Universidade Estadual do Oeste do Paraná - UNIOESTE
Centro de Ciências Exatas e Tecnológicas - CCET
Curso de Ciência da Computação

COMPUTAÇÃO GRÁFICA

NOTAS COMPLEMENTARES

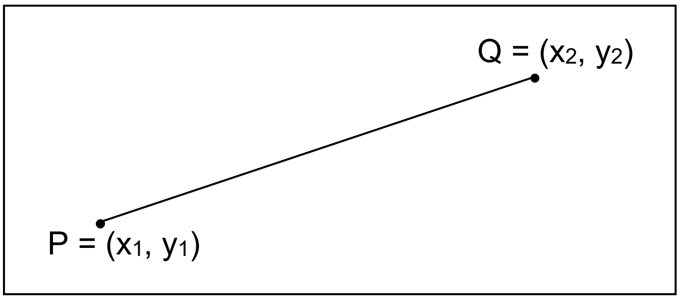
CASCAVEL - PR
2021

SUMÁRIO

1 PRINCÍPIOS GEOMÉTRICOS

1.1 EQUAÇÃO DA RETA

Dados 2 pontos P = (x_1, y_1) e Q = (x_2, y_2), temos a reta;



A equação da reta PQ pode ser assim escrita:

$$\frac{y-y_1}{x-x_1}=\frac{y_2-y_1}{x_2-x_1} \tag{1.1}$$

Fazendo

$$m=\frac{y_2-y_1}{x_2-x_1} \tag{1.2}$$

Substituindo (1.2) em (1.1) temos

$$\frac{y-y_1}{x-x_1}=m \Leftrightarrow y=m(x-x_1)+y_1 \tag{1.3}$$

Podemos escrever

$$b=y_1-mx_1 \tag{1.4}$$

Substituindo (1.4) em (1.3), escrevemos a reta PQ escrita como:

$$y=mx+b \tag{1.5}$$

Fazendo as devidas multiplicações em (1.1), temos:

$$(x - x_1)(y_2 - y_1) = (y - y_1)(x_2 - x_1)$$

$$(y_2 - y_1)x - (x_2 - x_1)y + x_2 y_1 - x_1 y_2 = 0 \tag{1.6}$$

Fazendo

$$\begin{aligned} r &= (y_2 - y_1) \\ s &= -(x_2 - x_1) \\ t &= x_2 y_1 - x_1 y_2 \end{aligned} \tag{1.7}$$

Substituindo (1.7) em (1.6), escrevemos

$$rx + sy + t = 0$$

De (1.4) e (1.6), concluímos que:

$$\begin{aligned} m &= \frac{-r}{s} \\ b &= \frac{-t}{s} \end{aligned}$$

1.2 INTERSEÇÃO ENTRE DUAS LINHAS

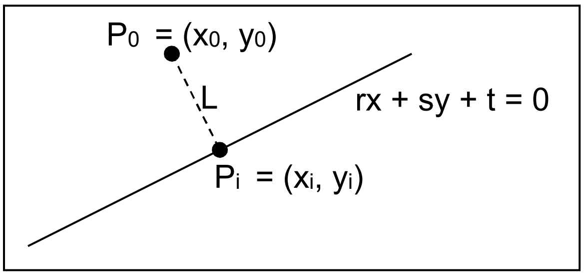
O ponto de interseção $P_i = (x_i, y_i)$ entre duas linhas é definido como:

$$P_i = \left(x_i = \frac{b_2 - b_1}{m_1 - m_2},\ y_i = \frac{b_2 m_1 - b_1 m_2}{m_1 - m_2} \right) \tag{1.8}$$

ou

$$P_i = \left(x_i = \frac{s_1 t_2 - s_2 t_1}{s_2 r_1 - s_1 r_2},\ y_i = \frac{t_1 r_2 - t_2 r_1}{s_2 r_1 - s_1 r_2} \right) \tag{1.9}$$

1.3 DISTÂNCIA ENTRE PONTO E RETA



A reta perpendicular a rx + sy + t = 0, que contém o segmento P_0P_i é dada por:

$$-sx + ry + (sx_0 - ry_0) = 0 \tag{1.10}$$

ou

$$y = \frac{s}{r}x + \left(y_0 - \frac{s}{r} \cdot x_0 \right) \tag{1.11}$$

com $m = \frac{s}{r}$ e $b = \left(y_0 - \frac{s}{r} \cdot x_0 \right)$.

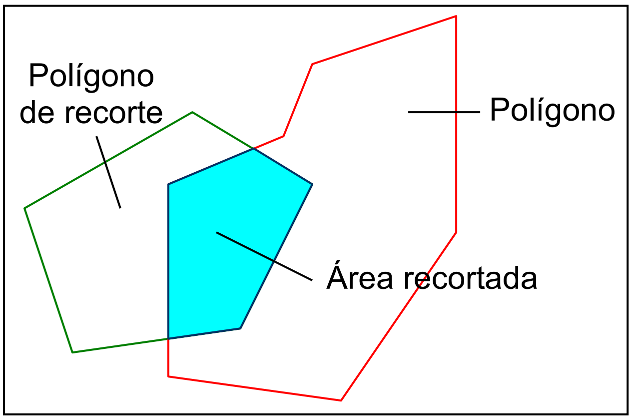
O ponto de interseção $P_i = (x_i, y_i)$ entre as retas rx + sy + t = 0 e a perpendicular, que contém o ponto P_0, pode ser calculado pelas equações (1.8) ou (1.9).

E a distância L é dada por:

$$L = \frac{|rx_0 + sy_0 + t|}{\sqrt{r^2 + s^2}}$$

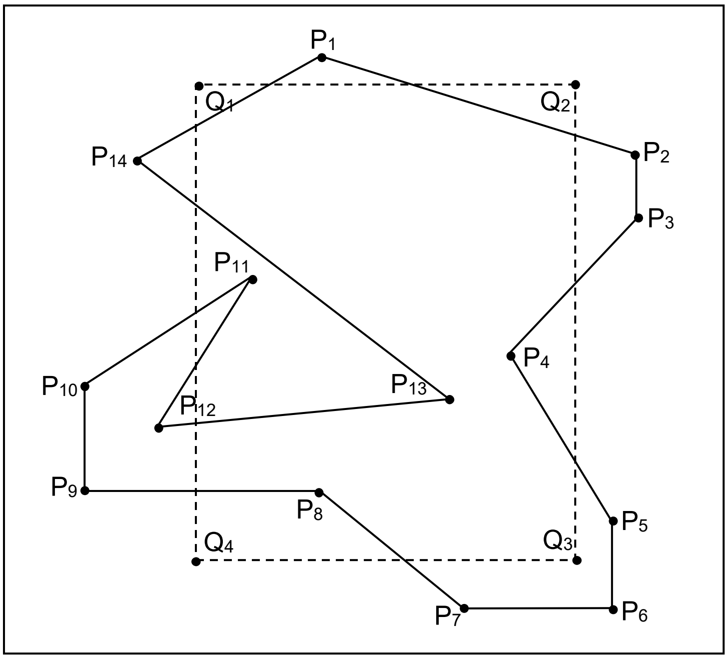
2 ALGORITMO DE WEILER-ATHERTON

Este algoritmo é utilizado para fazer o recorte de polígonos contra uma janela também poligonal. Veja um exemplo na figura a seguir.

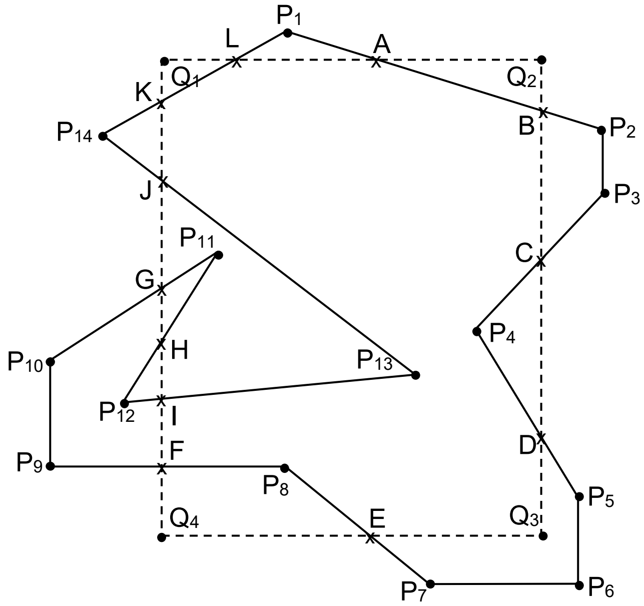


A única restrição é que ambos os polígonos não possuam auto-interseções. Para estes casos a solução indicada na literatura é a subdivisão destes em dois ou mais polígonos sem auto-interseções.

A seguir o algoritmo de Weiler-Atherton será detalhado, demonstrando num estudo de caso a seqüência de ações realizadas. O polígono P será recortado contra a janela Q.



Num primeiro passo calcule todas as interseções entre as arestas do polígono e as arestas da janela de recorte, rotulando-as como mostrado na figura a seguir.



Arbitre um sentido para realizar o caminhamento sobre as bordas do polígono e da janela de recorte. Deve ser o mesmo sentido para ambas as figuras geométricas. Adotou-se neste exemplo o sentido horário.

Caminhando sobre as arestas do polígono inserem-se os vértices originais e os vértices de interseção numa primeira lista. O mesmo processo é efetuado para a janela de recorte, dando origem a uma segunda lista de vértices. Numa terceira lista armazenam-se apenas os vértices de interseção para aquelas arestas do polígono que adentram a janela de recorte. As três listas são apresentadas a seguir.

Polígono	P_1	A	B	P_2	P_3	C	P_4	D	P_5	P_6	P_7	E	P_8	F	P_9	P_{10}	G	P_{11}	H	P_{12}	I	P_{13}	J	P_{14}	K	L

Janela	Q1	L	A	Q2	B	C	D	Q3	E	Q4	F	I	H	G	J	K

Vértices	A	C	E	G	I	K

O algoritmo consiste em retirar um vértice da lista de vértices e, a partir dele, caminhar sobre a lista do polígono. Encontrando um vértice de interseção troca-se para a lista da janela e caminha-se, a partir deste último vértice, sobre ela. Encontrando outro vértice de interseção troca-se novamente para lista de polígonos caminhando a partir do último vértice encontrado na lista da janela. O caminhamento sobre as lista pára quando um vértice já analisado for encontrado. Neste momento tem-se um polígono recortado. Se algum dos vértices deste polígono encontrar-se na lista de vértice deve-se removê-lo de lá.

Caso restem vértices na lista de vértices tem-se ainda outros polígonos a recortar. Para tanto repete-se o processo acima. O processo todo acaba quando a lista de vértices restar vazia. Observe esta seqüência de ações realizadas sobre as listas do exemplo.

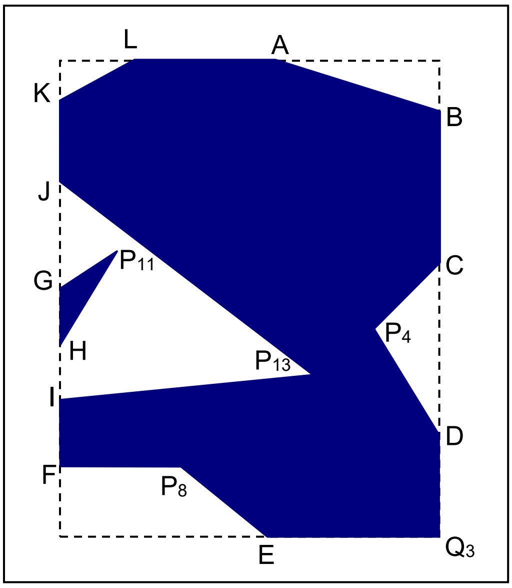
Inicie o algoritmo retirando um vértice da lista de vértice. Neste caso o vértice A. Na seqüência, caminhando sobre a lista do polígono, a partir de A encontra-se o vértice B. Troca-se para lista da janela e encontra-se o vértice C. Novamente troca-se para lista do polígono e encontram-se os vértices P_4 e D. Troca-se para lista da janela e encontram-se os vértices Q_3 e E. Trocamos para a lista do polígono e encontram-se os vértices P_8 e F. Troca-se para lista da janela e encontra-se o vértice I. Troca-se para a lista do polígono e encontram-se os vértices P_{13} e J. Troca-se para lista da janela e encontra-se o vértice K. Troca-se para a lista do polígono e encontra-se o vértice L. Troca-se para lista da janela e encontra-se o vértice A.

O vértice A é o primeiro vértice da lista, portanto o polígono recortado é formado pelas arestas dadas pelos vértices: A, B, C, P_4, D, Q_3, E, P_8, F, I, P_{13}, J, K, L.

Note que entre os vértices do primeiro polígono recortado estão os vértices C, E, I e K. Estes vértices fazem parte da lista de vértices e devem também ser de lá removidos.

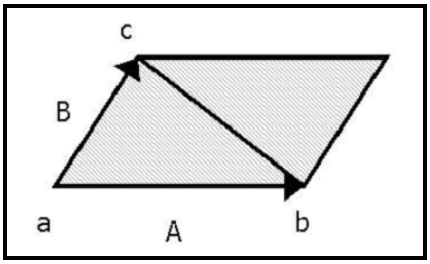
Restou ainda na lista de vértice o vértice G, indicando que há mais um polígono para recortar. Retira-se G da lista de vértice, inicia-se o caminhamento sobre a lista do polígono e encontram-se os vértices P_{11} e H. Troca-se para lista da janela e encontra-se o vértice G, já analisado. O segundo polígono é formado pelas arestas dadas pelos vértices: G, P_{11}, H.

Como a lista de vértices restou vazia conclui-se o algoritmo com o resultado apresentado na figura a seguir.


L
A
K
B
J
P11
G
C
P4
H
P13
I
D
F
P8
E
Q3

3 CÁLCULO DA ÁREA DE UM TRIÂNGULO

A área de um triângulo pode ser calculada a partir de seus três vértices a, b e c. O produto vetorial dos vetores A e B fornece a área do paralelogramo. Logo, a metade desta área corresponde à área do triângulo abc.



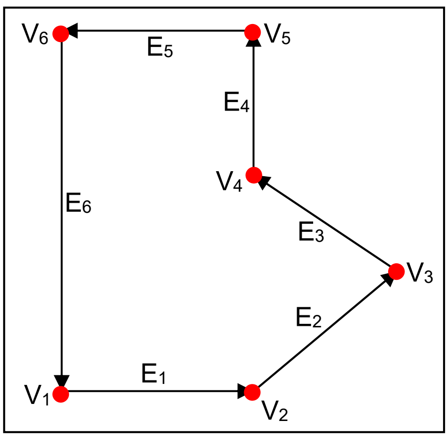
$$S = \frac{1}{2}\begin{vmatrix} 1 & 1 & 1 \\ X_A & Y_A & 0 \\ X_B & Y_B & 0 \end{vmatrix} = \frac{1}{2}\left(\left(x_b - x_a\right)\cdot\left(y_c - y_a\right) - \left(x_c - x_a\right)\cdot\left(y_b - y_a\right)\right)$$

3.1 IDENTIFICAÇÃO DA ORIENTAÇÃO DE TRÊS VÉRTICES

A equação anterior, além de fornecer a área do triângulo, também fornece a orientação dos três pontos que formam o triângulo. Caso a área seja negativa os pontos a, b e c estão no sentido-horário; caso seja positiva os pontos estão no sentido anti-horário. Se a área calculada for nula então os 3 pontos são colineares.

3.2 IDENTIFICAÇÃO DE POLÍGONOS CÔNCAVOS

Para identificar se um polígono é côncavo utiliza-se o produto vetorial. Veja o polígono da figura seguinte.



Computa-se o produto vetorial para todos os pares sucessivos de vetores de borda. Uma componente z negativa, resultante da multiplicação, posicionada entre componentes positivas indica uma concavidade local.

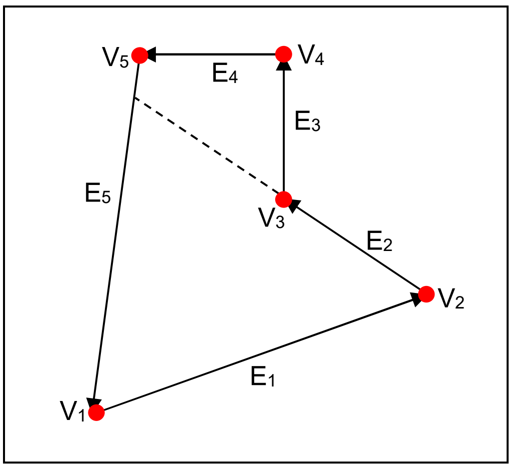
$$\left(\vec{E}_1 \times \vec{E}_2\right)_z > 0$$
$$\boxed{\begin{array}{l}\left(\vec{E}_2 \times \vec{E}_3\right)_z > 0 \\ \left(\vec{E}_3 \times \vec{E}_4\right)_z < 0 \\ \left(\vec{E}_4 \times \vec{E}_5\right)_z > 0\end{array}}$$
$$\left(\vec{E}_5 \times \vec{E}_6\right)_z > 0$$
$$\left(\vec{E}_6 \times \vec{E}_1\right)_z > 0$$

Outra forma de se fazer esta verificação é utilizar o teste de sentido apresentado no item 3.1. Se o sentido definido por três pontos consecutivos da fronteira de um polígono for sempre o mesmo então o polígono é convexo. Em polígonos côncavos pelo menos um dos testes indicará sentido contrário aos demais.

3.3 DIVISÃO DE POLÍGONOS CÔNCAVOS

Primeiramente identifique se o polígono é côncavo através do cálculo do produto vetorial entre os vetores de borda em sentido anti-horário.

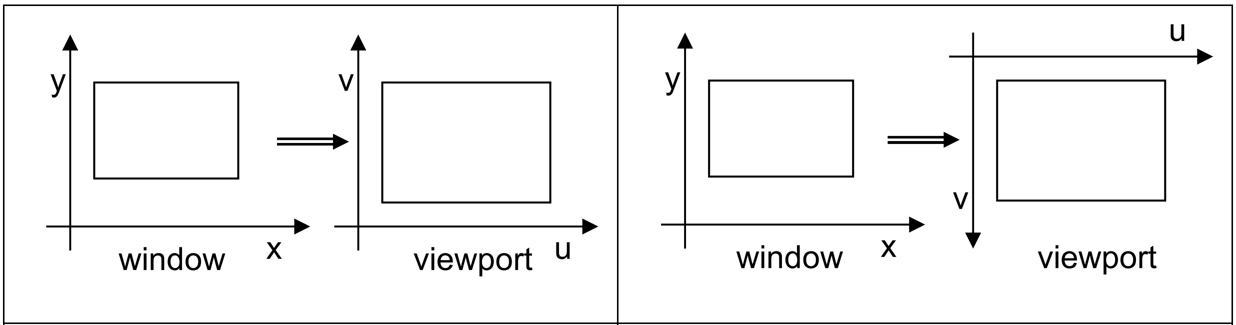
Se houver um par de vetores cujo produto vetorial apresente componente z negativa, conclui-se que o polígono é côncavo. Procede-se, então, a divisão ao longo do primeiro dos vetores deste produto vetorial. Veja o exemplo da figura seguinte.



4 TRANSFORMAÇÃO WINDOW-TO-VIEWPORT

O mapeamento de um objeto descrito em coordenadas de uma window (mundo) para uma viewport (tela) consiste numa seqüência de transformações geométricas simples envolvendo translações, escala e espelhamento.

Dois casos são possíveis. O primeiro caso ocorre quando temos os eixo y da window e o eixo v da viewport na mesma direção. No segundo caso o eixo Y e o eixo v estão em direções opostas.

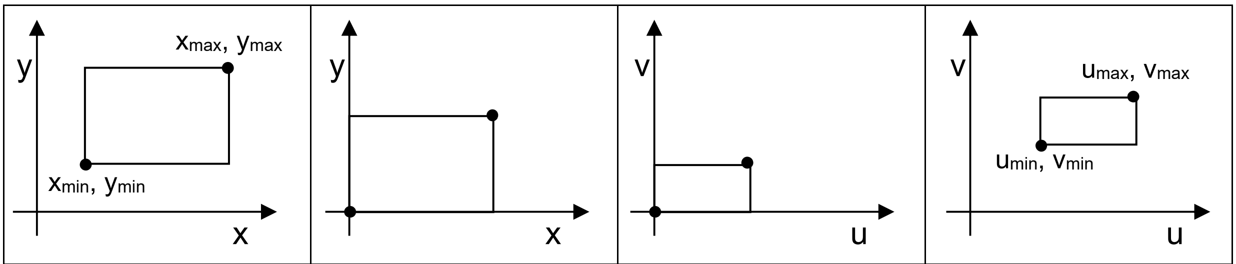


Caso 1 – Eixos y e u na mesma direção	Caso 2 – Eixos y e u em direções opostas

A seguir apresentaremos as soluções para ambos os casos.

4.1 CASO 1 – EIXOS Y E V NA MESMA DIREÇÃO

A solução para este caso consiste em encadear 3 transformações geométricas: uma translação que leva (x_{min}, y_{min}) para a origem do sistema de coordenadas da window, uma transformação de escala que iguala o tamanho da window e da viewport e uma translação que leva o ponto correspondente a (x_{min}, y_{min}) para a posição (u_{min}, v_{min}). Observe a figura abaixo.



window	translação	escala	translação

Encadeando as transformações temos:

$$M_{jp} = T(u_{min}, v_{min}) \cdot S\left(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}\right) \cdot T(-x_{min}, -y_{min})$$

$$M_{jp} = \begin{bmatrix} 1 & 0 & u_{min} \\ 0 & 1 & v_{min} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & 0 \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{min} \\ 0 & 1 & -y_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{jp} = \begin{bmatrix} \frac{u_{max} - u_{min}}{x_{max} - x_{min}} & 0 & -x_{min} \cdot \frac{u_{max} - u_{min}}{x_{max} - x_{min}} + u_{min} \\ 0 & \frac{v_{max} - v_{min}}{y_{max} - y_{min}} & -y_{min} \cdot \frac{v_{max} - v_{min}}{y_{max} - y_{min}} + v_{min} \\ 0 & 0 & 1 \end{bmatrix}$$

4.2 CASO 2 – EIXOS Y E V EM DIREÇÕES OPOSTAS

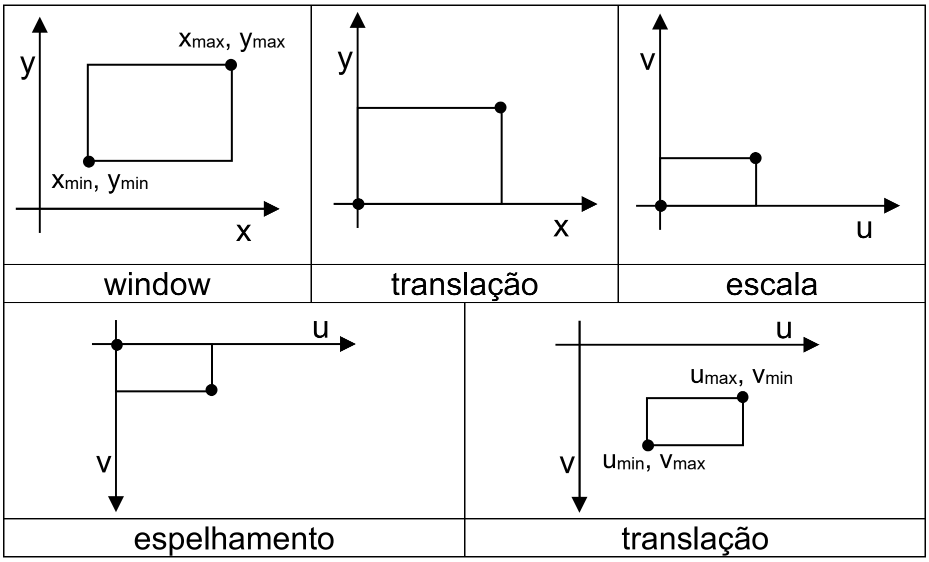
Este é o caso mais corriqueiro. Aprendemos, ainda na escola, a representar o plano cartesiano com a parte positiva do eixo y na ascendente; essa orientação é contrária a orientação utilizada nas telas dos computadores, onde a parte positiva do eixo y está orientada na descendente. A solução para este caso consiste em encadear 4 transformações geométricas: uma translação que leva (x_{min}, y_{min}) para a origem do sistema de coordenadas da window, uma transformação de escala que iguala o tamanho da window e da viewport, um espelhamento em relação ao eixo x para inverter a direção de y e uma translação que leva o ponto correspondente a (x_{min}, y_{min}) para a posição (u_{min}, v_{max}). Observe a figura seguinte.

Encadeando as transformações temos:

$$M_{jp} = T(u_{min}, v_{max}) \cdot E\left(\overrightarrow{Ox}\right) \cdot S\left(\frac{u_{max} - u_{min}}{x_{max} - x_{min}}, \frac{v_{max} - v_{min}}{y_{max} - y_{min}}\right) \cdot T(-x_{min}, -y_{min})$$

$$M_{jp} = \begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\max} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{jp} = \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & -x_{\min} \cdot \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ 0 & \frac{v_{\min} - v_{\max}}{y_{\max} - y_{\min}} & y_{\min} \cdot \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\max} \\ 0 & 0 & 1 \end{bmatrix}$$



window | translação | escala

espelhamento | translação

5 CONVERSÃO ENTRE SISTEMAS DE CORES

5.1 RGB E CMY

```
// Based on C Code in "Computer Graphics -- Principles and Practice,"
// Foley et al, 1996, p. 588
// R, G, B, C, M, Y each IN [0..255]

function CMYtoRGBTriple(const C, M, Y: integer): TRGBTriple;
begin
   with (result) do begin
      rgbtRed   := 255 - C;
      rgbtGreen := 255 - M;
      rgbtBlue  := 255 - Y
   end;
end; {CMYtoRGBTriple};

procedure RGBTripleToCMY(const RGB: TRGBTriple; var C, M, Y: integer);
begin
   with (RGB) do begin
      C := 255 - rgbtRed;
      M := 255 - rgbtGreen;
      Y := 255 - rgbtBlue
   end;
end; {RGBtoCMY};
```

5.2 RGB E CMYK

```
// Based on C Code in "Computer Graphics -- Principles and Practice,"
// Foley et al, 1996, p. 589
// R, G, B, C, M, Y,K each IN [0..255]

function CMYKtoRGBTriple(const C, M, Y, K: integer): TRGBTriple;
begin
   with (result) do begin
      rgbtRed   := 255 - (C + K);
      rgbtGreen := 255 - (M + K);
      rgbtBlue  := 255 - (Y + K)
   end;
end; {CMYtoRGBTriple};

procedure RGBTripleToCMYK(const RGB: TRGBTriple; var C, M, Y, K: integer);
begin
   RGBTripleToCMY(RGB, C, M, Y);
   K := MinIntValue([C, M, Y]);
   C := C - K;
   M := M - K;
   Y := Y - K
end; {RGBtoCMYK}
```

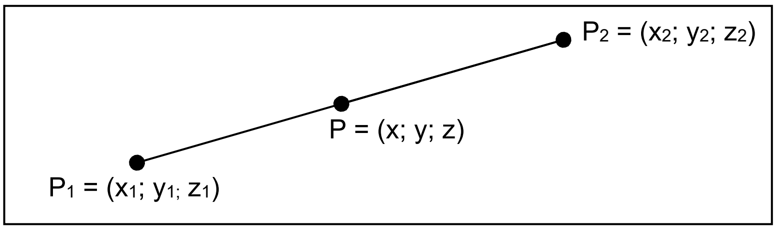
5.3 RGB E HSV

```
// H = 0 to 360 (corresponding to 0..360 degrees around hexcone)
// 0 (undefined) for S = 0
// S = 0 (shade of gray) to 255 (pure color)
// V = 0 (black) to 255 (white)
// RGB, each 0 to 255, to HSV.
```

```
function HSVtoRGBTriple(const H, S, V: integer): TRGBTriple;
const
   divisor:  integer = 255*60;
var
   f       : integer;
   hTemp   : integer;
   p, q, t : integer;
   vs      : integer;
begin
   if (s = 0) then result := RGBtoRGBTriple(V, V, V) //achromatic
   else begin //chromatic color
      if (H = 360) then hTemp := 0
      else hTemp := H;

      f := hTemp mod 60; // f is IN [0, 59]
      hTemp := hTemp div 60; // h is now IN [0,6)
      vs := V * S;
      p := V - vs div 255;
      q := V - (vs * f) div divisor;
      t := V - (vs *(60 - f)) div divisor;

      case (hTemp) of
         0: result := RGBtoRGBTriple(V, t, p);
         1: result := RGBtoRGBTriple(q, V, p);
         2: result := RGBtoRGBTriple(p, V, t);
         3: result := RGBtoRGBTriple(p, q, V);
         4: result := RGBtoRGBTriple(t, p, V);
         5: result := RGBtoRGBTriple(V, p, q);
         else result := RGBtoRGBTriple(0,0,0);
      end;
    end;
  end; {HSVtoRGBTriple};

procedure RGBTripleToHSV(const RGBTriple: TRGBTriple; var H, S, V: integer);
var
   Delta : integer;
   Min   : integer;
Begin
   with (RGBTriple) do begin
      Min := MinIntValue([rgbtRed, rgbtGreen, rgbtBlue]);
      V   := MaxIntValue([rgbtRed, rgbtGreen, rgbtBlue])
   end;

   Delta := V - Min;

   //Calculate saturation:  saturation is 0 if R, G and B are all 0
   if (V = 0) then S := 0
   else S := MulDiv(Delta, 255, V); //Int(Delta * 255/V)\

   if (S = 0) then H := 0 //Achromatic
   else begin
      with (RGBTriple) do begin
         if (rgbtRed = V) then //degrees -- between yellow and magenta
            H := MulDiv(rgbtGreen - rgbtBlue, 60, Delta)
         else
            if (rgbtGreen = V) then //between cyan and yellow
               H := 120 + MulDiv(rgbtBlue - rgbtRed, 60, Delta)
            else
               if (rgbtBlue = V) then //between magenta and cyan
                  H := 240 + MulDiv(rgbtRed - rgbtGreen, 60, Delta);
      end;

      if (H < 0) then H := H + 360;

    end;
end; {RGBTripleToHSV};
```

6 INTERPOLAÇÃO LINEAR

Dois pontos não coincidentes definem um segmento de reta. Veja a figura a seguir.

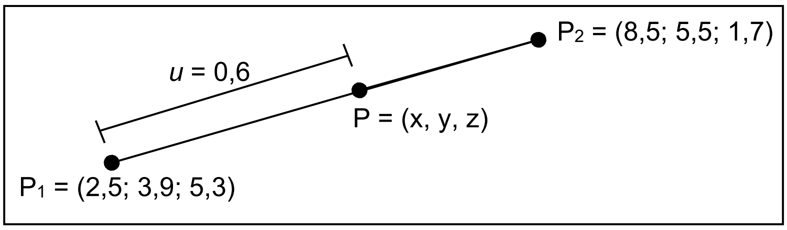


As coordenadas de qualquer ponto P = (x; y; z), compreendido no segmento P_1P_2, são obtidas por:

$$\begin{aligned} x &= x_1 + u \cdot (x_2 - x_1) \\ y &= y_1 + u \cdot (y_2 - y_1), \text{ com } 0 \le u \le 1 \\ z &= z_1 + u \cdot (z_2 - z_1) \end{aligned} \qquad (6.1)$$

O segmento de reta P_1P_2 está descrito parametricamente, em função dos vértices extremos e do parâmetro *u*. O parâmetro *u* assume valores entre 0 e 1; quando *u* = 0, P = P_1 e quando *u* = 1, P = P_2. Para quaisquer outros valores de u, dentro do limite possível, estaremos em um ponto interno ao segmento.

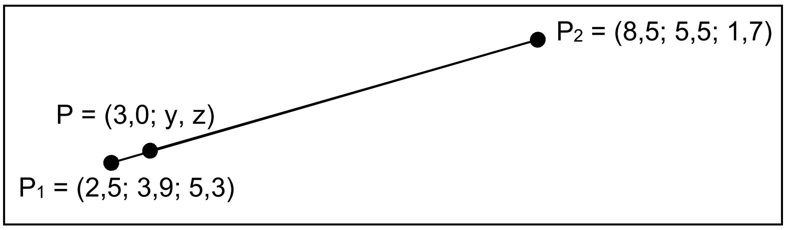
Vejamos um exemplo numérico. Assuma que P é um ponto que está a 60% do comprimento do segmento (*u* = 0,6), medido a partir do vértice P_1.



Aplicando o parâmetro *u* = 0,6 em (6.1) temos P = (x; y; z):

$$\begin{aligned} x &= 2{,}5 + 0{,}6 \cdot (8{,}5 - 2{,}5) = 6{,}1 \\ y &= 3{,}9 + 0{,}6 \cdot (5{,}5 - 3{,}9) = 4{,}86 \\ z &= 5{,}3 + 0{,}6 \cdot (1{,}7 - 5{,}3) = 3{,}14 \end{aligned}$$

Neste exemplo temos que o ponto P está a certa distância de P_1. Entretanto as equações (6.1) nos permitem interpolar linearmente qualquer ponto no segmento, desde que seja conhecia ao menos uma das coordenadas de P. Vejamos o exemplo a seguir, onde conhecemos a coordenada *x* = 3, do ponto P.



Como conhecemos o valor da coordenada *x* de P, a partir de (6.1) obtemos o valor do parâmetro *u*.

$$x = x_1 + u \cdot (x_2 - x_1)$$
$$3 = 2{,}5 + u \cdot (8{,}5 - 2{,}5)$$
$$u = \frac{0{,}5}{6} = 0{,}0833$$

Conhecido o valor de *u*, podemos calcular os valores das coordenadas y e *z*.

$$y = 3{,}9 + 0{,}0833 \cdot (5{,}5 - 3{,}9) = 4{,}033$$
$$z = 5{,}3 + 0{,}0833 \cdot (1{,}7 - 5{,}3) = 5{,}0$$

Ou seja, P = (3,0; 4,033; 5,0).

6.1 INTERPOLAÇÃO COM CÁLCULO INCREMENTAL

Suponha agora que desejamos interpolar outros pontos no segmento, considerando incrementos de um ao longo de x; ou seja, P = (4; y; z), P = (5; y; z), P = (6; y; z), P = (7; y, z) e P = (8; y; z).

Neste caso não é necessário realizar o cálculo de *u* para cada novo valor de *x* em P, e depois calcular os valores de *y* e *z*. Utilizamos aritmética incremental para tal, pois o segmento P_1P_2 é linear.

Como desejamos interpolar pontos com coordenadas x inteiras e incrementadas de um em um, o primeiro passo do processo de interpolação é obter um primeiro ponto P, no segmento P_1P_2, com x em valor inteiro. Este ponto já foi calculado e possui valor P_i = (3,0; 4,033; 5,0).

O segundo passo é calcular as taxas de variação das coordenadas *y* e *z* do segmento em função da variação de *x*, pois o incremento unitário foi considerado em *x*. As taxas são calculadas a seguir:

$$Ty = \frac{(P_2 y - P_1 y)}{(P_2 x - P_1 x)} = \frac{5{,}5 - 3{,}9}{8{,}5 - 2{,}5} = 0{,}2667$$

$$Tz = \frac{(P_2 z - P_1 z)}{(P_2 x - P_1 x)} = \frac{1{,}7 - 5{,}3}{8{,}5 - 2{,}5} = -0{,}6$$

Em outras palavras, para cada unidade de variação em x ao longo do segmento P_1P_2, a coordenada *y* sofre um acréscimo de 0,2667 unidades enquanto que a coordenada *z* sofre um decréscimo de 0,6 unidades.

O terceiro passo é realizar a interpolação incremental, somando-se as taxas correspondentes às coordenadas *y* e *z*, a partir do primeiro ponto com coordenada *x* inteira – P_i = (3,0; 4,033; 5,0). Ou seja, $P_{i+1} = P_i$ + Taxas

Coordenadas	Taxas	P_i	P_{i+1}	P_{i+2}	P_{i+3}	P_{i+4}	P_{i+5}
x	---	3	4	5	6	7	8
y	0,2667	4,033	4,2997	4,5664	4,8331	5,0998	5,3665
z	-0,6	5,0	4,4	3,8	3,2	2,6	2

7 ALGORITMO DE LIANG-BARSKY PARA RECORTE DE LINHAS

Este algoritmo tem como base a análise da equação paramétrica de um segmento de reta, que pode ser escrita como:

$$\begin{aligned} x &= x_1 + u \cdot \Delta x \\ y &= y_1 + u \cdot \Delta y,\ com\ 0 \le u \le 1 \end{aligned} \tag{7.1}$$

onde $\Delta x = x_2 - x_1$ e $\Delta y = y_2 - y_1$. Usando estas equações paramétricas, Cyrus e Beck desenvolveram um algoritmo que, no caso geral, é mais eficiente que o algoritmo de Cohen-Sutherland. Posteriormente, Liang e Barsky desenvolveram independentemente um algoritmo para recorte de linhas ainda mais rápido.

No algoritmo de Lyan-Barsky reescrevemos as condições para recorte de pontos (eq. 7.2) usando a forma paramétrica (eq. 7.3).

$$\begin{aligned} x_{\min} &\le x \le x_{\max} \\ y_{\min} &\le y \le y_{\max} \end{aligned} \tag{7.2}$$

$$\begin{aligned} x_{\min} &\le x_1 + u \cdot \Delta x \le x_{\max} \\ y_{\min} &\le y1 + u \cdot \Delta y \le y_{\max} \end{aligned} \tag{7.3}$$

Cada uma destas quatro desigualdades pode ser escrita como:

$$u \cdot p_k \le q_k, \quad com\ k = 1, 2, 3, 4 \tag{7.4}$$

cujos parâmetros *p* e *q* são definidos por:

$$\begin{aligned} p_1 &= -\Delta x, & q_1 &= x_1 - x_{\min} \\ p_2 &= \Delta x, & q_2 &= x_{\max} - x_1 \\ p_3 &= -\Delta y, & q_3 &= y_1 - y_{\min} \\ p_4 &= \Delta y, & q_4 &= y_{\max} - y_1 \end{aligned} \tag{7.5}$$

Qualquer linha paralela a alguma das bordas da área de recorte tem $p_k = 0$, para o valor de *k* correspondente àquela borda (*k* = 1, 2, 3 e 4, correspondentes às bordas esquerda, direita, abaixo e acima, respectivamente). Se, para aquele valor de *k*, também encontrarmos $q_k < 0$, então a linha está completamente fora da área de recorte e pode ser descartada. Se $q_k \ge 0$, a linha está no subespaço interno àquela borda paralela *k*.

Quando $p_k < 0$, a reta suporte da linha adentra o subespaço delimitado pela borda *k*. Se $p_k > 0$, a linha sai do subespaço delimitado pela borda *k*. Para valores não nulos de p_k podemos calcular o valor de u que corresponde ao ponto onde a reta suporte da linha intercepta a borda k por:

$$u = \frac{q_k}{p_k} \tag{7.6}$$

Para cada linha calculamos os valores dos parâmetros u_1 e u_2 que definem qual parte da linha fica dentro da área de recorte. O valor de u_1 é obtido observando-se as bordas da área de recorte para aquelas linhas que adentram a mesma ($p < 0$). Para estas bordas, calculamos $r_k = q_k/p_k$. O valor de u_1 é o maior do conjunto {0, valores de r}. Inversamente, o valor de u_2 é obtido observando-se as bordas da área de recorte para aquelas linhas que saem da mesma ($p > 0$). Um valor de r_k é calculado para cada uma destas bordas e o valor de u_2 é o menor valor do conjunto {1, valores de r}. Se $u_1 > u_2$, a linha está completamente fora da área de recorte e pode ser rejeitada. Por outro lado, os pontos de interseção são calculados usando os dois valores do parâmetro u.

A seguir temos o código em C do algoritmo de Liang-Barsky.

```
#define TRUE 1
#define FALSE 0
#define ROUND(a) ((int)(a+0.5))

int clipTest (float p, float q, float *u1, float *u2){
   float r;
   int retVal = TRUE;

   if (p < 0.0){
      r = q/p;
      if (r > *u2) retVal = FALSE;
      else if (r > *u1) *u1 = r;
   }
   else{
      if (p > 0.0){
         r = q/p;
         if (r < *u1) retVal = FALSE;
         else if (r < *u2) *u2 = r;
      }
      else //p = 0, portanto a linha é paralela à borda
         if (q < 0.0) retVal = FALSE; //linha fora da área
   }
   return(retVal);
}

void clipLine (int xmin, int ymin, int xmax, int ymax, int x1, int y1, int
x2, int y2){
   float u1 = 0.0, u2 = 1.0, dx = x2 - x1, dy;

   if (clipTest(-dx, x1 - xmin, &u1, &u2))
      if (clipTest(dx, xmax - x1, &u1, &u2)){
         dy = y2 - y1;
```

```
            if (clipTest(-dy, y1 - ymin, &u1, &u2))
               if (clipTest(dy, ymax - y1, &u1, &u2)){
                  if (u2 < 1.0){
                     x2 = x1 + u2 * dx;
                     y2 = y1 + u2 * dy;
                  }
                  if (u1 > 0.0){
                     x1 += u1 * dx;
                     y1 += u1 * dy;
                  }
                  Line(ROUND(x1), ROUND(y1), ROUND(x2), ROUND(y2));
               }
      }
}
```