# 3 Modelagem de Objetos Baseada em imagens

*But... Look, see those birds? At some point a program was written to govern them. A program was written to watch over the trees, and the wind, the sunrise, and sunset. There are programs running all over the place. (The Matrix)*

## 3.1 Introdução

Há diversas aplicações que podem requerer que apenas alguns grupos de objetos sejam representados por imagens (*image-based objects*), deixando que o restante da cena seja representado por geometria tradicional (geo*metry-based objects*). Isto permite que se possam utilizar as vantagens existentes em cada tipo de representação: geometrias simples e eficientes para serem processadas pelo *pipeline* da GPU são encaminhadas diretamente ao *hardware*. Objetos de natureza mais complexa ou que requerem uma visualização cujo *hardware* gráfico seja incapaz de calcular são pré-processados como objetos baseados em imagens e re-encaminhados ao *pipeline* da GPU na forma de *sprites*, *billboards*, impostores, portais ou panoramas.

Modelar um objeto através de imagens pode resumir-se a projetar uma textura com a figura do elemento que se deseja criar sobre um plano inserido no espaço. Entretanto, apenas fazer isto traz uma série de problemas: ao mover o objeto ou a câmera este plano texturizado não reproduz corretamente as novas vistas do objeto. Mesmo girando o plano, de forma que a sua normal esteja sempre apontada para o observador, não se tem idéia de tridimensionalidade do mesmo, pois a imagem permanece sempre a mesma. Neste capítulo são apresentadas diversas abordagens para representar objetos deste tipo, bem como soluções para o problema recém apresentado: *sprites*, impostores, objetos com texturas com relevo e finalmente a proposta deste trabalho, que são os impostores com relevo.

## 3.2 Sprites e Billboards

*Sprites* são planos (comumente representados por polígonos) com texturas aplicadas sobre si. Ao se criar um objeto através do mapeamento de uma imagem (foto ou imagem sintetizada) sobre um plano, surge um inconveniente: o objeto sempre será retangular. Isto ocorre devido à natureza retangular da imagem a ele aplicada, fazendo com que os elementos não possuam uma silhueta adequada. Para contornar este problema, costuma-se utilizar texturas com transparência: o valor da transparência de cada *pixel* será descrito por um byte extra, chamado de canal alfa (*alpha channel*). Para resolver a aparência plana do *sprite*, pode-se realizar uma rotação do mesmo sempre que ele ou o observador se movem, garantindo que a normal do *sprite* esteja sempre apontada para a câmera. Este processo também é conhecido como *billboarding* e o polígono correspondente é denominado de *billboard* (McReynolds, 1999).

figura 3.1 – Um *sprite* é normalmente implementado por um polígono, cuja normal aponta sempre para o observador e com uma textura com transparência. Neste exemplo pode-se reparar que, independente do ponto de vista, o cactos é sempre igual.

As técnicas de *billboarding* combinadas com alfa e animações podem ser usadas para representar fumaça, fogo, *fog*, nuvens e outros fenômenos não sólidos. Há várias formas de *billboards*, sendo as mais comuns:

- Alinhadas ao plano de visão (bons para *sprites* circulares, como partículas);
- Orientados ao ponto de vista (bons para fumaça, fogo, nuvens e outros fenômenos cujas aleatoriedades disfarçam bem as distorções provocadas por este tipo de *billboard*);
- Axiais (bons para árvores e outros objetos de natureza cilíndrica);

Em (Dally, 1996) estende-se o conceito de *sprite* para um objeto representado por uma esfera envolvente (*bounding sphere*). Várias vistas deste objeto são pré-calculadas para diversas posições de câmera ao redor da esfera. Estas imagens são armazenadas numa estrutura chamada de árvore delta (*Delta Tree*). Para cada posição de câmera calcula-se um polígono inscrito na projeção da esfera sobre o plano de projeção da câmera e escolhe-se a imagem mais adequada para esta posição. Este modelo tem como grande desvantagem o fato do centro de projeção do objeto ter que ser sempre o centro da esfera, fazendo com que objetos mal distribuídos ao redor deste ponto sejam deformados.

O trabalho de Pulli (1997) utiliza uma malha simplificada para representar um objeto 3D com poucos polígonos. Associado a esta malha, há um conjunto de imagens que são usadas como texturas e aplicadas sobre a geometria simplificada.

Os *Sprites with Depth* (Shade, 1998) permitem aumentar o poder de representação dos *sprites* realizando deslocamentos ortogonais dos *texels* da textura que os representam. O método utiliza um algoritmo de dois passos para calcular a cor dos *pixels* da imagem sendo gerada a partir da textura fonte do *sprite*. O primeiro passo consiste em gerar um mapa de profundidade intermediário através de um mapeamento que utiliza uma transformação 2D sobre o mapa de profundidade da imagem fonte. No segundo passo, cada *pixel* da imagem desejada passa por uma transformada homográfica (projeção perspectiva sobre um plano), sendo que as coordenadas resultantes são usadas para indexar o mapa de profundidade calculado no primeiro passo. Os valores de deslocamento encontrados são finalmente multiplicados pelo ponto epipolar da imagem sendo formada e adicionada ao resultado da homografia. Estas coordenadas são usadas para indexar a cor dos *pixels* de destino.

Uma abordagem mais recente para as LDI's (*Layered Depth images*) pode ser encontrada em (Bayakovski, 2002), onde as imagens são representadas através das cores dos *pixels* e uma tabela com a distância de cada *pixel* até a superfície que está sendo vista no mesmo. A vantagem desta representação sobre o LDI padrão é que as imagens podem representar objetos mais realistas, com características que tornam complexa sua visualização em tempo real. Além disso, a complexidade da visualização se torna apenas proporcional ao tamanho da imagem e não à complexidade geométrica sendo representada. Na implementação de Bayakovski (2002) utilizam-se duas maneiras de representar tais dados:

- *DepthImages* com texturas simples: Conjunto de imagens com seus respectivos mapas de profundidade. Estas imagens podem compor um objeto através de *Box-Texture* (faces de um *bounding box*) ou através de *Generalized Box Textures* (texturas posicionadas arbitrariamente sobre o objeto);
- *OctreeImages*: Representação através de um volume (*Binary Volumetric Octree*), na forma de *octree* (Hunter, 1978), dizendo se um *voxel* está ocupado ou não. Para otimizar tal estrutura, a *octree* é antes de mais nada convertida para a forma "*breadth-first traversal linkless*", onde cada nó é representado por um simples *byte* cujos *bits* indicam se o seu sub-cubo correspondente deve também ser dividido. Informações de cores são armazenadas num conjunto de imagens de referência, que são obtidas pela projeção dos *voxels* representados no plano das imagens pré-definidas.

Jakulin (2000), procurando representar vegetação através de sprites, apresenta uma outra extensão, onde se usam vários planos para representar a folhagem e fazendo um *blending* entre as texturas, dependendo da posição de onde se encontra o observador.

## 3.3 Impostores

Os impostores (Maciel, 1995) consistem também em métodos eficientes para representar objetos através de imagens. A idéia é representar um objeto tridimensional através de um *sprite*, porém diferentemente que no conceito padrão, estes objetos estão realmente definidos como modelos geométricos e são

calculados durante a aplicação corrente e projetados num plano como textura com transparência.

Impostores são *billboards* gerados em tempo real que distorcem a imagem de maneira semelhante ao que ocorreria com a geometria real do objeto. Devido à própria natureza dos impostores, os *billboards* "orientados ao ponto de vista" são os mais adequados. A imagem de textura de um impostor também pode ser tratada em tempo real para simular determinados efeitos (p. ex. imagens fora de foco para um efeito de profundidade de campo).

Na prática, um impostor deve ser reusado por vários pontos de vista suficientemente próximos (explorando a coerência quadro-a-quadro). Por esta razão, impostores são mais adequados para objetos estáticos pequenos (ou suficientemente distantes). Objetos lentos à grande distância são igualmente bons candidatos para esta técnica. Os testes para verificar se um determinado impostor é ainda válido para o ponto de vista corrente são uma questão fundamental. O presente trabalho trata exatamente desta questão quando propõe os impostores com relevo.

Schaufler (1995) e (1997) descreve os impostores como métodos adequados para minimizar o número de vezes em que se deve visualizar um determinado objeto que possui certa complexidade geométrica e que é, portanto, caro para o *pipeline* de tempo real (Forsyth, 2001). Sob este ponto de vista, os impostores são uma espécie de *buffer* para estes objetos, de forma que sua visualização é reaproveitada enquanto ainda "serve" para uma determinada posição do observador, como foi discutido na seção 2.2.3.

O tamanho da imagem gerada para estes objetos pode ser proporcional ao tamanho que ocupam na tela de projeção. Assim, se estão muito distantes, estes objetos são renderizados numa resolução pequena; porém, na medida em que se aproximam do observador e, portanto, aumentam em tamanho no plano de visualização, estas imagens são refinadas para resoluções maiores. Em (Damon, 2003) descreve-se uma forma de implementar impostores utilizando técnicas de renderização em memória de vídeo, recurso que está disponível nas placas gráficas mais recentes, podendo aumentar o número de objetos sendo representados com este método, desde que haja memória de vídeo suficiente.

Impostores podem conter a profundidade associada a cada *pixel*. Neste caso, Schaufler (1997) passa a chamar este tipo de impostor de *nailboard*. Entretanto,

na presente tese, não se faz distinção entre estes dois termos. A profundidade armazenada em impostores pode ser utilizada no *Z-Buffer* do *pipeline*, de forma que o objeto possa ser penetrado por outros elementos, não havendo problemas de oclusão. Esta profundidade pode ser armazenada no próprio canal alfa da imagem, deixando-se apenas um *bit* reservado para indicar se o *texel* é transparente ou não.

Schaufler (1998) implementa um impostor com mapa de profundidade utilizando camadas de polígonos: o *sprite* consiste numa série de polígonos mapeados com texturas correspondentes às camadas respectivas. Dependendo da informação de profundidade presente no *sprite*, um *pixel* é desenhado ao nível de profundidade que lhe corresponda. O uso de transparência é fundamental neste caso.

## 3.4 Texturas com Relevo

A técnica de *displacement-mapping* (Cook, 1984) permite modificar a geometria de uma superfície tridimensional a partir de um mapa de profundidade, podendo assim criar superfícies complexas utilizando apenas um conjunto de imagens aplicadas sobre superfícies. Poder-se-ia então criar um sprite com *displacement-mapping*, para dar profundidade ao polígono. Esta operação necessita, no entanto, de uma malha de polígonos refinada, uma vez que deforma a geometria no sentido da normal de cada ponto da superfície (figura 3.2 (b)). Além disso, esta técnica não possui uma solução adequada para um tratamento em tempo real, uma vez que não possui uma formulação que permite uma transformação direta da textura para a coordenada da tela (vários pontos da geometria podem ser projetados no mesmo *pixel* da imagem final).

A técnica de *bump-mapping* (Catmull, 1974), por outro lado, possui esta formulação direta, não precisa de uma malha de polígonos refinada, mas não permite modelar a geometria de um objeto, pois somente manipula as normais de cada ponto da superfície, alterando apenas o seu sombreamento (figura 3.2 (a)). Justamente por não alterar a geometria do objeto, o *bump-mapping* é incapaz de corrigir problemas relacionados à paralaxe. A técnica de mapeamento de texturas com relevo possibilita representar a geometria de uma superfície tridimensional e possui uma formulação direta para o mapeamento, podendo ser implementada por

*hardware* e, portanto, podendo ser utilizada em aplicações que exigem tratamento em tempo real.

O mapeamento de texturas com relevo (Oliveira, 2000b), ao contrário do *bump-mapping*, permite criar primitivas geométricas, baseadas unicamente em imagens com profundidade. Esta técnica é utilizada na implementação do presente trabalho para resgatar objetos pré-renderizados, utilizando o *Z-Buffer* previamente armazenado, possibilitando dar-lhes a profundidade e tentar garantir uma coerência de paralaxe do objeto em relação à cena.

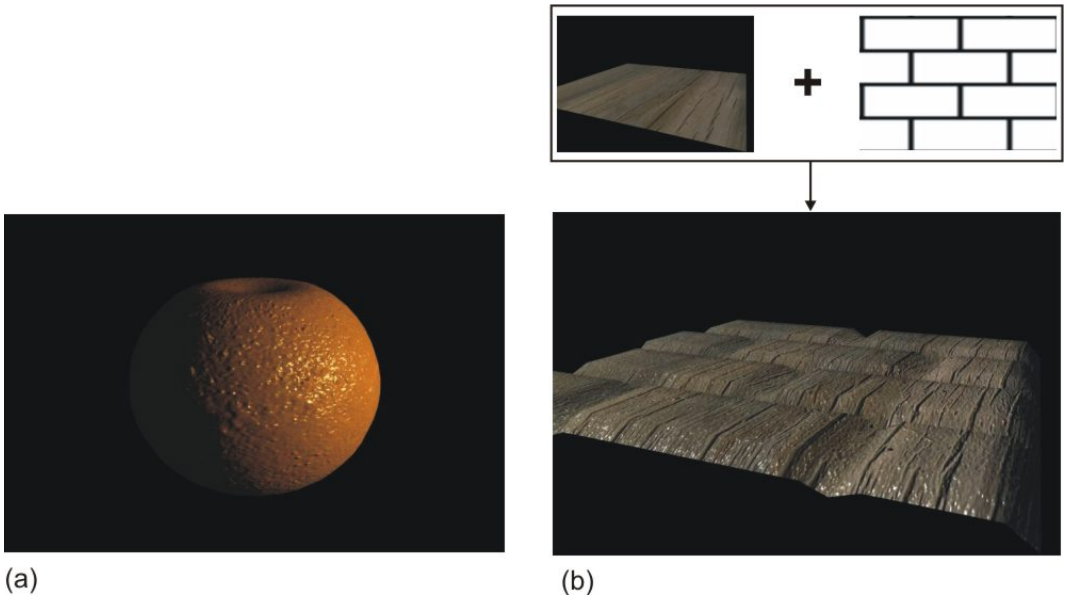



figura 3.2 – Em (a) o objeto possui um *bump-mapping* que lhe dá a aparência de ser rugoso. Perceba-se que o contorno do objeto não é deformado, uma vez que esta técnica apenas altera o sombreamento da superfície. Em (b) se está aplicando a textura de ladrilhos ao plano como *displacement-mapping*. Pode-se perceber que a geometria está sendo de fato alterada.

O método de texturas com relevo consiste numa fatorização da equação de 3D *image warping* (McMillan 1995, McMillan 1997), apresentado na seção 2.4.3, em duas etapas separadas: a primeira, denominada de *pre-warping*, corresponde à transformação *pixel* a *pixel*, proporcional ao valor da disparidade generalizada; a segunda, nada mais é do que um mapeamento de textura convencional, que corresponde à transformação perspectiva.

O *pre-warping* é aplicado a imagens que possuem mapa de profundidade para cada *texel* e consiste sobretudo no movimento (*warping*) destes *texels*. Este movimento é realizado de tal forma que procura corrigir ou reduzir o efeito de paralaxe, resultante do deslocamento do observador em relação ao objeto que

possui a textura com relevo. Cabe a esta etapa resolver o problema de preenchimento de espaços vazios que surgem. A etapa seguinte - texturização - realiza as operações de escala, rotação, filtragem e as deformações de perspectiva necessárias para o mapeamento correto nos polígonos da textura com relevo.

De acordo com Oliveira (2000a), uma imagem com profundidade é definida pelo par [$i_d$, $K$], onde $i$:$U' \rightarrow C'$ é uma imagem digital e $K$ é o modelo de câmera associada a $i$. Cada elemento do espaço de cores $C'$ de $i$ possui também um valor escalar que corresponde à distância, no espaço euclidiano, entre o ponto amostrado e um ponto de referência de $K$. No caso de $K$ ser o modelo de uma câmera com projeção perspectiva, este ponto de referência é o centro de projeção da câmera e a imagem é chamada de imagem de projeção perspectiva com profundidade. Entretanto, na implementação proposta por Oliveira (2000a), é utilizado um modelo de câmera ortográfica. Neste caso, o ponto de referência é o plano da imagem e a imagem correspondente é chamada de imagem de projeção paralela com profundidade. A figura 3.3 mostra os dois modelos de câmera.

A princípio, a imagem *i* pode ser vista como uma textura RGBA (*Red*, *Green*, *Blue* e *Alpha*), onde os canais RGB armazenam a cor deste *texel* e o canal *alpha* armazena a profundidade do mesmo. No entanto, fazendo-se isto, não é possível calcular uma iluminação correta para o objeto com a textura, pois não se pode reconstituir a normal original para este ponto. Portanto, na implementação feita, armazena-se em RGB a normal que o ponto sendo representado pelo *texel* possui na superfície geométrica, seguindo a indicação de Oliveira (2000a). Existe uma segunda imagem RGB que possui a cor de cada *pixel*, sem a sua iluminação difusa (o cálculo de iluminação é feito por *hardware* e é implementado através de um *Pixel Shaders*, na linguagem Cg, como será visto no capítulo 5).

Além das duas imagens, para se representar o objeto por uma textura com relevo é necessário o modelo da câmera ortográfica que lhe corresponde. Também é necessário definir um polígono, que é visto pelo *pipeline* como uma entidade geométrica da cena e sobre o qual a textura com relevo é aplicada.

A estrutura que armazena um objeto do tipo textura com relevo e o seu modelo de câmera é da seguinte forma:

*Tipo Relief_Object*

*{*

*Imagem_RGBA: Mapa_Normais_Profundidade;*

*Imagem_RGB: Mapa_de_cor;*

*Camera: Câmera_fonte;*

*Polígono: Plano_do_Relief_Object;*

*}*

*Tipo Camera*

*{*

*Vetor3D a; // Dimensão horizontal da câmera*

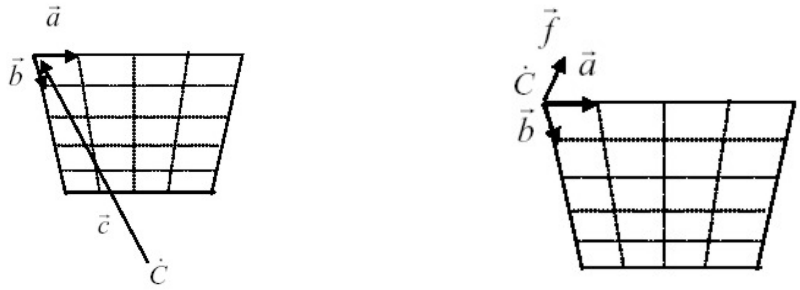
*Vetor3D b; // Dimensão vertical da câmera*

*Vetor3D c; // Centro de Projeção da câmera (projeção paralela)*

*Vetor3D f;* // $\vec{f} = \left|\vec{a} \times \vec{b}\right|$

*Vetor3D C;* // $\vec{c} = \dot{C} - Posição\_Corrente\_da\_camera$

*}*




(a) (b)

figura 3.3 – (a) corresponde a um modelo de câmera perspectiva, onde os vetores $\vec{a}$ e $\vec{b}$ formam a base para o plano da imagem. O comprimento destes vetores correspondem às medidas de cada pixel no espaço euclidiano, $\dot{C}$ é o centro de projeção e o ponto de referência de *K* e $\vec{c}$ é o vetor cuja direção é dada pelo centro de projeção e a origem do plano da imagem. (b) corresponde ao modelo de câmera ortográfica, onde o ponto de referência $\dot{C}$ coincide com a origem do plano da imagem e o vetor $\vec{f}$ é ortogonal ao plano da imagem e possui módulo unitário.

A profundidade armazenada no canal alpha deve estar quantizada para valores inteiros que estão entre 0 (pontos pertencentes à face frontal do *bounding box* do modelo) e 254 (pontos pertencentes à face de traz do *bounding box* do modelo), com valores intermediários para pontos que estão entre as duas faces. Reserva-se um valor (no caso, 255) para representar pontos vazios da textura com relevo. Esta limitação de valores ocorre porque o alpha de um RGBA é do tamanho de um byte. Caso isto não fosse feito, dever-se-ia utilizar uma matriz de pontos flutuantes, fazendo com que o tempo de transferência destes valores para a memória de textura fosse consideravelmente maior (Oliveira, 2000a).

A fatorização descrita inicialmente pode também ser interpretada da seguinte forma: a imagem fonte sofre primeiro uma pré-deformação (*pre-warping*) que depende das informações de profundidade de cada pixel e da posição do observador destino. Esta pré-deformação é feita no próprio plano de projeção da imagem para o observador fonte. Feito isto, utiliza-se a técnica padrão de mapeamento de textura para projetar a imagem deformada sobre o plano correspondente ao observador destino (figura 3.4).

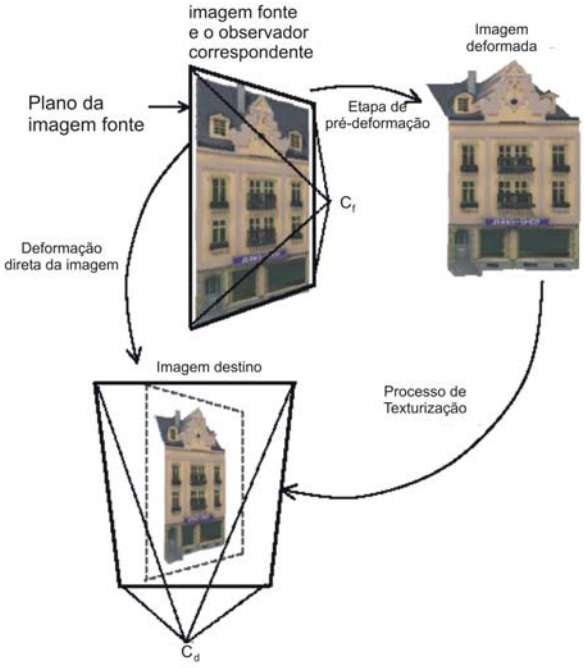



figura 3.4 – Etapas da fatorização da equação de McMillan (figura retirada de (Oliveira, 2000a) )

A etapa de *pre-warping*, por sua vez, corresponde ao seguinte problema: encontrar o par ($u_i$, $v_i$) para cada pixel ($u_f$, $v_f$) de forma que ao ver a imagem deformada a partir do observador destino $C_d$, este novo par corresponda ao local onde ($u_f$, $v_f$) deveria estar no plano de projeção do observador fonte (figura 3.5).

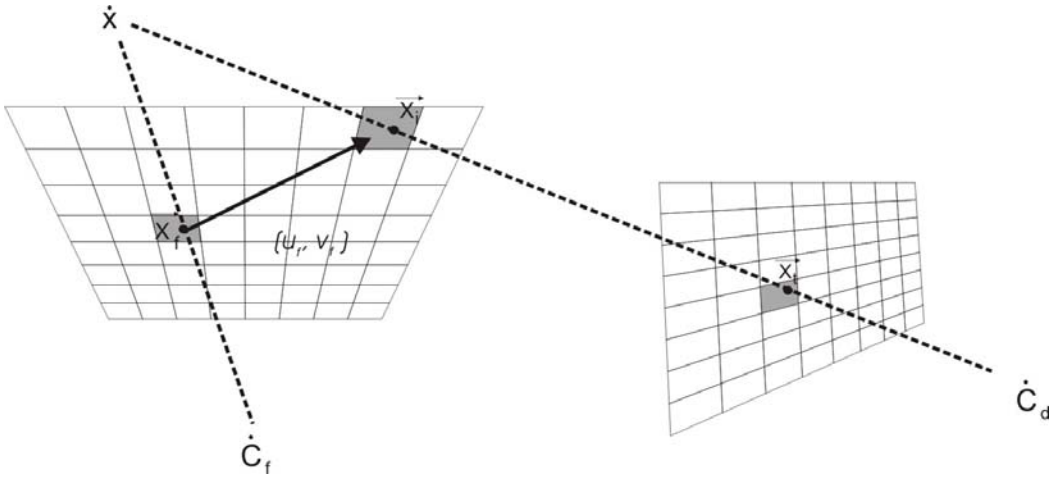



figura 3.5 – A etapa de *pre-warping* pode ser vista como uma transformação sobre o pixel $x_f$ sobre o plano da imagem do observador fonte de forma que corresponda à projeção de $\dot{x}$ sobre este plano para o observador destino.

Como foi discutido no capítulo 2.4.3, a equação de 3D *image warping* que descreve o deslocamento relativo de um *pixel* possui, em relação ao observador destino, dependência apenas da sua posição final. Desta maneira, escolhendo-se adequadamente valores para $\vec{a}_d$, $\vec{b}_d$ e $\vec{c}_d$ uma série de simplificações podem ser feitas na equação de McMillan, chegando-se à seguinte formulação reduzida para a etapa de *pre-warping*:

$$u_i = \frac{u_f - k_1 \partial(u_f, v_f)}{1 + k_3 \partial(u_f, v_f)} \tag{3-1}$$

e $$v_i = \frac{v_f - k_2 \partial(u_f, v_f)}{1 + k_3 \partial(u_f, v_f)} \tag{3-2}$$

onde $k_1 = \frac{\vec{f}.(\vec{b} \times \vec{c})}{\vec{a}.(\vec{b} \times \vec{c})}$, $k_2 = \frac{\vec{f}.(\vec{c} \times \vec{a})}{\vec{b}.(\vec{c} \times \vec{a})}$, $k_3 = \frac{\vec{f}.(\vec{a} \times \vec{b})}{\vec{c}.(\vec{a} \times \vec{b})}$, sendo que *a*, *b*, *c* e *f* são os vetores ilustrados na figura 3.3. As formulações (3-1) e (3-2) são denominadas de equações de *pre-warping* para texturas com relevo.

Da fatorização destas equações, verifica-se que, apesar da etapa de *pre-warping* ser da mesma natureza bidimensional da imagem, pode também ser vista como um processo unidimensional, já que existe total independência entre as coordenadas $u_i$ e $v_i$, ou seja, para determinar $u_i$ não é necessário o valor de $v_f$ e para determinar $v_i$ não é necessário o valor de $u_f$. Isto implica na possibilidade de realizar o *warping* horizontal independentemente do *warping* vertical, podendo-se fazer um processo linear de *warping*, aplicando-o primeiro sobre as linhas e depois sobre as colunas. Esta independência possibilitou que no presente trabalho fossem implementadas diversas versões da etapa de *pre-warping*, conforme se apresenta na seção 8.2. Também se pode notar que quando $\partial(u_f, v_f) = 0$ a etapa de p*re-warping* não deforma a imagem original ($u_i = u_f$ e $v_i = v_f$), tornando o 3D *image warping* no processo padrão de texturização (segunda etapa do mapeamento de texturas com relevo).

O algoritmo que realiza o mapeamento de texturas com relevo pode ser descrito resumidamente da seguinte maneira:

*Determinar o $\dot{C}_d$ e calcular as constantes $k_1$, $k_2$ e $k_3$;*

*Realizar o pre-warping partindo-se da imagem fonte:*

- *Calcular a posição ($u_d$, $v_d$) para cada pixel ($u_f$, $v_f$), usando as eq.(3-1) e (3-2)*
- *Copiar a cor do pixel da imagem fonte para as posições de pre-warping;*
- *Tratar o conflito de pixels sobre uma mesma área e os buracos surgidos;*

*Renderizar os polígonos aplicando as imagens com pre-warping como textura;*

### 3.4.1 Ordem Compatível com Oclusão

Ao realizar o *warping*, surge o problema de conflito de *pixels* sobre uma mesma área: isto ocorre porque a profundidade de cada *pixel* da imagem pertence a pontos de superfícies com profundidades diferentes no mundo real. Assim sendo, um *pixel* pode deslocar-se mais do que o seu vizinho no processo de *warping* (isto pode ser facilmente visto na equação de McMillan observando-se o parâmetro $\partial(u_f, v_f)$).

Além disso, quando um *pixel* sofre um *warping* para uma determinada posição, mas nenhum outro *pixel* é movido para a posição que era ocupada por ele

anteriormente, surge o problema dos buracos. Caso estes espaços não sejam preenchidos com alguma cor, a imagem resultante do *warping* conterá regiões não pintadas.

Para resolver o problema de conflito de *pixels* (Oliveira, 2000a) sugere que seja utilizado o seguinte algoritmo:

*Determinar a interseção do centro de projeção da imagem destino na imagem fonte*[*]*;*
*Dividir a imagem fonte em quadrantes, usando o ponto de interseção como pivô;*

*Se o Centro de Projeção da imagem fonte estiver atrás do centro de projeção da imagem destino, então para cada quadrante:*

*Começar o warping a partir do ponto de interseção encontrado e avançar em direção às bordas do quadrante;*

*Senão*

*Começar o warping a partir das bordas do quadrante e avançar em direção ao ponto de interseção encontrado;*

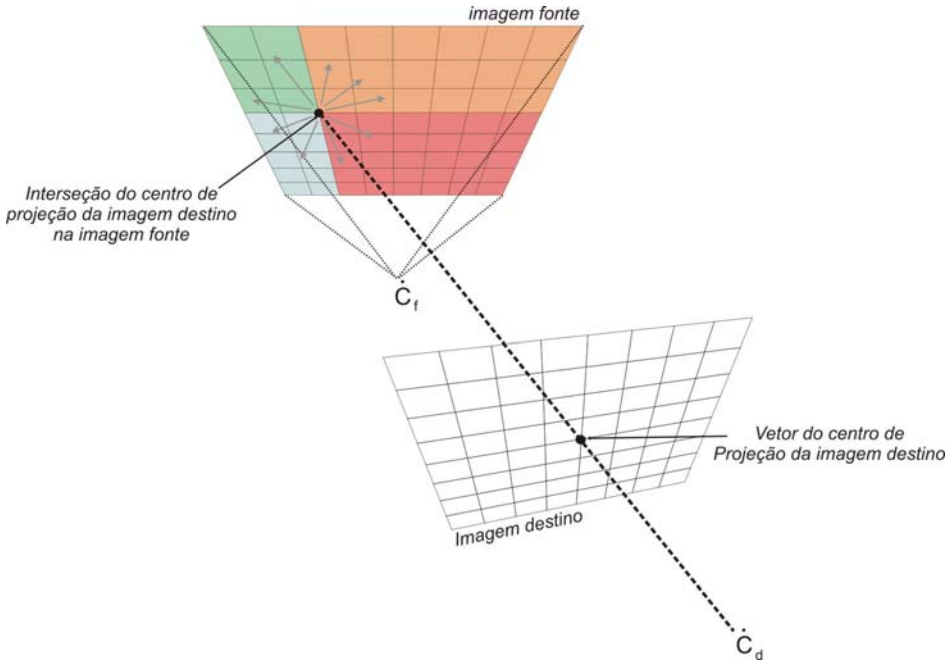



figura 3.6 – Algoritmo do Pintor adaptado para critério de ordem de plotagem dos *pixels* com profundidade, implementado no mapeamento de texturas com relevo. O ponto de interseção do Centro de Projeção da imagem destino com o plano da imagem fonte está ilustrado e cada quadrante criado possui uma cor. Note-se que neste exemplo $\dot{C}_d$

[*] Como foi visto, trata-se do epipolo.

(Centro de Projeção da imagem destino) está atrás $\dot{C}_f$ (Centro de projeção da imagem fonte) e portanto o *warping* ocorre do ponto epipolar em direção às bordas.

A figura 3.6 ilustra este algoritmo, que na verdade é uma adaptação do algoritmo do pintor. Intuitivamente, pode-se dizer que este algoritmo dita a ordem em que os *pixels* da imagem fonte devem sofrer o *warping* para que não seja necessário realizar um teste de profundidade no caso de dois ou mais *pixels* caírem numa mesma posição na imagem destino, como se pode ver na figura 3.7. No exemplo ilustrado da figura 3.6, realiza-se primeiro o *warping* nos *pixels* que estão mais próximos do ponto epipolar, já que estão mais próximos do centro de projeção da imagem destino.

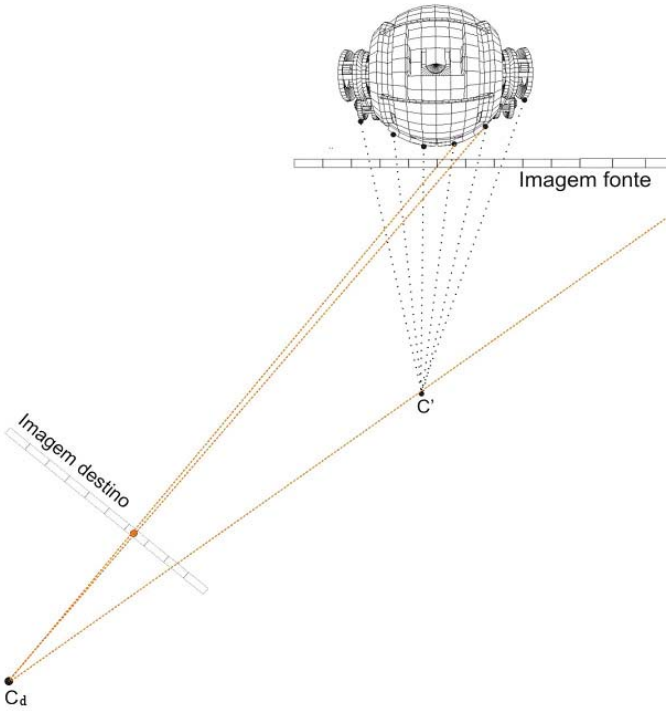



figura 3.7 – Sempre que várias amostras são projetadas sob um mesmo *pixel* da imagem destino, aquela cujo *pixel* da imagem fonte estiver mais distante da linha epipolar é a que está mais próxima do centro de projeção da imagem destino e portanto deverá ser o último a ser plotado.

### 3.4.2 Texturas com relevo em panoramas cilíndricos

Em (ElHelw, 2003), estende-se o conceito de texturas com relevo para imagens mapeadas em cilindros. Uma das aplicações deste trabalho consiste em criar panoramas com detalhes 3D e com uma pequena correção para paralaxe nos

movimentos realizados em seu interior. Como se pode ver na figura 3.8, as etapas deste algoritmo consistem em:

- Transformar uma textura com relevo que possui uma projeção plana para um espaço cilíndrico, através de um mapeamento simples (observe-se que neste caso o centro de projeção da textura, que era no infinito, passa a estar no eixo do cilindro);

- Gerar uma imagem intermediária, através de uma deformação da textura com relevo transformada na etapa anterior utilizando uma equação de *warping* cilíndrico;

- Projetar a imagem intermediária para o cilindro, por meio de mapeamento de textura padrão.

De forma resumida, pode-se dizer que este método faz o *warping* dos *pixels* da textura com relevo, transformando-o para um sistema espacial cilíndrico.

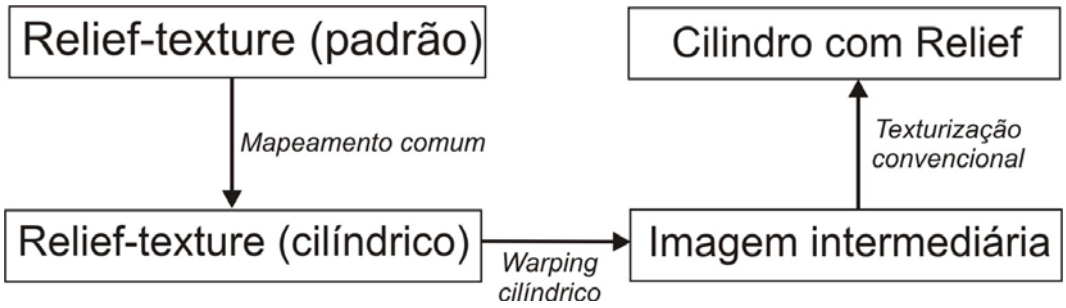



figura 3.8 – Etapas do mapeamento de texturas com relevo para espaços cilíndricos.

### 3.4.3 Representação de objetos 3D utilizando um conjunto de texturas com relevo

Apenas uma textura com relevo não permite que um objeto seja representado por completo. Como não se disponha de dados para partes oclusas do objeto, chega um momento em que o *warping* acumula demasiados erros, invalidando a textura inicial. Para solucionar este problema, Oliveira (1999) e Oliveira (2000a) propõem representar um objeto através de um paralelepípedo, onde cada uma das 6 faces possui uma textura com profundidade gerada para a vista correspondente, através de uma câmera ortogonal. Entretanto, utilizando-se apenas o 3D *image warping* e o critério de oclusão discutido na seção 3.4.1 para cada face separadamente, *pixels* do objeto podem ser projetados em ordem errada,

pois partes da superfície de uma face podem cair no espaço de outra face, dependendo do *warping* que está sofrendo. Desta forma, Oliveira (1999) descreve uma ordem adequada para a escolha das faces a sofrerem o *warping*. Este método possui uma grande vantagem que consiste em permitir que o objeto representado sofra as transformações básicas (translação, rotação e escala) com simples manipulações sobre o seu Centro de Projeção e os vetores $\vec{a}, \vec{b}$ e $\vec{c}$ que definem as câmeras.

Reduzindo-se o problema ao espaço bidimensional, sem perda de generalidade, pode-se descrever o algoritmo da seguinte forma:

Dividir o espaço em 8 regiões, conforme ilustra a figura 3.9. Se o observador estiver numa região ímpar (como é o caso de $O_1$ da figura 3.9), as 3 faces mais próximas e que contribuem para a visibilidade do objeto, são classificadas de frente (face *a* para $O_1$), esquerda (face *b* para $O_1$) e direita (face *d* para $O_1$). Desta forma, aplica-se um *pre-warping* nas faces esquerda e direita sobre o polígono que representa a face da frente. Depois disto, realiza-se o *pre-warping* da face frontal sobre o mesmo polígono, sobre-escrevendo as regiões que foram pintadas pelos *pre-warpings* anteriores.

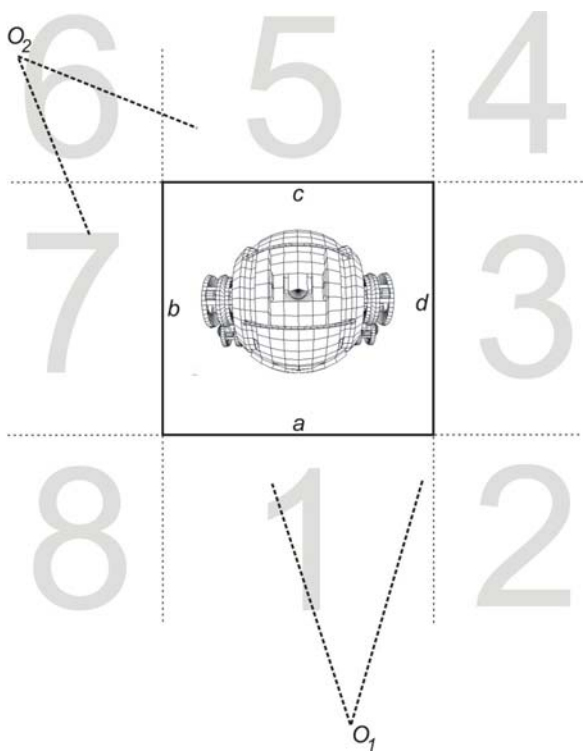



figura 3.9 – Objeto sendo representado por 6 texturas com relevo, visto de cima. O espaço é dividido em 8 regiões, que podem ser de 2 tipos: região par e região ímpar.

Se o observador estiver numa região par (como é o caso de $O_2$ na figura 3.9), as duas faces potencialmente visíveis serão classificadas de face esquerda (face *c* para o caso de $O_2$) e face direita (face *b* para o caso de $O_2$). Desta maneira, realiza-se o *pre-warping* da face esquerda sobre o polígono que sustenta a face da direita e a seguir se realiza o *pre-warping* da face direita sobre seu próprio polígono, sobre-escrevendo as regiões já preenchidas. De igual maneira, realiza-se o *pre-warping* da face direita sobre o polígono que sustenta a face esquerda e a seguir o *pre-warping* da face esquerda sobre seu próprio polígono.