3. Sistemas de Coordenadas e Primitivas Básicas

O Sistema de Coordenadas de Vídeo

Existem diferentes dispositivos de vídeo, com diferentes medidas em pixels (resolução). Se desejarmos trabalhar na direção da máxima independência, devemos especificar coordenadas em outra unidade que não seja o pixel e criar um processo de Transformação que converta essas coordenadas para o sistema de coordenadas absolutas do dispositivo de vídeo em uso.

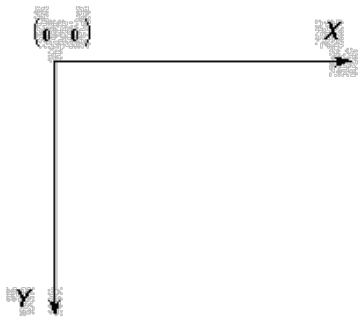

Fig. 3.1 O Sistema de Coordenadas de Vídeo.

Um outro aspecto importante é que este sistema não é confortável e compatível com a nossa experiência visual e intuição do que seja um sistema de coordenadas cartesianas.

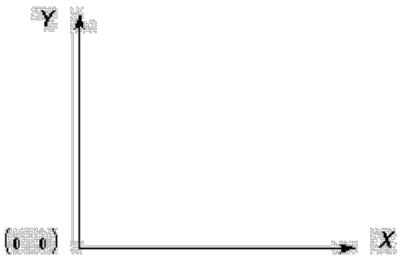

Fig. 3.2 O Sistema de Coordenadas de Vídeo segundo a intuição do usuário.

Se especificarmos um objeto no sistema de coordenadas usual e em seguida exibirmos na istema de coordenadas do vídeo, este objeto aparecerá invertido, conforme mostra a figura 3.2.

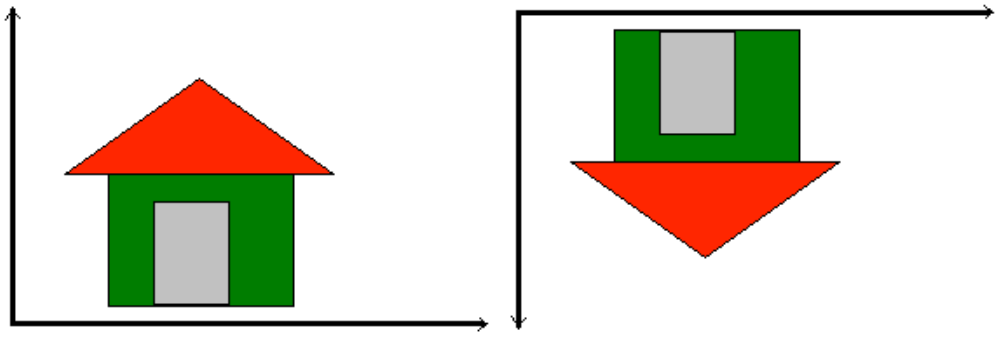

Fig. 3.3 Aplicação do Processo de Transformação.

Vamos dar um primeiro passo na criação de um processo de transformação que permita especificar objetos em um sistema de coordenadas mais adequado à nossa experiência visual. Dado um ponto ***P*** com coordenadas (x_u, y_u) no sistema de coordenadas usual, então as coordenadas (x_t , y_t) de P no sistema do vídeo podem ser calculadas como

$x_t = x_u$ *e* $y_t = H - y_u$, onde ***H*** é altura da janela onde o ponto será exibido (Fig 3.4).

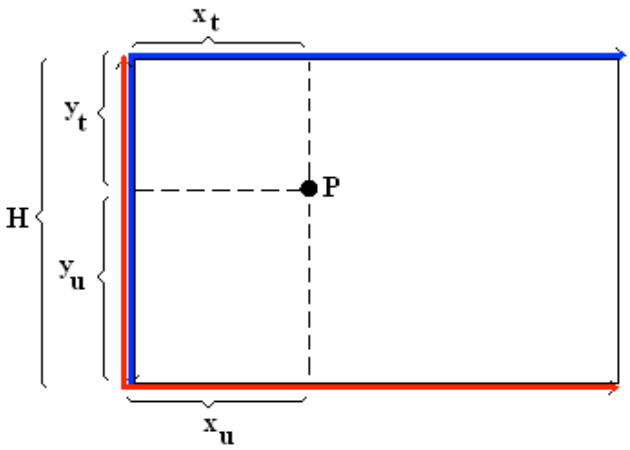

Fig 3.4

De modo geral, se a origem do sistema de coordenadas cartesianas usual estiver localizado na posição ***O***=(O_x ,O_y) da tela, então a relação entre as coordenadas cartesianas (x_u, y_u) e (x_t , y_t) será

$x_t = O_x + x_u$ *e* $y_t = O_y + y_u$ (Fig 3.5).

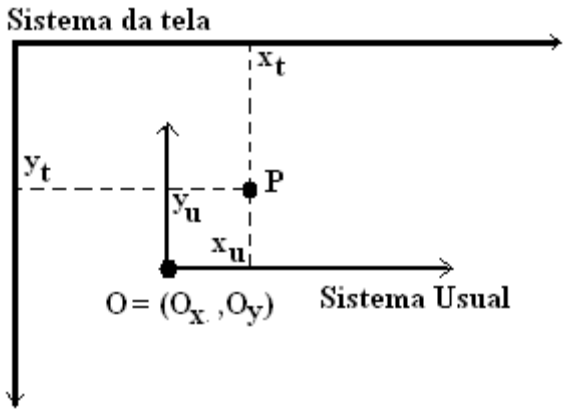

Fig 3.5.

Estas transformações rientam o eixo “Y” na direção “para cima” e não “para baixo” como é a orientação "default" no Ambiente MS Windows.

Primitivas Básicas

- **Ponto**

Um ponto 2D é uma primitiva especificada a partir de suas coordenadas x e y.

Com a biblioteca Opengl, para desenhar pontos é necessário inidicar ao sistema através da função *glBegin(GL_POINTS)*. Podemos, então desenhar um ponto 2D através da função *glVertex2f (float x, float y)*, onde x e y sâo as coordenadas do ponto. Algumas variações desta função são *glVertex2d* e *glVertex2i* que desenham um ponto em coordenadas inteiras.

Atributos: sendo o ponto, por definição, um ente geométrico infinitesimal não é cabível qualquer outro atributo além da sua **posição no espaço**. O seu equivalente no contexto gráfico digital é o Pixel, que por ser um elemento finito possui outros atributos além da **posição**, tais como, a **cor** e a **tamanho** do ponto. Para especificar a cor de um ponto em Opengl, deve-se chamar a função *glColor3f (float r, float g, float b)*, antes de chamar glVertex, onde *r*, *g* e *b* são as componentes vermelho ,verde e azul que definem a cor do ponto. Para especificar o tamanho do ponto deve-se chamar a função *glPointSize (float t),* onde *t* indica o tamanho do ponto.

Quando não mais se deseja desenhar pontos, deve-se chamar a função *glEnd()*.

- **A Primitiva Linha**

Esta operação corresponde ao traçado de um segmento de linha por **dois** pontos. Para desenhar linhas em Opengl, deve-se primeiro chamar a função *glBegin(GL_LINES)*. Isto indica que, a partir de agora, cada dois pontos formarão uma linha, ou seja deve-se chamar a função *glVertez2f* um número *n* par de vezes para desenhar *n*/2 linhas. Após desenhar as linhas desejadas, deve-se chamar a função *glEnd()*.

Atributos: Podemos controlar os seguintes atributos da linha:

1. Estilo - contínuo, pontilhado, etc., através da função *glLineStiple(int fator, int padrao)*
2. Largura ou Espessura – através da função *glLineWidth(float s)*
3. Cor – através da função *glColor3f (float r, float g, float b)*

Estas funções devem ser chamadas antes de começar a desenhar a linha .

Existem duas variações da primitiva GL_LINES:

- GL_LINE_STRIP: Dados vértices v0,v1,...vn, através da função glVertex, esta primitiva desenhar os segmentos de reta v0v1, v1v2, ... vn-1vn.
- GL_LINE_LOOP: Dados vértices v0,v1,...vn, através da função glVertex, esta primitiva desenhar os segmentos de reta v0v1, v1v2, ... vn-1vn e vnv0.

- **A Primitiva Triângulo**

Para desenhar triângulos em Opengl, deve-se primeiro chamar a função *glBegin(GL_TRIANGLE)*. Isto indica que, a partir de agora, cada **três** pontos formarão um triângulo, ou seja deve-se chamar três vezes a função *glVertez2f* (ou um número de vezes múltimplo de três). Após desenhar os triângulos desejados, deve-se chamar a função *glEnd()*.

O principal atributo de peenchimento é a cor, através da função *glColor3f (float r, float g, float b)* , que deve ser chamada antes de começar a desenhar o triângulo.

- **A Primitiva Polígono**

Para desenhar polígonos em geral, em Opengl, deve-se primeiro chamar a função *glBegin(GL_POLYGON)*. Isto indica que, a partir de agora, todos os pontos desenhados com *glVertez2f* formarão um polígono. Após desenhar o polígono desejado, deve-se chamar a função *glEnd()*. É interessante destacar que esta função só é capaz de desenhar polígonos convexos. Para desenhar polígonos não convexos, é necessário usar primitivas mais simples que sejam convexas.

O principal atributo de peenchimento é a cor, através da função *glColor3f (float r, float g, float b)* , que deve ser chamada antes de começar a desenhar o polígono.

- **A Primitiva Quadrilátero**

Analogamente à primitiva polígono, podemos usar a primitiva GL_QUADS para desenhar polígonos de quatro lados, ou quadriláteros.

Dica: experimente também usar as primitivas GL_TRIANGLE_STRIP, GL_TRIANGLE_FAN e GL_QUAD_STRIP.