8. Síntese de Imagens: Ocultamento das Superficies

Aprendemos a modelar e a projetar objetos 3D em um sistema de vídeo que é apenas bi-dimensional. Ocorre, no entanto, que todas as partes do objeto são sempre exibidas, resultando em imagens que se tornam emaranhados de linhas ou confusas superposições de polígonos.

As imagens criadas a partir de linhas apresentam-se transparentes, como se fossem contornos de um suposto corpo sólido. Essas imagens são chamadas *wireframes*. Torna-se difícil avaliar quando uma linha pertence a uma parte frontal ou posterior de um objeto. Quando se tratam de polígonos apresentados aleatoriamente, a confusão visual permanece.

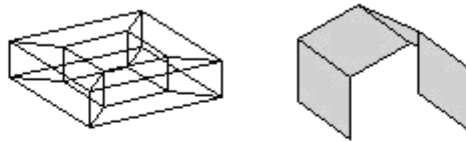

Fig. 8.1 Imagens sem realismo visual.

Assim, necessitamos desenvolver procedimentos que nos permitam remover as partes invisíveis das imagens, criando representações mais realísticas dos objetos. No mundo real, materiais opacos obstruem os raios de luz provenientes das partes escondidas dos objetos, impedindo sua visualização. Um dos desafios da Computação Gráfica é justamente remover as partes escondidas da imagem de um objeto sólido.

• 8.1. Ocultamento (Remoção de Áreas Escondidas)

Independente da forma como os objetos são modelados, os fundamentos dos algoritmos para ocultamento e shading não se alteram. Como vimos nas aulas anteriores, a primitiva básica que usaremos na modelagem de objetos é o polígono (mais especificamente, triângulos e quadriláteros). Podemos associar um polígono a uma *face*. É cada face (também chamada de *patch*) que, sendo opaca, causa a invisibilidade de outras partes do objeto, ou também, por causa de sua própria posição relativa ao observador, torna-se visível ou não. E ainda, é sobre a face ou o patch que efetuamos o shading.

O processo de ocultamento pode se tornar pesado, à medida que utilizamos mais polígonos para representar os objetos. Esse aumento é inevitável quando modelamos formas suavizadas e refinamos o processo de shading (este processo de shading também se chama de *renderização,* que significa “acabamento refinado”).

8.1.1. Remoção de Áreas Escondidas em OpenGL

Imagine que queremos desenhar uma cena na qual encontram-se dois objetos fixos A e B e que a posição do observador (ou da câmera) mude cada vez que apertamos uma tecla ou movemos o mouse. O trecho de código abaixo dá uma idéia deste processo:

```
while (1){
      Atualizar_posicao_camera();
```

```
        glClear(GL_COLOR_BUFFER_BIT)   //Ja vimos que isto limpa o buffer
        de cores
        Desenhar_objeto_A();
        Desenhar_objet_B();
}
```

Em algumas posições da câmera o objeto A aparece à frente do objeto B, ofuscando-o. Em outras posições pode acontecer o inverso. Porém, se nada for acrescentado ao código anterior, o objeto B sempre ofuscará o objeto A, pois está sendo desenhado por último.

Para eliminar as partes dos objetos que estão escondidas o Opengl usa um dos mais tradicionais algoritmos, chamado de *Z-buffer*. A idéia do Z-buffer, é ter um buffer (memória) na qual, para cada pixel da imagem, é associada uma lista de distâncias que indicam a distância de cada superfície à tela virtual do volume de visão, como mostra a figura 8.2. Estas distâncias são medidas ao longo do eixo Z da câmera,

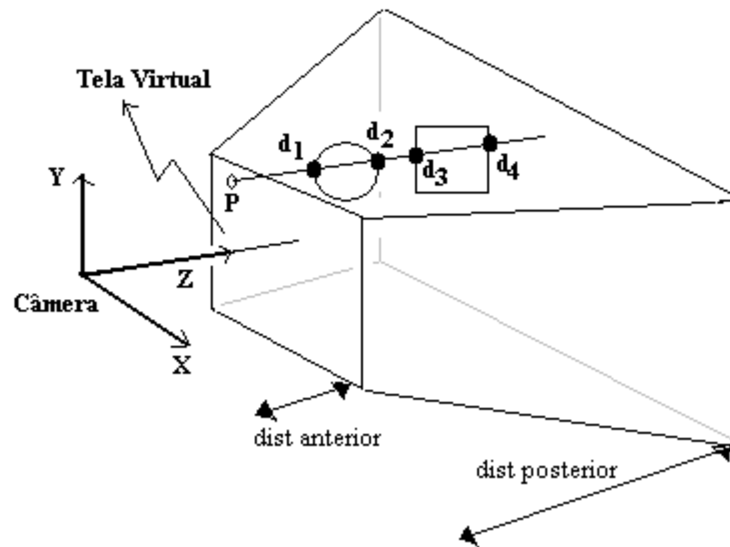

Fig. 8.2 Lista de distâncias associadas ao ponto P .

Para indicar ao Opengl que iremos utilizar remoção de área escondida devemos primeiramente utilizar a função glEnable com o parâmetro GL_DEPTH_TEST:

```
glEnable (GL_DEPTH_TEST);
```

Com isto o Opengl sabe que deverá usar seu buffer de profundidades. Devemos iniciar este buffer com as maiores distâncias possíveis, dentro do nosso contexto. Estas distâncias são a distância posterior do plano de recorte mais distante, utilizado no volume de visão (fig 8.2). Para iniciar o buffer de profundidade com estes valores , basta chamar a função glClear, passando o parâmetro GL_DEPTH_BUFFER_BIT.

```
glClear (GL_DEPTH_BUFFER_BIT);
```

O processo de geração da imagem é feito através de cálculos que convertem cada superfície a ser desenhada em um conjunto de pixels na janela onde a superfície irá aparecer (caso não esteja

escondida por outra superfície). Além disso, para cada superfície é calculada sua distância à tela virtual (ao longo do eixo Z). Se a opção de profundida foi habilitada (com glEnable(GL_DEPTH_TEST), então antes de cada pixel ser desenhado, é feita a comparação entre esta distância calculada e a distância armazenada no buffer de profundidade. Se a distância do novo pixel for menor do que a que estava armazenada, então o buffer de profundidade é atualizado com a nova distância (é importante destacar que o buffer de cores também é atualizado com a cor do novo pixel, que está mais próximo). Se distância do novo pixel nâo for menor, significa que a superfície está escondida neste ponto, e o pixel será descartado.

O código anterior pode ser reescrito agora utilizando-se a remoção de área escondida:

```
glutInitDisplayMode(GLUT_DEPTH|...);//Para a GLUT suportar profundidade
glEnable(GL_DEPTH_TEST); //Ativar a profundidade
. . .
while (1){
      glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT) //limpar buffers
      Atualizar_posicao_camera();
      Desenhar_objeto_A();
      Desenhar_objet_B();
}
```

O **`glEnable(GL_DEPTH_TEST)`** só precisa ser chamado uma única vez. Antes de desenhar, cada vez que a cena é exibida, é necessário limpar o buffer de profundida (a exemplo do buffer de cores), usando o **glClear**.