UM ALGORITMO PARA REMOÇÃO DE LINHAS OCULTAS

Luiz Cristovão Gomes Coelho[1]
Marcelo Gattass[2]
Waldemar Celes Filho[1]
Depto. Eng. Civil, PUC-Rio, 22453 Rio de Janeiro

RESUMO - Apresenta-se um novo algoritmo para remoção de linhas ocultas de modelos tridimensionais, representados por linhas e faces planas. O algoritmo aceita faces com número qualquer de vértices e permite que as linhas perfurem as faces.

1 - INTRODUÇÃO

A idéia do algoritmo nasceu da necessidade de representar modelos tridimensionais de plataformas semisubmersíveis projetadas pelo CENPES/PETROBRAS. Estes modelos, por possuirem faces muito alongadas e arestas posicionadas arbitrariamente, se mostraram bastante difíceis de serem tratados pelos algoritmos disponíveis na literatura ([JANS83], [SPIL86]).

É proposto um algoritmo que trata faces planas (convexas ou côncavas), com um número qualquer de vértices, abordando ainda os casos em que linhas perfurem estas faces.

A idéia do algoritmo consiste no cálculo da interseção dos trechos visíveis de cada linha contra todas as faces.

2 - CONCEPÇÃO BÁSICA

O algoritmo proposto remove (ou traceja) as linhas ocultas de um modelo. Partindo de uma lista de arestas, $A = \{a_1, a_2, a_3, \ldots, a_n\}$ que descrevem as linhas, e de uma lista de

---

1 Mestrando

2 Prof. Associado

faces, $F = \{f_1, f_2, f_3, \ldots f_m\}$, não necessariamente relacionadas entre si, o algoritmo testa cada aresta contra todas as faces da lista, sendo portanto um procedimento de ordem n × m.

A concepção básica do algoritmo é a mesma de todos os algoritmos para remoção de linhas ocultas, e consiste na determinação dos trechos visíveis de cada aresta contra todas as faces do modelo. Para o cálculo dos trechos visíveis, utiliza-se a equação paramétrica (1) do segmento de reta definida pelos vértices inicial e final da aresta, assim:

$$x = x_{vi} + t\,(x_{vf} - x_{vi}) \qquad (1)$$
$$y = y_{vi} + t\,(y_{vf} - y_{vi})$$

onde: $0 \leq t \leq 1$,
vi e vf são os vértices inicial e final da aresta

Os trechos visíveis da aresta são armazenados em um vetor, denominado **vetor de visualização**, por meio dos valores do parâmetro t. Assim sendo, registram-se nas posições ímpares do vetor de visualização os pontos de início dos trechos visíveis (*start points*), reservando as posições pares para os pontos de término (*end points*) ({ SP,EP,SP,EP,... }).

O algoritmo trabalha com dois vetores de visualização, um global e outro local. Para cada face determina-se o vetor de visualização local para a aresta em questão. Após o cálculo do vetor local, atualiza-se o vetor de visualização global por meio da interseção dos trechos visíveis, sendo que o vetor global é inicializado considerando-se toda a aresta como visível.

O trecho em pseudo-código a seguir esclarece a lógica do algoritmo.

```
/* determinação do vetor de visualização */
Para cada aresta                    /* loop das arestas */
{
    Vg = { 0.0, 1.0 }               /* inicializa vetor global */
    Para cada face                  /* loop das faces */
    {
        Determina Vl                /* determina vetor local */
        Atualiza Vg                 /* atualiza vetor global */
    }
    Desenha Vg                      /* desenha os trechos visíveis */
}
```

Após percorrer todas as faces, obtém-se o vetor de visualização final que é então desenhado. Se durante o processo o vetor se anular, significando que aresta foi totalmente escondida, o processo é então interrompido, passando-se à análise da próxima aresta.

3 - DETERMINAÇAO DOS TRECHOS VISÍVEIS

O cálculo dos trechos visíveis de uma aresta contra uma determinada face do modelo (cálculo do vetor de visualização local) é dividido em seis etapas. A seguir cada etapa é apresentada separadamente:

Passo 0: "Pick Face" e rotação para a horizontal

A determinação dos pontos de interseção da aresta com relação à face é feita com base no algoritmo que verifica se um ponto pertence ou não a uma face (*pick face*), implementado no GKS/puc. A resposta deste algoritmo está no número de interseções que uma semi-reta com origem no ponto em questão tem com as arestas da face. Se o número de interseções for ímpar significa que o ponto é interno à face, caso contrário é externo.

Como a direção do "tiro" é indiferente, escolheu-se a direção do eixo positivo de X para facilitar o cálculo das interseções (figura 1).

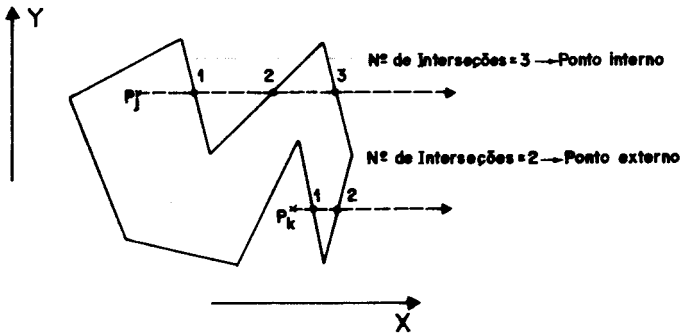


Fig. 1 - Algoritmo de *pick*.

Para que a projeção de uma aresta fique na horizontal, coincidindo com a direção do "tiro", deve-se executar uma rotação na aresta e na face em questão de maneira que a linha se torne horizontal com o vértice inicial à esquerda, como ilustrado na figura 2.

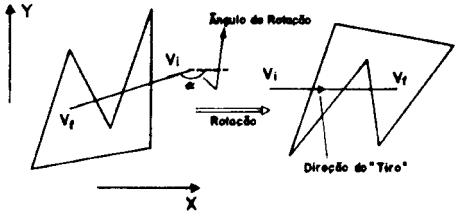


Fig. 2 - Rotação do conjunto aresta e face.

Passo 1: Cálculo das interseções intermediárias

Nesta etapa, é realizado o cálculo dos pontos de interseção intermediários da aresta contra a face. O cálculo das interseções é feito em função do parâmetro t da equação (1).

Deve-se observar que todo o procedimento é feito no sistema XYZ, destrógiro, oriundo da transformação de uma projeção qualquer em uma projeção ortográfiga no plano XY. Note-se que objetos com maior coordenada Z obscurecem os demais.

O cálculo das interseções, no entanto, é feito no plano XY, havendo necessidade de uma análise tridimensional. De maneira geral, chama-se ponto de interseção para o algoritmo, os pontos onde a aresta passa a ser obscurecida pela face (ponto de término) ou onde a aresta volta ou passa a ser visível em relação a face (ponto de início). Em ambas as situações, é necessário que a face esteja à frente da aresta, isto é, a coordenada Z da face deve ser maior que a coordenada Z da aresta no ponto de interseção, tendo como caso particular a situação em que o Z da face é igual ao Z da aresta.

Portanto, um ponto de interseção no espaço bidimensional só pode ser considerado ponto de interseção para o algoritmo proposto se $Z_{face} > Z_{aresta}$ no ponto em quetão. Caso $Z_{face} = Z_{aresta}$, armazena-se o valor paramétrico da interseção numa variável auxiliar (tzero), não incluindo-o no vetor de visualização.

Deve-se observar ainda, que como o algoritmo só trabalha com faces planas e arestas (linhas retas), só pode ocorrer uma única situação ao longo da aresta em que as coordenadas Z são coincidentes. Se ocorrer esta situação mais de uma vez, significa que face e aresta pertencem ao mesmo plano e neste caso a face não pode obscurecer a aresta.

Passo 2: Análise do vértice inicial

Nesta etapa do procedimento, é verificado se o ponto inicial da aresta (t=Ø.Ø) deve ou não ser incluído no vetor de visualização.

O vértice inicial só pode ser considerado ponto de interseção se ele for um ponto de início de trecho visível. Para que isto ocorra, há duas possibilidades:

a) O vértice inicial é um ponto externo à face;
b) O vértice inicial é um ponto interno à face mas $Z_{vi} > Z_{face}$, isto é, a face está atrás do vértice inicial.

Para a determinação dessas situações utiliza-se o algoritmo de *pick face* descrito no Passo Ø deste procedimento. Uma situação particular que requer uma outra verificação é quando o vértice está sobre o contorno da face, neste caso ele pode vir a ser considerado um ponto externo ou interno, dependendo da situação. Para tanto, calcula-se o código (INTERNO-CONTORNO-EXTERNO) do ponto médio definido pelo vértice em questão e o ponto de interseção mais próximo. Caso o número de interseções seja nulo, o ponto médio é definido como sendo o ponto médio da aresta. Se o código do ponto médio for INTERNO ou CONTORNO indica que o vértice deve ser considerado como ponto interno à face, caso contrário como ponto externo. A figura 3 ilustra essas situações.

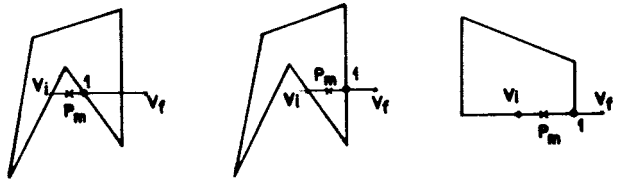


Fig. 3 - Análise do ponto médio.

Observa-se também que, análogo ao que foi feito no Passo 1, caso a coordenada Z do vértice seja coincidente à coordenada Z do plano definido pela face, armazena-se o valor zero (t=Ø.Ø) na variável auxiliar (tzero), não incluindo-o no vetor de visualização.

Passo 3: Análise do vértice final

Esta etapa é análoga à anterior, sendo que agora faz-se a verificação do vértice final.

Para que o vértice final (t=1.Ø) seja incluído no vetor de visualização é necessário que ele seja um ponto de término, havendo duas situações em que isso ocorre:

a) O vértice final é um ponto externo à face;
b) O vértice final é um ponto interno à face mas $Z_{vf} > Z_{face}$, isto é, a face está atrás do vértice final.

Passo 4: Casos particulares

Deve-se notar que o número de pontos armazenados no vetor de visualização tem que ser par, pois se este vetor define os trechos visíveis da aresta, para cada ponto de início deve haver um ponto de término correspondente.

Portanto, nesta etapa verifica-se o número de pontos armazenados no vetor de visualização, se for par, pode-se passar a etapa seguinte, isto é, o vetor está completo.

No entanto, se o número de pontos armazenados for ímpar, tem-se duas situações possíveis: a primeira é que ocorreu um ponto de interseção em que a coordenada Z da face foi igual à coordenada Z da aresta, neste caso inclui-se o valor paramétrico armazenado na variável auxiliar (tzero) no vetor de visualização. A segunda possibilidade é que além do número de pontos no vetor ser ímpar, não houve igualdade das coordenadas Z dos pontos de interseção com a face. Neste caso, obrigatoriamente, ocorreu a situação em que a aresta perfura a face. O cálculo do valor paramétrico em que isto ocorre é feito em função das distâncias dos nós inicial e final em relação ao plano definido pela face, este valor é então incluído no vetor de visualização.

Passo 5: Ordenação do vetor

Pode-se observar que o cálculo dos pontos de interseção foram feitos de maneira não sequencial. Portanto, nesta etapa, deve-se ordenar o vetor de visualização, pois assim cada ponto de início será seguido pelo ponto de término correspondente, possibilitando a atualização do vetor de visualização global.

4 - TESTES PRELIMINARES

Com o fim de acelerar o tempo de processamento do algoritmo, podem ser incluidos testes preliminares para cada face antes de efetuar-se os cálculos geométricos ao longo dos passos do processo para ocultar uma linha.

O teste das coordenadas máximas e mínimas, no plano de projeção, da linha em questão com as coordenadas mínimas e máximas da face define se esta face pode ou não ocultar a linha.

Outro teste possível é o de verificar se a linha se localiza totalmente à frente da face (pode ser feito calculando-se os valores dos dois vértices da aresta na equação do plano definida pela face). Neste caso pode-se descartar esta face.

5 - EXEMPLOS

A figura 4 ilustra a capacidade do algoritmo de representar linhas de fluxo através de sólidos.

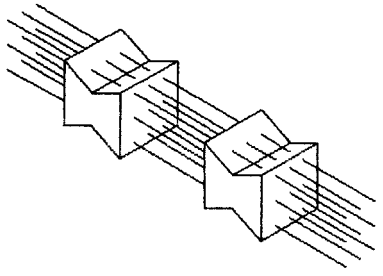
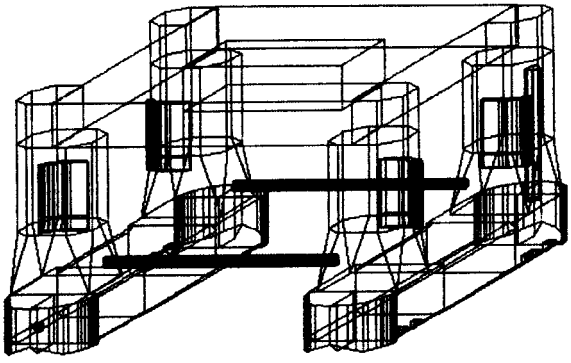
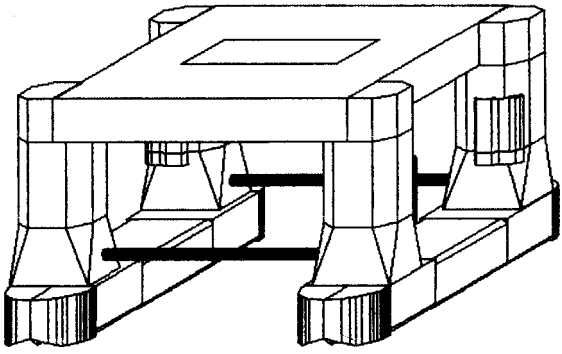


Fig. 4 - Linhas de fluxo através de sólidos.

A figura 5 ilustra o modelo de uma plataforma semisubmersível projetada pelo CENPES/PETROBRÁS. A figura 5.a mostra todas as arestas do modelo e a figura 5.b o resultado do algoritmo apresentado neste trabalho. Note-se que o volume central do convés é um volume interno de reforço e não um furo.



(a)



(b)

Fig. 5 - Modelo da plataforma Ava 137.

6 - CONCLUSÃO

O algoritmo proposto se mostrou robusto e eficiente no trato de modelos onde outros algoritmos da literatura falharam.

REFERÊNCIAS BIBLIOGRÁFICAS

[FOLE84] Foley, J. D. and Van Dam, A., Fundamentals of interactive computer graphics, Addison-Wesley, Reading, Massachussets, 1984.

[HARR87] Harrington, S. - Computer Graphics, A Programming Approach, McGraw-Hill Book Co., 1987.

[JANS83] Janssen, T. L. - "A simple efficient hidden line algorithm", Computers & Structures, Vol.17, 1983, pp.563-571.

[SCHI87] Schildt, H. - C avançado: guia do usuário, São Paulo: McGraw-Hill, 1987.

[SPIL86] Spillers, W. R. and Law, K. H. - "On the hidden line removal problem", Computers & Structures, Vol.26, 1986, pp.709-717.

AGRADECIMENTOS

A Luiz Henrique Figueiredo, pelo auxílio no trabalho e desenvolvimento do algoritmo de atualização do vetor de visualização.

Ao CNPq e ao CENPES/PETROBRÁS pelo auxílio financeiro.