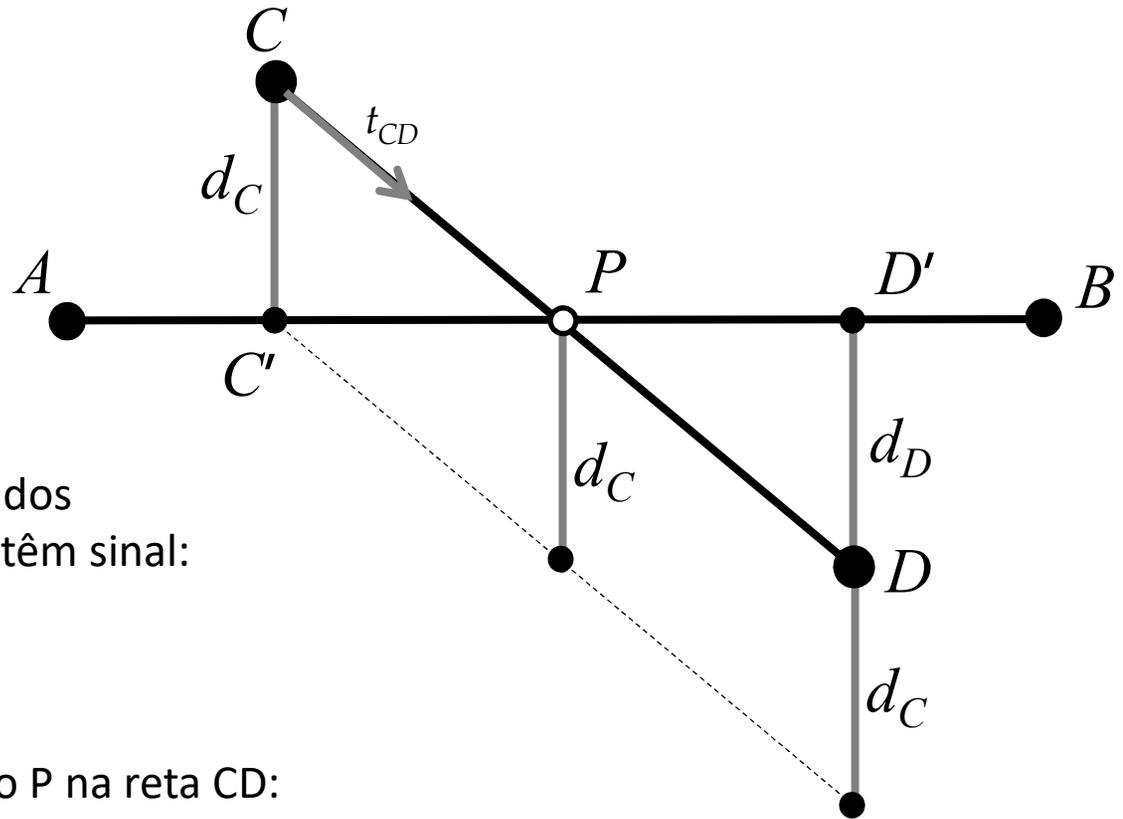


Fonte: Ricardo Marques

INTERSEÇÃO BASEADA NA REPRESENTAÇÃO PARAMÉTRICA DOS SEGMENTOS

$$P = A + t_{AB}(B - A)$$

$$P = C + t_{CD}(D - C)$$



Considere que as distâncias dos pontos C e D para a reta AB têm sinal:

$$d_C > 0$$

$$d_D < 0$$

Valor paramétrico do ponto P na reta CD:

$$t_{CD} = \frac{|d_C|}{|d_C| + |d_D|}$$

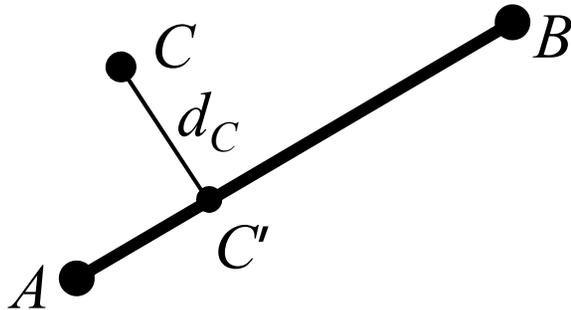
$$t_{CD} = \frac{d_C}{d_C - d_D}$$

$$0 \leq t_{CD} \leq 1$$

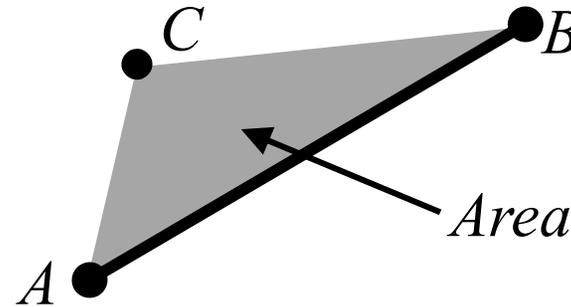
M. Gavrilova and J. G. Rokne. (2000) “Reliable line segment intersection testing”, CAD 32, 737–746

INTERSEÇÃO BASEADA NA REPRESENTAÇÃO PARAMÉTRICA DOS SEGMENTOS

Distância com sinal pode ser substituída por produto vetorial



$$Area = \frac{|\vec{AB}| \cdot d_C}{2}$$



$$Area = \frac{\vec{AB} \times \vec{AC}}{2}$$

$$Area = \frac{(B - A) \times (C - A)}{2}$$

Definição: (dobro da) área orientada do triângulo

$$orient2d(A, B, C) = (B - A) \times (C - A)$$

Portanto:

$$d_C = \frac{orient2d(A, B, C)}{|\vec{AB}|}$$

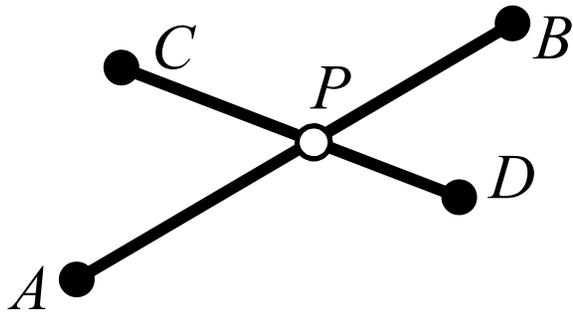
Analogamente:

$$d_D = \frac{orient2d(A, B, D)}{|\vec{AB}|}$$

Observe que os sinais das distâncias são resolvidos naturalmente.

INTERSEÇÃO BASEADA NA REPRESENTAÇÃO PARAMÉTRICA DOS SEGMENTOS

Valor paramétrico do ponto P na reta CD:



$$t_{CD} = \frac{d_C}{d_C - d_D}$$

$$d_C = \frac{\text{orient2d}(A, B, C)}{|\vec{AB}|}$$

$$d_D = \frac{\text{orient2d}(A, B, D)}{|\vec{AB}|}$$

Portanto:

$$t_{CD} = \frac{\text{orient2d}(A, B, C)}{\text{orient2d}(A, B, C) - \text{orient2d}(A, B, D)}$$

$$P = C + t_{CD}(D - C)$$

Analogamente:

$$t_{AB} = \frac{\text{orient2d}(C, D, A)}{\text{orient2d}(C, D, A) - \text{orient2d}(C, D, B)}$$

$$P = A + t_{AB}(B - A)$$

Segment_Segment_Intersection(u, v):

if both endpoints of **u** are over **v** **then**

return false $orient2d(C, D, A) > 0$ $orient2d(C, D, B) > 0$
end if

if both endpoints of **u** are under **v** **then**

return false $orient2d(C, D, A) < 0$ $orient2d(C, D, B) < 0$
end if

if both endpoints of **v** are over **u** **then**

return false $orient2d(A, B, C) > 0$ $orient2d(A, B, D) > 0$
end if

if both endpoints of **v** are under **u** **then**

return false $orient2d(A, B, C) < 0$ $orient2d(A, B, D) < 0$
end if

if **u** and **v** are collinear **then**

return false $orient2d(C, D, A) = 0$ $orient2d(C, D, B) = 0$
end if

ou $orient2d(A, B, C) = 0$ $orient2d(A, B, D) = 0$

~~**if** **u** and **v** are parallel **then** $orient2d(C, D, A) = orient2d(C, D, B)$~~

~~**return false** *ou* $orient2d(A, B, C) = orient2d(A, B, D)$~~
~~**end if**~~

if **u** touches **v** **then**

(there are many cases)

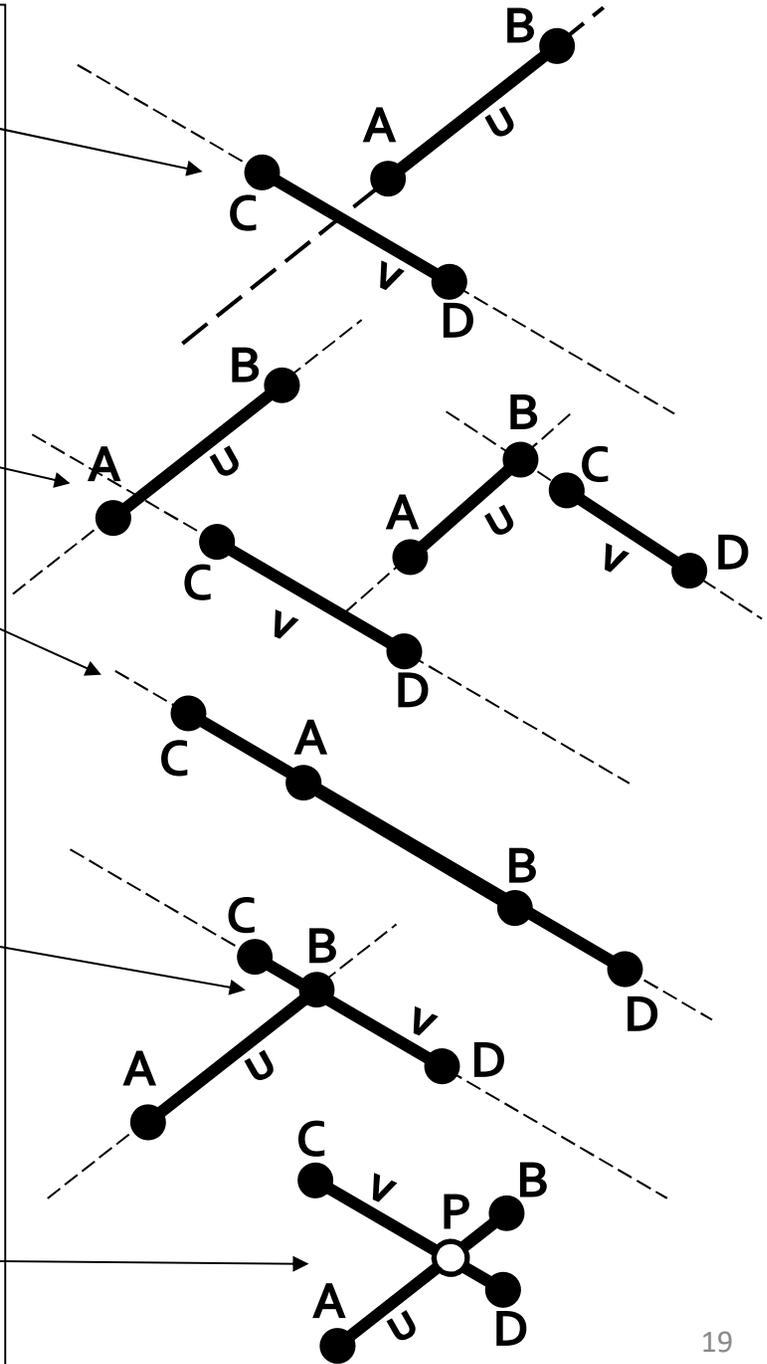
return true $orient2d(C, D, B) = 0$ $orient2d(C, D, A) < 0$
end if

$P = B$

// When get to this point, there is an intersection point

$$t_{CD} = \frac{orient2d(A, B, C)}{orient2d(A, B, C) - orient2d(A, B, D)}$$

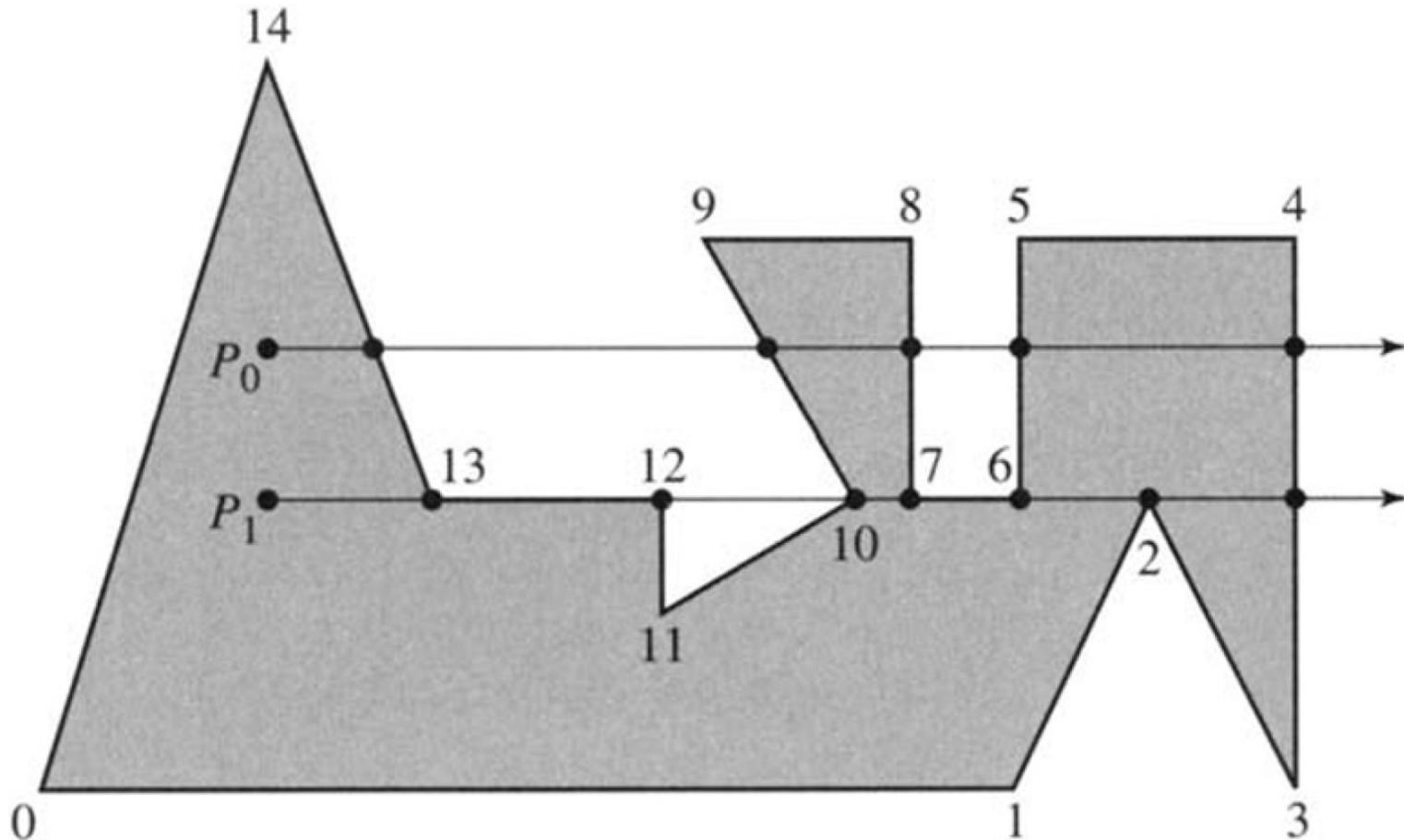
return true $P = C + t_{CD}(D - C)$



Algoritmo para verificação de ponto dentro de polígono

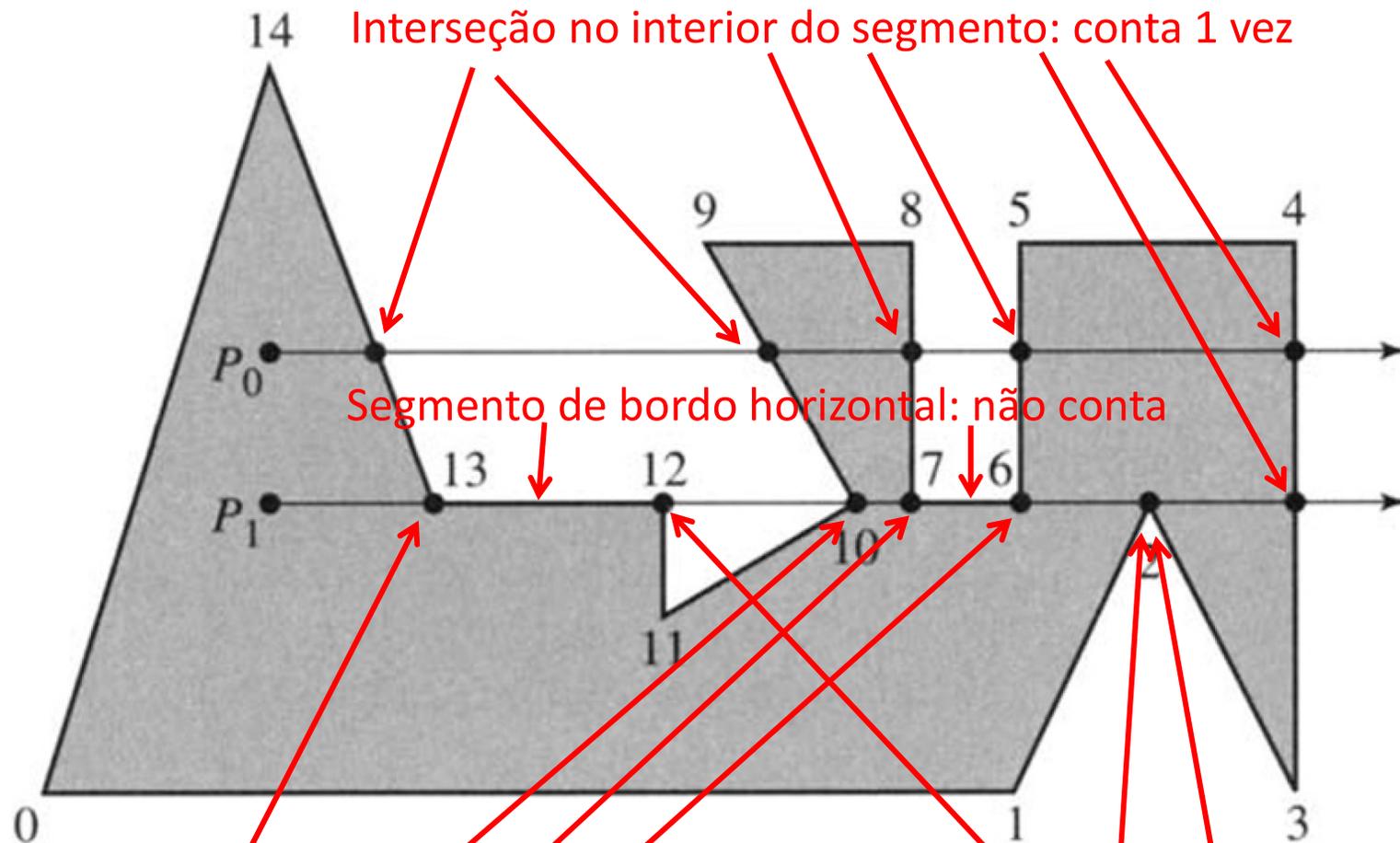
Algoritmo do raio (ou tiro)

Philip Schneider and David Eberly *Geometric Tools for Computer Graphics*, 2003, p.70



Uma semirreta (raio) que parte de qualquer ponto dentro de polígono em uma direção qualquer cortará as curvas no bordo do polígono um número ímpar de vezes. Se a semirreta cortar a fronteira do polígono um número par de vezes, o ponto está fora do polígono.

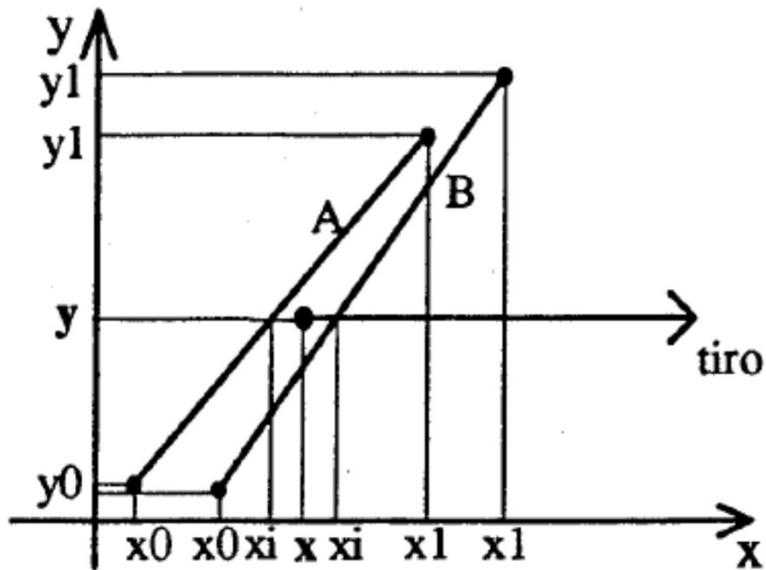
Critérios para contar interseções do raio com um segmento de bordo



Interseção no ponto inferior do segmento: conta 1 vez

Interseção no ponto superior do segmento: não conta

Tratamento dos casos não triviais:



$$x_i = x_0 + \frac{(y - y_0) \cdot (x_1 - x_0)}{(y_1 - y_0)}$$

$x_i \geq x \Rightarrow$ Interseção conta 1 vez

```

def isPointInPolygon(_poly, _p):
    x = _p.getX()
    y = _p.getY()
    n = len(_poly) # number of polygon points
    ni = 0 # number of intersections

    for i in range(0, n):
        p1 = _poly[i] # first point of current line segment
        p2 = _poly[(i+1) % n] # second point of current line segment

        if (p1.getY() == p2.getY()): # discard horizontal line
            continue

        if p1.getY() > y and p2.getY() > y: # discard line above ray
            continue

        if p1.getX() < x and p2.getX() < x: # discard line to the left of point
            continue

        if p1.getY() < y and p2.getY() < y: # discard line below ray
            continue

        if p1.getY() == y: # ray passes at first line point
            if p1.getX() > x and p2.getY() > y:
                # Count intersection if first point is to the right of given point
                # and second point is above.
                ni += 1
            else:
                if p2.getY() == y: # ray passes at second point
                    if p2.getX() > x and p1.getY() > y:
                        # Count intersection if first point is to the right of given point
                        # and second point is above.
                        ni += 1
                    else: # ray passes with first and second points
                        if p1.getX() > x and p2.getX() > x:
                            # Count intersection if first point is to the right of given point
                            # and second point is above.
                            ni += 1
                        else:
                            # Compute x coordinate of intersection of ray with line segment
                            dx = p1.getX() - p2.getX()
                            xc = p1.getX()

                            if dx != 0.0:
                                xc += (y - p1.getY())*dx / (p1.getY()-p2.getY())

                            if xc > x:
                                # Count intersection if first point is to the right of given point
                                # and second point is above.
                                ni += 1

    # If number of intersections is odd, point is inside polygon.
    if (ni % 2) > 0:
        return True

    # If number of intersections if even, point is outside polygon.
    return False

```