# PLANEJAMENTO DE CAMINHOS

## MÉTODOS BASEADOS EM MAPA DE ROTAS

**Princípio**: Capturar a conectividade de $C_L$ na forma de uma rede de curvas unidimensionais **R**, (**Mapa de Rotas**), que é usada como um conjunto de caminhos padrão. O planejamento se reduz a buscar um caminho em **R** entre $q_{ini}$ e $q_{fin}$. Exemplo: Grafo de Visibilidade, Diagrama de Voronoi, Rede de Caminhos Livres, Método da Silhueta, etc.

**Planejamento baseado em Grafo de Visibilidade**:

- Aplicação: $\mathbf{W} = \mathbf{R}^2$, robô **A** poligonal e com orientação fixa, com obstáculos $\mathbf{B}_i$ poligonais.
- Princípio: Construir um caminho semi-livre entre $q_{ini}$ e $q_{fin}$ formado por uma linha poligonal através dos vértices de **CB**.

$\Rightarrow$ Existe um caminho semi-livre entre $q_{ini}$ e $q_{fin}$ se e somente se existe uma linha poligonal simples, $\boldsymbol{\tau} \in \mathrm{cl}(\mathbf{C_L})$, cujos pontos extremos são $q_{ini}$ e $q_{fin}$ e tal que seus vértices $\in$ **CB**.

Prova: Se existe um caminho semi-livre, existe um caminho de comprimento euclidiano mínimo, $\boldsymbol{\tau}$, que, para ser o mais curto, deve ser também localmente mínimo. Assim, qualquer sub-caminho $\tau'$ de $\tau$ deve ser o mais curto entre os seus extremos, portanto, em qualquer configuração q, se q não é vértice de CB, $\tau$ deve ter curvatura zero $\Rightarrow$ os vértices de $\tau$ são vértices de CB.

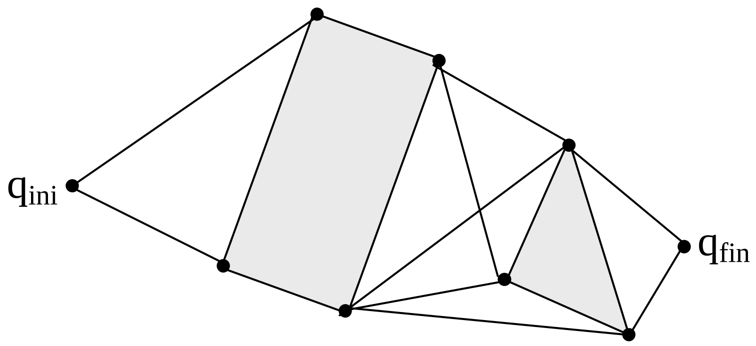
$\Rightarrow$ Para achar um caminho livre entre $q_{ini}$ e $q_{fin}$, é suficiente considerar o conjunto de linhas poligonais em $\mathrm{cl}(\mathbf{C_L})$ através dos vértices de **CB** (Grafo de Visibilidade).

$\Rightarrow$ O Grafo de Visibilidade contém o caminho mais curto, (em métrica euclidiana em $\mathbf{R}^2$), entre $q_{ini}$ e $q_{fin}$.

**Grafo de Visibilidade -** Grafo não direcional, **G**:

- Os Nós de G são $q_{ini}$ e $q_{fin}$ e os vértices de CB.
- Dois nós de G são conexos por um arco se e somente se o segmento que os une é um eixo de CB ou está contido inteiramente em $C_L$, com possível exceção dos seus extremos.

Exemplo:




**Algoritmo de Planejamento**:

1. Construir o Grafo G.
2. Buscar um caminho em G de $q_{ini}$ até $q_{fin}$.
3. Se o caminho é encontrado, retorná-lo, se não, reportar falha.

**Construção do Grafo de Visibilidade**:

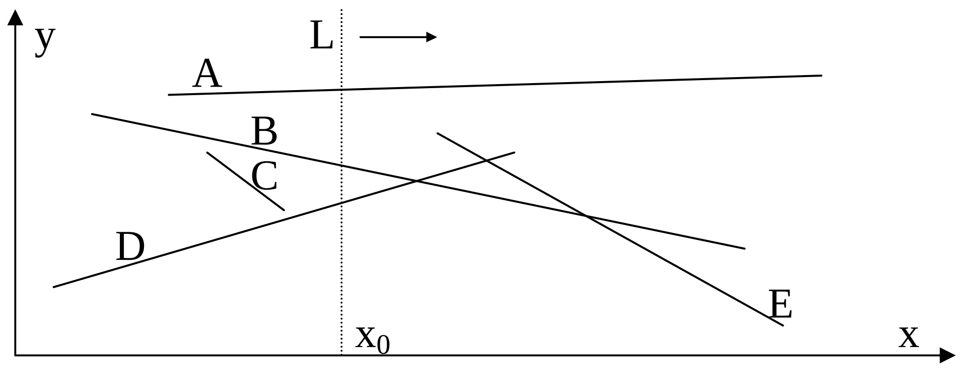
1. Tomar pares (X, X’) de nós de G.
2. Se X e X’ são extremos do mesmo eixo em CB, conectá-los por um arco em G.
3. Se X e X’ não são extremos do mesmo eixo em CB, computar as interseções da linha suporte de (X, X’) com CB. Caso não houver nenhuma interseção em (X, X’), ligar os nós por um arco em G.

Observação: complexidade computacional de ordem $O(n^3)$.

Algoritmo de Varredura de Linha - $O(n^2.\log(n))$:

1. Para cada ponto $X_i \in G$, computar a orientação $\alpha_{ij}$ da semi-reta com origem em $X_i$ e passando por cada um dos outros pontos $X_j \in G$.
2. Ordenar as orientações $\alpha_{ij}$.
3. Varrer as orientações $\alpha_{ij}$ no sentido anti-horário, de 0 a $2\pi$.
4. Para cada $\alpha_{ij}$, computar as interseções da semi-reta com CB.
5. Determinar o ponto de interseção P mais próximo a $X_i$.
6. Se P está contido no segmento $X_iX_j$, existe interseção, portanto não deve ser criado um arco entre os seus extremos; se não, o arco é criado.

Algoritmo de Varredura de Linha:




Dadas as seguinte estruturas de dados:

- *S* = Lista Status de **segmentos ativos** na abscissa corrente ordenados pelas ordenadas das interseções da reta de varredura L com os segmentos ativos. Se o segmento $X_1$ ocorre antes de $X_2$ em *S*, $X_1$ está *Abaixo* de $X_2$ e $X_2$ está *Acima* de $X_1$. Por exemplo, em $x = x_0$, $S = \{D, B, A\}$.
- *E* = Fila de eventos futuros, ordenados de acordo com as abscissas dos pontos extremos dos n segmentos ou dos c pontos de interseção.
- *L* = Lista ordenada por abscissas dos pontos de interseção.
- *Q* = Fila auxiliar usada internamente pelo algoritmo.

Dadas as seguinte operações sobre as estruturas de dados:

- INSERIR(X, *S*): insere o segmento X em *S*.
- APAGAR(X, *S*): apaga o segmento X de *S*.
- ACIMA(X, *S*): retorna segmento acima de X em *S*, se existir.
- ABAIXO(X, *S*): retorna segmento abaixo de X em *S*, se existir.
- TROCAR($X_1$, $X_2$, S): troca a ordem entre os segmentos consecutivos de *S*, $X_1$, $X_2$.
- PRIMEIRO(*E*): remove a menor abscissa de *E*, retornando-a.
- INSERIR(x, *E*): insere a abscissa x em *E*.
- MEMBRO(x, *E*): determina se a abscissa x é um membro de *E*.
- INSERIR(($X_1$, $X_2$), *Q*): insere par de segmentos ($X_1$, $X_2$) em *Q*.
- INSERIR(x, *L*): insere abscissa x em *L*.

Procedimento:

- A reta L varre os pontos críticos ($x_c$, $y_c$), (extremos ou interseções de segmento), da esquerda para a direita.
- A menor abscissa (próximo evento) em *E* é removida a cada iteração e *S* é atualizada de acordo com o seu tipo:
  - Se $x_c$ é abscissa do extremo esquerdo de um segmento X, X é adicionado a *S*.
  - Se $x_c$ é abscissa do extremo direito de um segmento X, X é removido de *S*.
  - Se $x_c$ é abscissa do ponto de interseção dos segmentos $X_1$ e $X_2$, $X_1$ e $X_2$ são trocados em *S*.
- Checa-se a interseção entre todo par de segmentos que se torna adjacente em *S*.
- Toda abscissa removida de *E* que corresponde a um ponto de interseção novo é inserida em *L*.

- No final, *L* conterá a lista ordenada das abscissas de todos os pontos de interseção.
- Complexidade computacional: *O*((n+c).log(n)).

```
Procedimento VARREDURA_DE_LINHA
começar
    colocar as abscissas dos 2.n pontos extremos dos n segmentos em E.
    S ← ∅; L ← ∅; Q ← ∅;
    enquanto E ≠ ∅, faça
    começar
            x ← PRIMEIRO(E);
            p ← ponto extremo ou de interseção do qual x é abscissa;
            se p é  um ponto extremo esquerdo, então
            começar
                    X ← segmento do qual p é extremo;
                    INSERIR(X, S);
                    X1 ← ACIMA(X, S);
                    X2 ← ABAIXO(X, S);
                    se X1 intersecta X, então INSERIR((X1, X), Q);
                    se X2 intersecta X, então INSERIR((X, X2), Q);
            fim;
            se não
            se p é  um ponto extremo direito, então
            começar
                    X ← segmento do qual p é extremo;
                    X1 ← ACIMA(X, S);
                    X2 ← ABAIXO(X, S);
                    se X1 intersecta X2, então INSERIR((X1, X2), Q);
                    APAGAR(X, S);
            fim;
            se não /* p é ponto de interseção */
            começar
                    INSERIR(x, L);
                    (X1, X2) ← par de segmentos se intersectando em p;
                    /* com X1 = ACIMA(X2) à esquerda de p */
                    X3 ← ACIMA(X1, S);
                    X4 ← ABAIXO(X2, S);
                    se X3 intersecta X2, então INSERIR((X3, X2), Q);
                    se X1 intersecta X4, então INSERIR((X1, X4), Q);
                    TROCAR(X1, X2, S);
            fim;
            enquanto Q ≠ ∅, faça
            começar
                    (X, X’) ← PRIMEIRO(Q);
                    x ← abscissa do ponto de interseção de X e X’;
                    se ¬MEMBRO(x, E), então INSERIR(x, E);
            fim;
    fim;
fim;
```

**Busca do Menor Caminho em G de $q_{ini}$ até $q_{fin}$:**

- Técnicas de busca em grafos. Exemplo: Algoritmo $A^*$, usando a distância euclidiana ao alvo como função heurística para guiar a busca.
- Grafo G = (X, A), onde X é o conjunto de n nós e A é o conjunto de r arcos.
- G representado por listas de adjacências, uma para cada nó N (lista de todos os nós N' ligados por um arco a N).

<u>Algoritmo A* *O*(r.log(n))</u>:

- Aplicável a grafos em que os arcos têm custos associados, por exemplo, distância euclidiana entre os nós.
- Custo de um caminho = soma dos custos dos seus arcos.
- Permite obter o menor caminho em termos da métrica adotada.

Procedimento:

- G explorado iterativamente seguindo caminhos a partir de $N_{ini}$.
- Para cada nó visitado, um ou mais caminhos são gerados a partir de $N_{ini}$, mas só o de menor custo é memorizado.
- O conjunto de caminhos gerados forma uma árvore T do subconjunto de G já explorado.
- T é representada através de ponteiros, associados a cada nó visitado, os quais apontam para os correspondentes nós pais.
- Para cada nó N, atribui-se uma função de custo, estimativa do custo do caminho de custo mínimo passando por N:

  $$f(N) = g(N) + h(N)$$

- g(N) = custo do caminho entre $N_{ini}$ e N em T corrente.
- h(N) = estimativa heurística do custo $h^*(N)$ do caminho de mínimo custo entre N e $N_{fin}$.

- h(N) é <u>admissível</u> se e somente se: $\forall N \in G: 0 \leq h(N) \leq h^*(N)$.
- Se h(N) é admissível, $A^*$ garante encontrar o caminho de menor custo entre quaisquer dois nós conexos por G.
- Função heurística admissível trivial (Dijkstra): $h_1(N) = 0$.
- Função heurística admissível, aplicável quando os nós representam pontos em ⌡$^n$: $\|N_{fin} - N\|$.
- $h_2$ é <u>mais informada</u> que $h_1 \Rightarrow \forall N \in G: h_1(N) \leq h_2(N)$.
- Se $h_2$ é mais informada que $h_1$, todo nó expandido por $A^*$ usando $h_2$ pode ser expandido por $A^*$ usando $h_1$.
- h é <u>consistente localmente</u> se: $0 \leq h(N') \leq h(N) + k(N, N')$, onde k(N, N') é o custo associado ao arco entre N e N'.
- Se h é consistente localmente, é também admissível.
- $h_1$ e $h_2$ são consistentes localmente.

Dadas as seguinte estruturas de dados e funções:

- G(X,A) = Grafo com n nós $\in$ X e r arcos $\in$ A, com conectividade representada por listas de adjacências entre nós.
- T = Árvore do subconjunto de G já visitado.
- L = Lista que armazena nós de G ordenados por f(N).
- $k : X \times X \rightarrow$ ⌡$^+$ função que especifica o custo de cada arco.
- h(N): estimativa do custo do caminho mínimo entre N e $N_{fin}$.
- g(N) = custo do caminho entre $N_{ini}$ e N em T corrente.

Dadas as seguinte operações sobre a lista L:

- PRIMEIRO(L): retorna o nó com menor valor de f(N) em L e o remove da lista.
- INSERIR(N,L): insere o nó N na lista L.
- APAGAR(N,L): apaga o nó N da lista L.
- MEMBRO(N,A): determina se o nó N é membro da lista L.
- VAZIA(L): determina se a lista L está vazia.

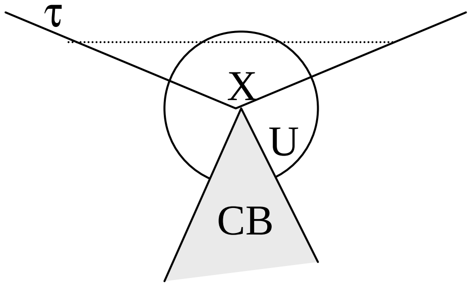
```
**Procedimento** A*(G, N_ini, N_fin, k, h);
```

**Procedimento** $A^*(G, N_{ini}, N_{fin}, k, h)$;
**começar**
    $N_{ini}$ em T;
    INSERIR($N_{ini}$, L); marcar $N_{ini}$ como visitado;
    **enquanto** ¬VAZIA(L), **faça**
    **começar**
        N ← PRIMEIRO(L);
        **se** $N = N_{fin}$, **então** sair do laço while;
        **para** cada nó N' adjacente a N em G, **faça**
            **se** N' é não visitado, **então**
            **começar**
                adicionar N' a T com um ponteiro para N;
                INSERIR(N',L); marcar N' como visitado;
            **fim**
            **se não, se** $g(N') > g(N) + k(N, N')$, **então**
            **começar**
                redirecionar o ponteiro de N' para N em T;
                **se** MEMBRO(N',L),**então** APAGAR(N',L);
                INSERIR(N',L);
            **fim**
    **fim;**
    **se** ¬VAZIA(L), **então**
        retornar o caminho traçando os ponteiros de $N_{fin}$ a $N_{ini}$;
    **se não** reportar falha;
**fim;**

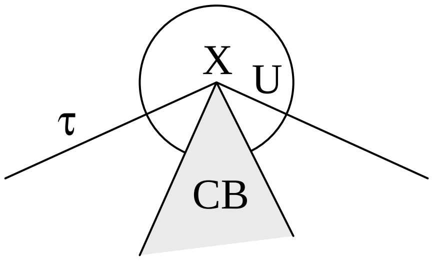
**Grafo de Visibilidade Reduzido**

- Setor Convexo no vértice X de um caminho poligonal $\tau$ é a região convexa fechada entre as duas semi-retas originadas em X e que contém os segmentos de $\tau$ adjacentes a X.

$\Rightarrow$ para $\tau$ ser o caminho mais curto $\Leftrightarrow \forall\ X \in \tau$, X deve possuir uma vizinhança U tal que $(CB \cap U) \subset$ setor convexo de $\tau$ em X.

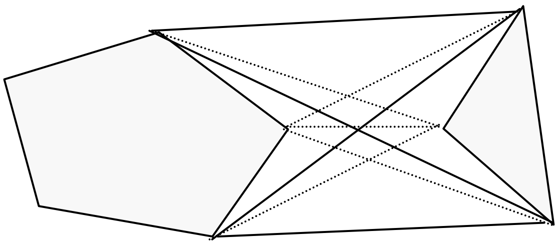


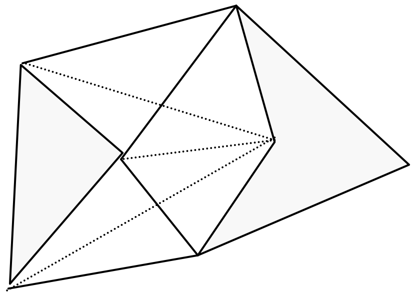

a) Pode ser minimizado.




b) Não pode ser minimizado.

- Uma reta L passando em um vértice X de CB é tangente a CB em X se e somente se o interior de CB está inteiramente em um único lado de L para uma vizinhança U de X.
- Seja L uma reta passando por dois nós X e X' de G. O segmento de L, XX' é um Segmento Tangente se e somente se:
  - se X é vértice de CB, então L é tangente a CB em X;
  - se X' é vértice de CB, então L é tangente a CB em X'.
- Segmentos tangentes possuem as seguintes propriedades:
  - Entre dois polígonos convexos disjuntos existem exatamente quatro segmentos tangentes: dois são suportes (os dois polígonos ficam do mesmo lado), os outros dois são separadores (os dois polígonos ficam em lados opostos).
  - Se o vértice X é côncavo, (ângulo interno em X maior do que $\pi$), nenhum segmento tangente termina em X.





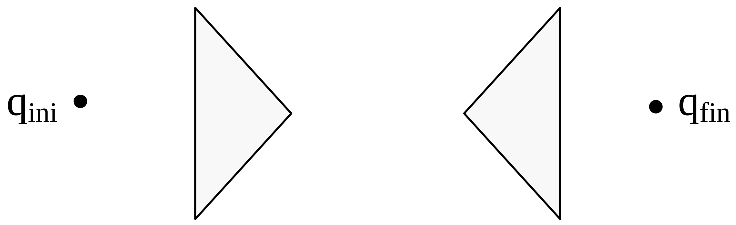
⇒ Alguns arcos do grafo de visibilidade não são necessários.

Procedimento de construção do Grafo de Visibilidade Reduzido:

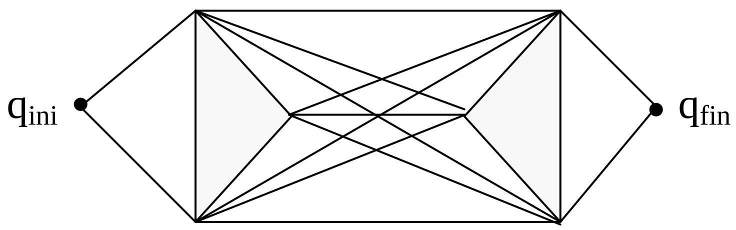
- Se o segmento entre X e X' não é segmento tangente, não precisa ser incluído no grafo.

Observação: qualquer caminho $\tau$ que inclui um segmento não tangente entre X e X' é tal que não está situado localmente no setor convexo de $\tau$ em X e/ou de X'.

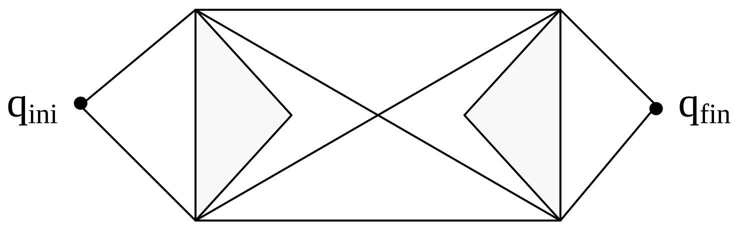
- O subgrafo G' obtido pela remoção de todos os segmentos não tangentes é denominado de Grafo de Visibilidade Reduzido.
- Se existe um caminho semi-livre entre $q_{ini}$ e $q_{fin}$, G' contém o menor caminho em $cl(C_L)$ entre estas duas configurações.




a) Espaço de Configuração.



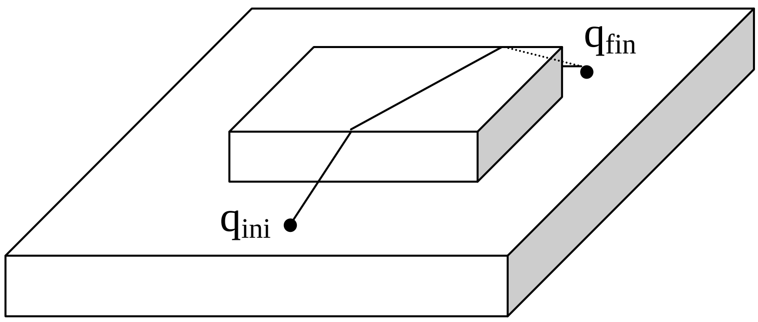

b) Grafo de Visibilidade.




c) Grafo de Visibilidade Reduzido.

## **Grafo de Visibilidade para espaços de dimensão maior**:

- O Problema de gerar o menor caminho semi-livre em um espaço C = $\mathbf{R}^3$ povoado de C-obstáculos poliédricos é NP.
- Para esta situação, o método do grafo de visibilidade pode ser aplicado, mas sem garantir o caminho mínimo.
- Somente se pode garantir que o caminho mínimo será uma linha poligonal cujos vértices estão contidos nos eixos dos C-Obstáculos.
- Caminhos mínimos podem ser obtidos adicionando vértices fictícios nos eixos dos poliedros.




- O método Grafo de Visibilidade não é estendido diretamente ao caso em que o robô é um polígono capaz de transladar-se e girar em um espaço povoado de obstáculos poligonais.
- Neste caso, uma possível maneira de tratar a rotação consiste em "fatiar" o espaço de configuração em um número finito p de intervalos $[\theta_k, \theta_{k+1}]$ e computar a projeção $CB_k$ de CB dentro deste intervalo em ⌡$^2$.
- $CB_k$ é um polígono generalizado, ao qual pode ser associado um grafo de visibilidade $G_k$.
- Os p grafos $G_i$'s podem ser combinados em um grafo de visibilidade global ligando por um arco cada nó $X \in G_k$ a cada nó $X' \in G_{k+1}$, (k=1,...,p mod p), sempre que o segmento de aberto ligando as duas configurações não intersecta $CB_k$ nem $CB_{k+1}$. Este método não completo: pode falhar em achar um caminho, mesmo se existir um.

## <u>Planejamento baseado em Retração</u>:

- Princípio: Definir um mapeamento contínuo de $C_L$ numa rede de curvas unidimensionais **R** contida em $C_L$, a qual preserva a conectividade do mesmo.
- Seja **X** um espaço topológico e **Y** um subconjunto de **X**. o mapeamento sobrejetor $\rho$: **X** $\rightarrow$ **Y** é chamado de <u>Retração</u> de **X** em **Y** $\Leftrightarrow$ $\rho$ é contínuo e sua restrição a **Y** é o mapeamento identidade.
- Seja $\rho$ a retração de **X** em **Y**. $\rho$ <u>preserva a conectividade</u> de **X** $\Leftrightarrow \forall x \in$ **X**, x e $\rho(x)$ pertencem à mesma componente conexa de **X**, ou seja, existe caminho entre x e $\rho(x)$.
- <u>Proposição</u>: Seja $\rho$: $C_L \rightarrow$ **R** uma retração preservadora da conectividade de $C_L$ na rede de curvas unidimensionais **R** $\subset C_L$. Existe um caminho livre entre $q_{ini}$ e $q_{fin}$ $\Leftrightarrow$ existe um caminho em **R** entre $\rho(q_{ini})$ e $\rho(q_{fin})$.

<u>Prova</u>: seja $\tau : [0, 1] \rightarrow C_L$ um caminho livre entre $q_{ini}$ e $q_{fin}$. O mapeamento $\rho(\tau) : [0,1] \rightarrow$ **R** é um caminho livre entre $\rho(q_{ini})$ e $\rho(q_{fin})$, visto que $\rho$ é contínuo. Por outro lado, se existe um caminho em **R** entre $\rho(q_{ini})$ e $\rho(q_{fin})$, então, $q_{ini}$ e $q_{fin}$ são conexas por um caminho livre que é a composição de três caminhos: um caminho livre de $q_{ini}$ a $\rho(q_{ini})$, um caminho em **R** entre $\rho(q_{ini})$ e $\rho(q_{fin})$ e um caminho livre de $\rho(q_{fin})$ a $q_{fin}$. O primeiro e o terceiro caminho existem, visto que $\rho$ preserva a conectividade de $C_L$.

- Método de planejamento baseado em retração:
  - Escolher o mapeamento $\rho$ e construir **R**.
  - Construir um grafo **G** que represente **R**.
  - Determinar os mapeamentos $\rho(q_{ini})$ e $\rho(q_{fin})$.
  - Gerar um caminho livre entre $q_{ini}$ e $\rho(q_{ini})$.
  - Buscar em **G** um caminho entre $\rho(q_{ini})$ e $\rho(q_{fin})$.
  - Gerar um caminho livre entre $\rho(q_{fin})$ e $q_{fin}$.

**<u>Planejamento baseado em Diagrama de Voronoi</u>**:

- Aplicável a $C = \mathbf{R}^2$ e $C_L$ = interior de uma região poligonal limitada.
- Propriedade: maximiza a distância do robô aos obstáculos.

<u>Definições</u>:

- Seja $\beta = \partial C_L$. $\forall q \in C_L$, <u>Claridade</u> de q = clar(q) = $\min_{p\in\beta}\|q-p\|$

- <u>Perto</u> de q = perto(q) = $\{p \in \beta \;/\; \|q-p\| = \text{clar}(q)\}$

- <u>Diagrama de Voronoi de $C_L$</u>:

  $\text{Vor}(C_L) = \{q \in C_L \;/\; \text{card}(\text{perto}(q))>1\}$

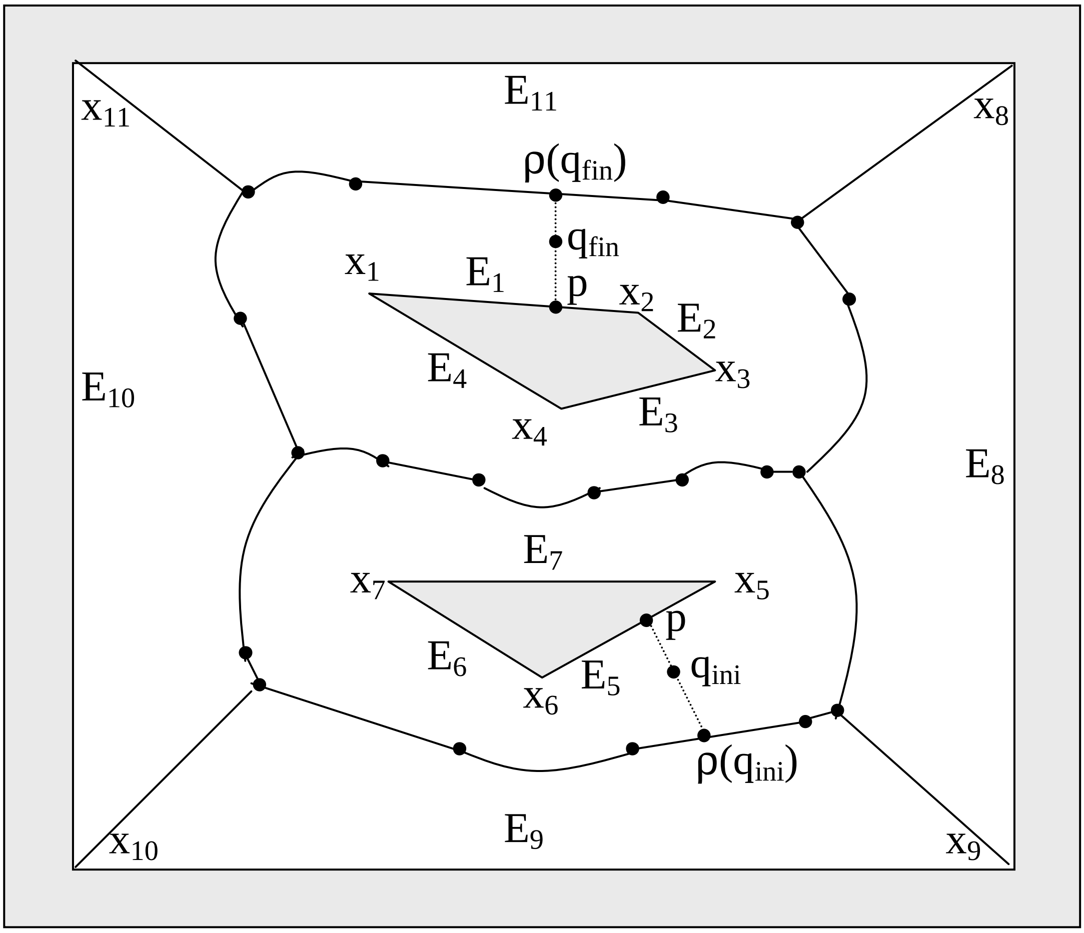
Onde card(X) retorna a cardinalidade do conjunto X.

⇒ O Diagrama de Voronoi é o mapa de rotas **R** usado em método de planejamento baseado em retração.

- $\text{Vor}(C_L)$ = conjunto de segmentos de reta e parabólicos:
  - Segmentos de reta entre (eixo, eixo) ou (vértice, vértice).
  - Segmentos parabólicos entre (eixo, vértice).

⇒ Estes segmentos e os eixos de $\partial C_L$ delimitam regiões de $C_L$ para as quais card(perto(q)) = 1.

- Dados $q \notin \text{Vor}(C_L)$ e $p \in \partial C_L$, tal que $\|q-p\| = \text{clar}(q)$. Considere a semi- reta L com origem em p e passando por q. O segmento de reta conectando p à interseção mais próxima de L com $\text{Vor}(C_L)$, $\rho(q)$, (imagem de q em $\text{Vor}(C_L)$), segue o gradiente positivo de clar(q). Além de $\rho(q)$, o gradiente muda de sinal. Para $q = q_{ini}$ (ou $q = q_{fin}$), este procedimento permite obter as suas imagens em $\text{Vor}(C_L)$, $\rho(q_{ini})$ (ou $\rho(q_{fin})$).




Dado que:

- $\forall q \in Vor(C_L) \Rightarrow \rho(q) = q$ (mapeamento identidade)
- $\rho : C_L \rightarrow Vor(C_L)$ é mapeamento contínuo.

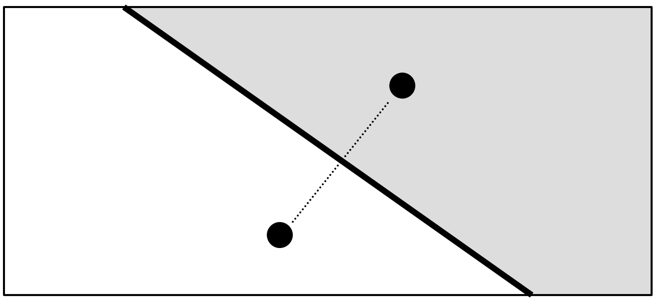
$\Rightarrow$ ρ é uma retração preservadora da conectividade.

Planejamento baseado no mapa de rotas definido por $Vor(C_L)$:

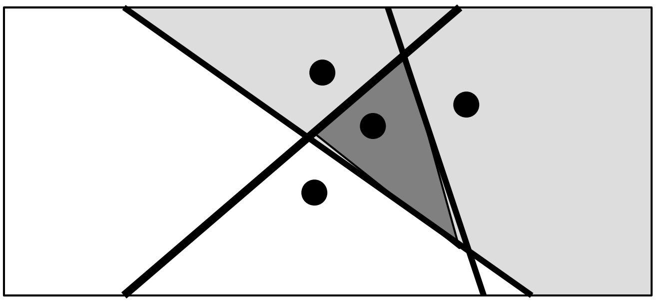
1. Computar $Vor(C_L)$ $\Rightarrow$ uma vez computado $Vor(C_L)$ pode ser utilizado para quaisquer $q_{ini}$ e $q_{fin}$.
2. Computar $\rho(q_{ini})$ e $\rho(q_{fin})$ $\rightarrow$ identificar os arcos de $Vor(C_L)$ contendo estes pontos.
3. Buscar uma seqüência de arcos $A_1, \ldots, A_p$ em $Vor(C_L)$, tal que $\rho(q_{ini}) \in A_1$ e $\rho(q_{fin}) \in A_p$ e, para todo $i \in [1, p-1]$, $A_i$ e $A_{i+1}$ compartilham uma extremidade.
4. Se a busca tem sucesso, retornar $\rho(q_{ini})$, $\rho(q_{fin})$ e a seqüência de arcos $A_1, \ldots, A_p$, se não, reportar falha.

**Diagrama de Voronoi de um conjunto de pontos (Sítios)**:

- Aplicável em um espaço de trabalho bidimensional povoado de obstáculos pontuais.
- Um conjunto de n sítios P = {$p_1$, $p_2$, ..., $p_n$} no plano gera um diagrama de Voronoi associado.
- Cada par ($p_i$, $p_j$) pode ser separado por uma reta eqüidistante aos mesmos e perpendicular ao segmento que os une.
- A reta divide o plano em dois semiplanos, cada um contendo um dos pontos do par.

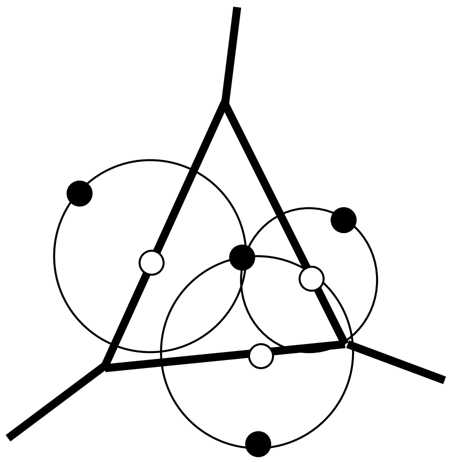


- A interseção dos semiplanos gera uma célula poligonal convexa (eventualmente ilimitada).
- Célula de Voronoi V($p_i$) de $p_i$ é o conjunto de pontos que estão mais próximos de $p_i$ do que de qualquer outro sítio de P.
- A célula de Voronoi associada a um ponto $p_i$ é definida pela interseção dos semiplanos que contêm $p_i$, gerados por todos os pares de pontos que incluem $p_i$.

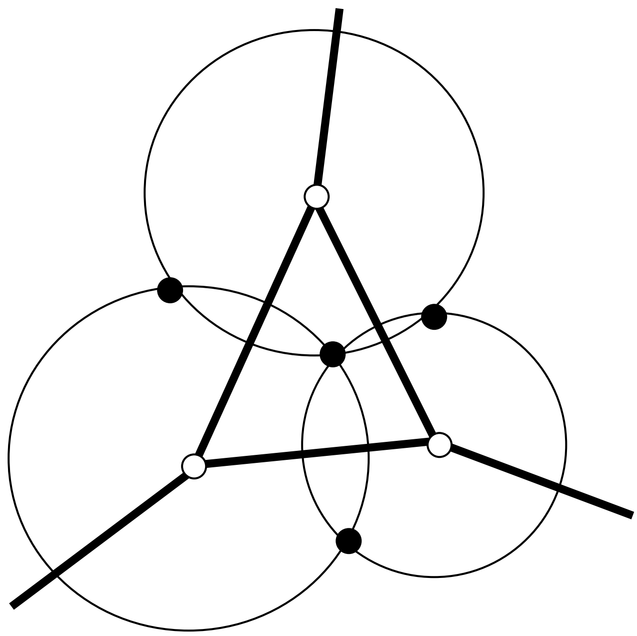


- O Diagrama de Voronoi Vor(P) é definido pelos limites das células de Voronoi.

- Cada par de sítios ($p_i$, $p_j$) pertencentes a células adjacentes é separado por um segmento de reta denominado **Arestas de Voronoi**.
- Cada ponto pertencente à Aresta de Voronoi é centro de um círculo passando por $p_i$ e por $p_j$.



- Cada tripla de sítios ($p_i$, $p_j$, $p_k$) pertencentes a células adjacentes entre si duas a duas define três arestas que se intersectam em um único ponto denominado **Vértice de Voronoi**.
- O Vértice de Voronoi é o centro de um círculo que passa pelos três sítios.



- Algoritmo Trivial para cômputo do diagrama de Voronoi:

1. Para cada tripla de sítios ($p_i$, $p_j$, $p_k$) em P computar o ortocentro (o ponto eqüidistante aos três sítios).
2. Para cada ortocentro x, checar todos os pontos em P de modo a verificar se ($p_i$, $p_j$, $p_k$) são os mais próximos de x.

   - Se existe algum sítio mais próximo de x do que ($p_i$, $p_j$, $p_k$), rejeitar o ortocentro x.

3. Para cada ortocentro x remanescente gerado por uma tripla de sítios ($p_i$, $p_j$, $p_k$), buscar em P os ortocentros $y_{ij}$, $y_{jk}$, $y_{ki}$ que compartilham com x os pares de sítios geradores ($p_i$, $p_j$), ($p_j$, $p_k$) e ($p_k$, $p_i$), respectivamente.

   - Para cada ortocentro y encontrado associado a um ortocentro x definir a Aresta de Voronoi cujos extremos são x e y.
   - Caso não exista um ortocentro y associado a x para um dado par de seus sítios geradores ($p_i$, $p_j$), definir uma Aresta de Voronoi infinita constituída pela semi-reta com origem em x e perpendicular ao segmento entre $p_i$ e $p_j$.

Exemplo:

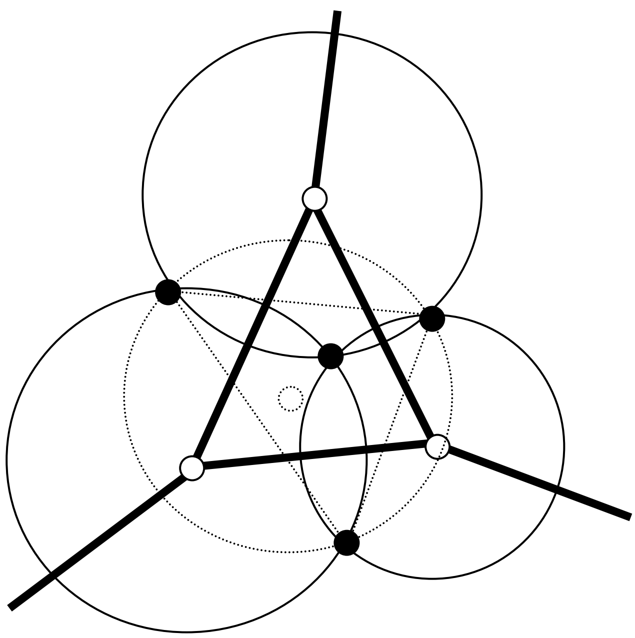


**Diagrama de Voronoi de um conjunto de Polígonos**:

- Aplicável em um espaço bidimensional povoado de obstáculos poligonais.

- **Algoritmo Aproximado**:

1. Aproximar os limites dos obstáculos poligonais com um grande número de pontos, resultantes da subdivisão dos lados dos polígonos em pequenos segmentos.
2. Computar o Diagrama de Voronoi desta coleção de pontos.
3. Eliminar do Diagrama de Voronoi resultante as arestas que possuem ao menos um dos seus extremos no interior de qualquer obstáculo.
4. Conectar as configurações inicial e final do robô ao diagrama através de segmentos de reta entre as mesmas e os Vértices de Voronoi mais próximos correspondentes. (Verificar eventuais interseções destes segmentos com os obstáculos).

**Algoritmo Exato (Trivial)**:

1. Para cada par (eixo,eixo), (vértice,vértice) e (eixo,vértice) da região de C-Obstáculos, computar as curva eqüidistante correspondente (reta ou parábola).

2. Computar as interseções entre estas curvas.

3. Considerar apenas os segmentos de curvas cujas extremidades são definidas pelas interseções mais próximas dos seus elementos geradores (eixos ou vértices).

Observação: algoritmos ótimos baseados em varredura do plano, (derivados do algoritmo de Steven Fortune, [Fortune, S. J.. "A sweepline algorithm for Voronoi diagrams." Algorithmica, 3 (1987) : 153-174.]) computam o Diagrama de Voronoi com complexidade O(n.log(n)).