3
Técnicas Propostas

Neste capítulo são descritas as técnicas implementadas. Primeiro é apresentado como é feita a construção e a manutenção do cubo de distância. Em seguida, as soluções baseadas nessa estrutura são discutidas.

3.1
Cubo de Distâncias

O cubo de distâncias é formado por 6 imagens, resultantes da renderização da cena com a câmera orientada em 6 direções diferentes. Isso é feito de forma que os planos de projeção da câmera correspondam às faces do cubo. É usada uma câmera perspectiva com ângulo de abertura de 90^0, fazendo com que a combinação dos 6 frustums resultantes cubra todo o espaço.

Os canais $rgba$ de cada pixel das imagens das faces guardam respectivamente um vetor, que aponta da posição no espaço do mundo referente a esse pixel até a câmera, e o valor da distância dessa posição até a câmera. Isso é feito usando um shader que calcula esses dados. Pixels que não representam nenhuma geometria têm valor $rgba$ igual a $\{1.0, 1.0, 1.0, 1.0\}$.

Os valores de distância guardados do canal a das imagens são normalizados no intervalo [0.0, 1.0] com relação aos valores atuais dos planos *near* e *far*, através da seguinte equação:

$$dNorm = \frac{d - near}{far - near} \tag{3-1}$$

Na equação 3-1, $dNorm$ é o valor de distância normalizado, d é o valor não normalizado e $near$ e far são, respectivamente os valores dos planos de corte.

A Figura 3.1, retirada de [19], ilustra o funcionamento do cubo de distâncias. Em seu trabalho, [19] orientam a câmera nas 6 direções canônicas: X positivo, X negativo, Y positivo, Y negativo, Z positivo e Z negativo. Uma representação visual do cubo de distâncias pode ser vista na parte inferior da Figura 3.1.

O cubo de distâncias é atualizado sempre que a câmera muda de posição. A vantagem de se usar as direções canônicas para orientar a câmera é que

PUC-Rio - Certificação Digital Nº 0812547/CA

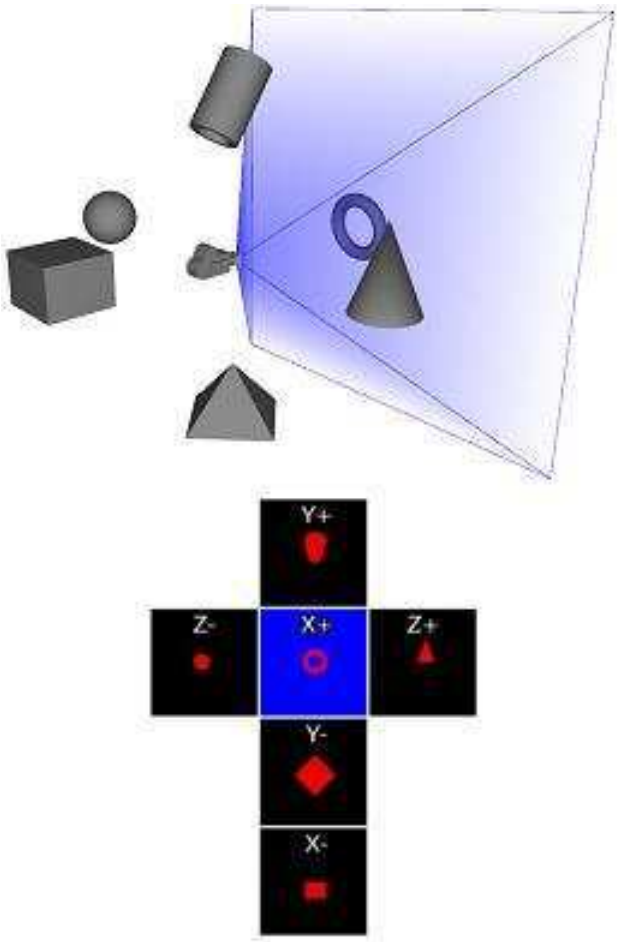

Figura 3.1: Cubo de Distâncias [19].

somente a nova posição dessa é necessária para realizar a atualização. Porém, optamos neste trabalho pela construção do cubo de forma que ele sempre fique orientado com a câmera. Apesar da desvantagem de necessitar também da orientação da câmera, essa abordagem permite obter mais facilmente informações adicionais, como por exemplo a estimativa da distância até o centro da tela.

O algoritmo 3.1 mostra o pseudocódigo que descreve os passos necessários para a atualização do cubo de distâncias.

Pseudocódigo 3.1: Procedimento *atualizaCubo*

```
1 Variáveis:
2  – cubo: cubo de distâncias, representado por um vetor
3 de 6 imagens
4  – camera: câmera da aplicação
5  – camera.pos: posicao atual da câmera
6  – camera.dir: vetor que indica a orientação da câmera
7  – camera.up: vetor up da câmera
8  – camera.right: vetor right da câmera
```

PUC-Rio - Certificação Digital Nº 0812547/CA

```
 9  - camera.fov: ângulo de abertura da câmera
10  - camera.lookAt: matriz que posiciona e orienta a câmera
11  - corFundo: cor de fundo atual
12  - newAt: novo vetor que indica o local para onde a
13  câmera deve apontar
14  - newUp: novo vetor up da câmera
15  ------------------------------------------------------------
16      { guarda o estado atual da aplicação }
17      camPos = camera.pos;
18      camDir = camera.dir;
19      camUp = camera.up;
20      camRight = camera.right;
21      camFov = camera.fov
22      corFundo = corFundo();
23
24      { altera o fov e a cor de fundo }
25      camera.fov = 90^0
26      alteraCorFundo(1.0, 1.0, 1.0, 1.0);
27
28      { ativa o shader responsável por calcular os valores
29        no cubo de distâncias }
30      ativaShader();
31
32      { renderiza a face frontal }
33      alteraAlvoRender(cubo[FACE_FRONTAL]);
34      novoAt = camPos + camDir;
35      novoUp = camUp;
36      camera.lookAt = geraLookAt(camPos, novoAt, novoUp);
37      render();
38
39      { renderiza a face posterior }
40      alteraAlvoRender(cubo[FACE_POSTERIOR]);
41      novoAt = camPos - camDir;
42      novoUp = camUp;
43      camera.lookAt = geraLookAt(camPos, novoAt, novoUp);
44      render();
45
46      { renderiza a face superior }
47      alteraAlvoRender(cubo[FACE_SUPERIOR]);
48      novoAt = camPos + camUp;
49      novoUp = camRight;
```

PUC-Rio - Certificação Digital Nº 0812547/CA

```
50      camera.lookAt = geraLookAt(camPos, novoAt, novoUp);
51      render();
52
53      { renderiza a face de baixo }
54      alteraAlvoRender(cubo[FACE_INFERIOR]);
55      novoAt = camPos − camUp;
56      novoUp = camRight;
57      camera.lookAt = geraLookAt(camPos, novoAt, novoUp);
58      render();
59
60      { renderiza a face direita }
61      alteraAlvoRender(cubo[FACE_DIREITA]);
62      novoAt = camPos + camRight;
63      novoUp = camUp;
64      camera.lookAt = geraLookAt(camPos, novoAt, novoUp);
65      render();
66
67      { renderiza a face esquerda }
68      alteraAlvoRender(cubo[FACE_ESQUERDA]);
69      novoAt = camPos − camRight;
70      novoUp = camUp;
71      camera.lookAt = geraLookAt(camPos, novoAt, novoUp);
72      render();
73
74      { restaura o estado anterior da aplicação }
75      camera.lookAt = geraLookAt(camPos, camPos + camDir, camUp);
76      camera.fov = camFov;
77      alteraCorFundo(corFundo);
78      alteraAlvoRender(TELA);
79      desativaShader();
```

No ínicio do procedimento *atualizaCubo* são guardados alguns parâmetros da câmera e da aplicação (linhas 16 a 22). Em seguida, o ângulo de abertura da câmera é configurado em 90^0 e a cor de fundo é alterada para {1.0, 1.0, 1.0, 1.0}, através da função *alteraCorFundo* (linhas 25 e 26). Dessa forma, qualquer posição do cubo com esses valores representa um local do espaço sem geometria.

Antes do início da renderização das faces do cubo, o shader responsável por calcular os vetores e as distâncias é ativado (linha 30). A seguir são renderizadas todas as faces do cubo de distâncias (linhas 32 a 72). A função *alteraAlvoRender* configura o local onde a cena será renderizada, podendo este

PUC-Rio - Certificação Digital Nº 0812547/CA

ser uma das imagens do cubo ou a própria tela. Para obter maior precisão, são usadas imagens com 32 bits para cada canal. A função *geraLookAt* gera as novas matrizes de *lookAt* com o objetivo de orientar a câmera de acordo com cada face. No final, quando o cubo de distâncias já está completamente atualizado, o estado anterior da aplicação é restaurado (linhas 75 a 79).

Os códigos referentes ao shader usado na linha 30 são mostrados nos algoritmos 3.2 e 3.3.

Pseudocódigo 3.2: *Vertex Shader*

```
1  uniform mat4 worldMatrix;
2  varying vec3 wcPosition;
3
4  void main()
5  {
6      gl_Position = ftransform();
7
8      // Calcula a posição no espaço do mundo do vértice
9      wcPosition = ( OSGWorldMatrix * gl_Vertex ).xyz;
10 }
```

Pseudocódigo 3.3: *Fragment Shader*

```
1  varying vec3 wcPosition;
2  uniform vec3 camPosition;
3  uniform float near;
4  uniform float far;
5
6  void main()
7  {
8      // Calcula o vetor que aponta do fragmento para a câmera
9      vec3 forceDirection = camPosition - wcPosition;
10
11     // Calcula a distância desse fragmento até a câmera
12     float dist = length( forceDirection );
13
14     forceDirection = normalize( forceDirection );
15
16     // Normaliza o resultado para o intervalo [0.0, 1.0]
17     float normalizedDist = ( dist - near ) / ( far - near );
18
19     gl_FragColor = vec4( forceDirection, normalizedDist );
20 }
```

PUC-Rio - Certificação Digital Nº 0812547/CA

Como pode ser visto no algoritmo 3.1, a atualização do cubo de distâncias requer mais 6 passadas de renderização. Como resultado, a aplicação pode sofrer uma perda considerável de desempenho. Para que isso não ocorra, é usada uma resolução menor na renderização das faces. Dessa forma, o cubo de distâncias representa uma amostragem de informações referentes à geometria da cena. [19] dizem que, em geral, para uma dada resolução res do cubo, a resolução de amostragem de um objeto que está a uma distância d da câmera é igual a:

$$resAmos = \frac{2d}{res} \tag{3-2}$$

Por exemplo, se $res = 64 \times 64$, então um objeto localizado a 3.2 metros tem uma resolução de amostragem de 10 cm. A equação 3-2 resume uma das vantagens do cubo de distâncias: objetos próximos da câmera, geralmente mais relevantes, têm naturalmente uma resolução de amostragem maior do que a de objetos que estão distantes.

A posição no espaço do mundo referente ao pixel (x, y) na face i pode ser obtida da seguinte maneira:

$$pos = camPos - vec(x, y, i)dist(x, y, i) \tag{3-3}$$

Na equação 3-3, $camPos$ é a posição da câmera no mundo, $vec(x, y, i)$ é o vetor unitário armazenado nos canais rgb do ponto (x, y) na face i e $dist(x, y, i)$ é distância não normalizada referente ao valor do canal a.

3.2 Ajuste Automático da Velocidade de Navegação da Ferramenta Voar

A velocidade de navegação está relacionada à escala dos ambientes a serem explorados. Ambientes maiores exigem uma velocidade maior, enquanto o oposto é mais conveniente quando em escalas menores.

Em diversas aplicações, a escala do mundo virtual não varia muito e é bem conhecida, sendo possível pré-estabelecer uma velocidade de navegação fixa. Esse é o caso de muitos jogos eletrônicos, por exemplo. Ambientes multiescalas como o do SiVIEP, entretanto, exigem que haja uma forma de se estimar a escala atual em que a câmera se encontra a fim de se poder ajustar corretamente a velocidade de navegação.

[19] usam os valores armazenados no cubo de distância para obter tal estimativa. Eles chegam à conclusão de que o menor valor de distância produz o melhor resultado na prática, especialmente quando não se sabe nenhuma informação adicional sobre as características geométricas da cena. Com base nisso, foi inicialmente proposto que a velocidade de navegação para a ferramenta *Voar* fosse ajustada de acordo com a seguinte equação:

PUC-Rio - Certificação Digital Nº 0812547/CA

$$V_{nav} = k\ minDist \tag{3-4}$$

Na equação 3-4, V_{nav} é a velocidade de navegação ajustada, $minDist$ é o menor valor (não normalizado) presente no cubo de distâncias e k é uma constante de proporcionalidade. O valor de $minDist$ é determinado através da varredura das 6 faces do cubo. Como será visto nas próximas seções, esse valor também é usado por outras técnicas. Para evitar que cada consulta por $minDist$ incorra em uma nova varredura, essa última só é realizada uma vez ao final de cada atualização do cubo.

A constante k tem como efeito aumentar ou diminuir a aceleração aplicada na câmera. Valores maiores de k fazem com que a câmera desacelere mais rapidamente ao se aproximar de um objeto, enquanto que valores menores fazem o oposto. Observamos que essas duas situações provocavam desconforto e desorientação em alguns usuários: quando k é muito alto, ao se afastar da geometria a câmera acelera rápido demais, produzindo um efeito similar ao de um teleporte, e o usuário perde a noção de localização. Valores baixos de k podem aumentar consideravelmente o tempo necessário para se chegar ao ponto desejado, criando uma situação onde a navegação se torna uma operação tediosa. Dessa forma, k deve ser escolhido de forma a equilibrar esses dois extremos.

Devido à dificuldade em estabelecer qual valor de k é o melhor para todos os usuários, decidiu-se deixar que esses pudessem defini-lo manualmente, através do botão de scroll do mouse. Entretanto, isso não foi suficiente para eliminar os problemas descritos anteriormente.

Ao navegarem muito próximos de um objeto, alguns usuários aumentavam o valor de k para poderem ir mais rápido, mas esqueciam de reajustá-lo quando a câmera já estava distante da geometria. Consequentemente, acabavam caindo no mesmo caso em que k é muito alto.

As situações descritas revelam uma desvantagem em se usar somente $minDist$ como estimativa para o ajuste da velocidade: quando muito pequeno, $minDist$ funciona como um freio mesmo nos momentos em que o usuário deseja se mover mais rápido. Por exemplo, ao navegar em um corredor, o usuário experimenta lentidão na navegação uma vez que a câmera pode se encontrar próxima às paredes. Da mesma forma, a aproximação tangencial de qualquer objeto provoca o mesmo efeito. Isso ocorre pois, na maioria das vezes, $minDist$ não expressa o real desejo do usuário, que pode ser por exemplo chegar até o fim de um corredor em uma plataforma, ou ir de um ponto a outro entre duas camadas de um reservatório. É conveniente então que outro tipo de estimativa seja usada.

PUC-Rio - Certificação Digital Nº 0812547/CA

Na técnica proposta por [18], a velocidade de navegação é ajustada com base na distância em que a câmera se encontra do ponto de destino escolhido, como pode ser visto na equação 2-1. Com isso, os autores conseguem um ajuste que eles creem se adaptar melhor às necessidades dos usuários.

O problema tratado aqui, contudo, consiste em ajustar a velocidade da ferramenta *voar*, a qual não necessita que o usuário escolha um ponto de destino. Se fosse possível determinar tal ponto, poderia-se usar a abordagem de [18].

Com esse objetivo tentamos usar o local apontado pela câmera, ou seja, o ponto central da tela, como destino. Isso é razoável uma vez que esse ponto representa momentaneamente o local em que o usuário deseja chegar. Dessa forma, a velocidade de navegação passou a ser ajustada usando *centroDist*, a distância da câmera até o centro da tela, ao invés de *minDist*.

O uso de *centroDist*, entretanto, resultou em um comportamento estranho. O movimento da câmera deixou de ser suave e passou a apresentar picos de velocidade, dando a impressão que esta parava ou acelerava instantaneamente.

A análise do gráfico da Figura 3.2 permite entender o motivo disso. Esse gráfico mostra o comportamento das curvas das estimativas *minDist* e *centroDist*, para um mesmo caminho percorrido pela câmera em um intervalo de tempo de aproximadamente 6 segundos.

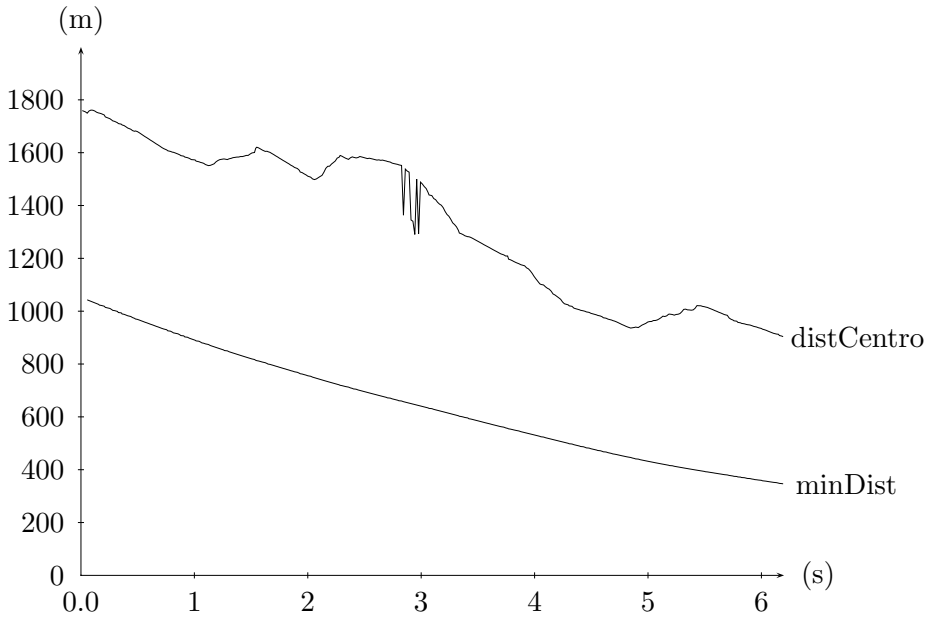

Figura 3.2: Gráfico do comportamento das curvas *minDist* e *distCentro*.

Percebe-se que a curva de *minDist* é suave, enquanto que a de *centroDist*

PUC-Rio - Certificação Digital Nº 0812547/CA

é mais ruidosa, apresentando em alguns pontos valores de pico que distoam completamente do comportamento geral da curva. O motivo disso é a liberdade que a ferramenta *voar* proporciona ao usuário: este pode orientar a câmera para qualquer direção, a qualquer momento. Em um dado instante, o ponto central da tela pode estar localizado em um objeto distante da câmera. O usuário pode então, quase que instantaneamente, girar a câmera para um objeto que está ao seu lado fazendo com que o valor de *centroDist* caia abruptamente. Isso reflete então no ajuste da velocidade, criando uma desaceleração brusca. O contrário também configura um problema: se, por exemplo, a câmera estiver dentro de uma plataforma e o usuário sem querer aponta aquela em direção a um ponto externo, a câmera sofrerá uma rápida aceleração e será jogada para fora da plataforma. Esses efeitos não ocorrem quando se usa *minDist* como estimativa, pois este independe da orientação da câmera.

As situações provocadas pelos valores de pico da curva *centroDist* podem ser evitadas se esta for suavizada. Isso pode ser feito usando-se uma *média exponencial móvel (MEM)* para gerar uma nova curva:

$$MEM_i = MEM_{i-1} + A\ (centroDist_i - MEM_{i-1}) \tag{3-5}$$

Na equação 3-5, MEM_i é o valor suavizado de $centroDist_i$ no instante i, MEM_{i-1} é o valor suavizado no instante $i-1$ e A é uma constante que influencia em quão suave será a nova curva e a rapidez com que essa converge para os valores de *centroDist*. Quanto menor A, mais suave a curva e maior é o tempo necessário para a convergência. O valor máximo de $A = 1$ equivale a curva original. Na Figura 3.3 são mostrados os resultados da suavização de um intervalo da curva *centroDist* (representada pelo traçado em negrito) para três diferentes valores de A.

A suavização da curva *centroDist* permite que essa seja usada no ajuste de velocidade. Entretanto, ainda resta um último problema: o caso em que a câmera está próxima de uma geometria, mas está sendo apontada para um local que se encontra a uma distância muito maior. Quando isso acontece, a velocidade de navegação tende a aumentar, mesmo que suavemente, até convergir novamente para *centroDist*. Em algumas situações isso é indesejável do ponto de vista do usuário. Por exemplo, o caso em que a câmera está dentro da plataforma, mas se quer visualizar a parte externa dessa última. Para fazer isso, o usuário deve navegar para fora da plataforma e então apontar a câmera na direção daquela. Durante essa operação, a câmera pode ficar apontada para algum ponto distante da plataforma por um período de tempo suficiente para que a velocidade aumente demais.

Isso ocorre pois a percepção de proximidade fornecida por *minDist* não

PUC-Rio - Certificação Digital Nº 0812547/CA

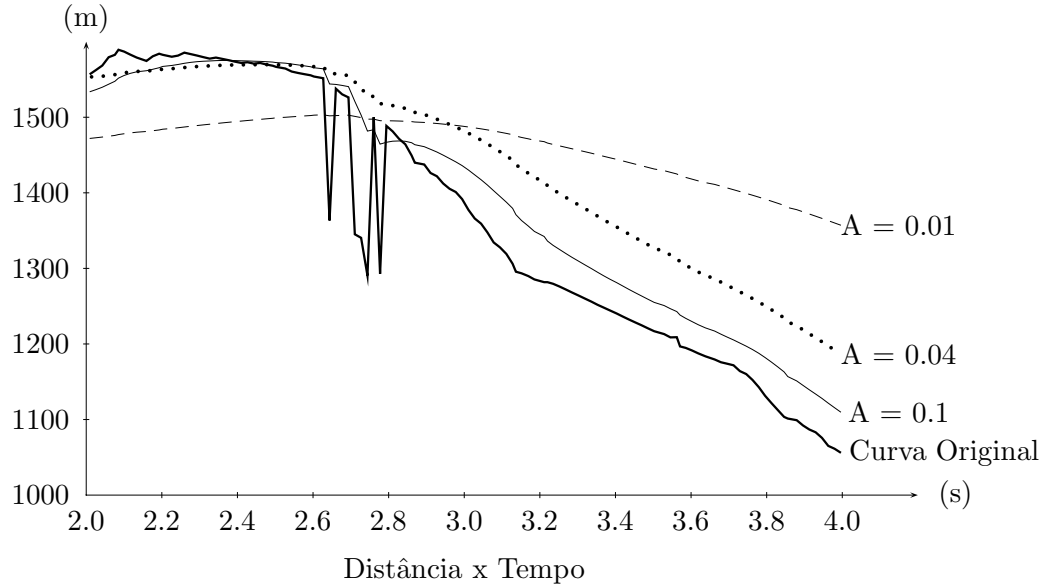

Figura 3.3: Efeito da aplicação da *média exponencial móvel* sobre um curva.

está mais presente. Nesses casos, $minDist$ acaba atuando como um freio, impedindo que a câmera se mova para longe muito rápido.

Para resolver o problema mencionado anteriormente, valores de $centroDist$ maiores do que uma certa proporção de $minDist$ são descartados. Por exemplo, no SiVIEP considera-se que valores maiores do que $10 \times minDist$ não são usados para ajustar a velocidade. Dessa forma, $minDist$ fornece um critério para se decidir quando um valor de $centroDist$ pode ou não ser considerado um ponto fora da curva.

Essa última solução corresponde ao híbrido de uso das estimativas $minDist$ e $centroDist$: por um lado, $centroDist$ faz com que o ajuste de velocidade seja mais próximo ao desejo do usuário. Já $minDist$ atua como um freio para os casos em que a velocidade aumentaria exageradamente. Pode-se pensar também nessa solução como uma forma de se determinar automaticamente a constante k da equação 3-4. Considerando-se $k = \frac{centroDist_{suavizado}}{minDist}$, obtem-se uma constante de multiplicação que varia suavemente entre 1, caso em que $centroDist_{suavizado} = minDist$, e 10, caso em que $centroDist_{suavizado} > minDist$. Dessa forma, não é mais necessária a intervenção do usuário e, por isso, problemas causados por ajustes ruins feitos por ele são eliminados.

3.3 Ajuste Automático dos Planos de Corte

Como foi discutido na Seção 1.2.2, o ajuste correto dos planos de corte é um requisito importante para a visualização correta de modelos 3D. A má configuração desses parâmetros pode levar a problemas que vão desde o corte

PUC-Rio - Certificação Digital Nº 0812547/CA

indevido de objetos até o aparecimento de artefatos em objetos distantes da câmera.

[19] desenvolveram uma técnica de ajuste automático para os planos de corte. Essa consiste em utilizar a informação *minDist* presente no cubo de distâncias a fim de selecionar valores ótimos para os parâmetros *near* e *far*. A ideia deles é manter a geometria vísivel sempre no intervalo entre *near* e *far*. A cada quadro, os planos de corte são atualizados de acordo com as seguintes equações:

$$n = \begin{cases} \alpha n & \text{se } minDist < An \\ \beta n & \text{se } minDist > Bn \\ n & \text{caso contrário} \end{cases} \tag{3-6}$$

$$f = Cn \tag{3-7}$$

Na equações 3-6 e 3-7, n é o valor de *near*, f o valor de *far*, $minDist$ é a menor distância (não normalizada) armazenada no cubo de distâncias, α, β, A, B são constantes que indicam quando e como os planos de corte devem ser atualizados. Na implementação de [19], assim como no SiVIEP, os valores $\alpha = 0.75$, $\beta = 1.5$, $A = 2$, $B = 10$ produziram resultados satisfatórios. Por último, C expressa a razão entre os valores de n e f. O valor de C deve ser escolhido de forma que não ocorram cortes em objetos localizados próximos do plano *far*. Ao mesmo tempo, C não pode assumir valores muito altos pois isso provocaria perda de precisão no *depth buffer*. No SiVIEP, C foi fixado em 10000.

A Figura 3.4, retirada de [19], ilustra o modo como as equações 3-6 e 3-7 agem sobre os valores dos planos de corte. No caso *(a)*, o ponto mais próximo da cena, representada por uma estrela, está localizado no intervalo $[An, Bn]$ e dessa forma não é necessária nenhuma modificação dos planos de corte. Esse é o caso ideal. No caso *(b)*, $minDist$ é menor do que An, ou seja, está mais próximo do plano *near*. Os planos de corte são então ajustados para valores menores, de forma a garantir, com certa margem de segurança, que a geometria da cena fique dentro do frustum de visão. Da mesma forma, no caso *(c)* os planos de corte são ajustados para valores maiores, uma vez que $minDist$ está mais distante da câmera ($minDist > Bn$).

Para evitar que os valores *near* e *far* sejam ajustados para zero quando a câmera se aproxima demais de um objeto, é estabelecido um valor mínimo igual a 0.1 para o plano *near*.

Um dos problemas dessa técnica acontece quando a câmera se aproxima muito de um objeto, mas ainda assim é capaz de visualizar outros objetos mais distantes. Devido à proximidade da geometria, o valor de *near* é reduzido.

PUC-Rio - Certificação Digital Nº 0812547/CA

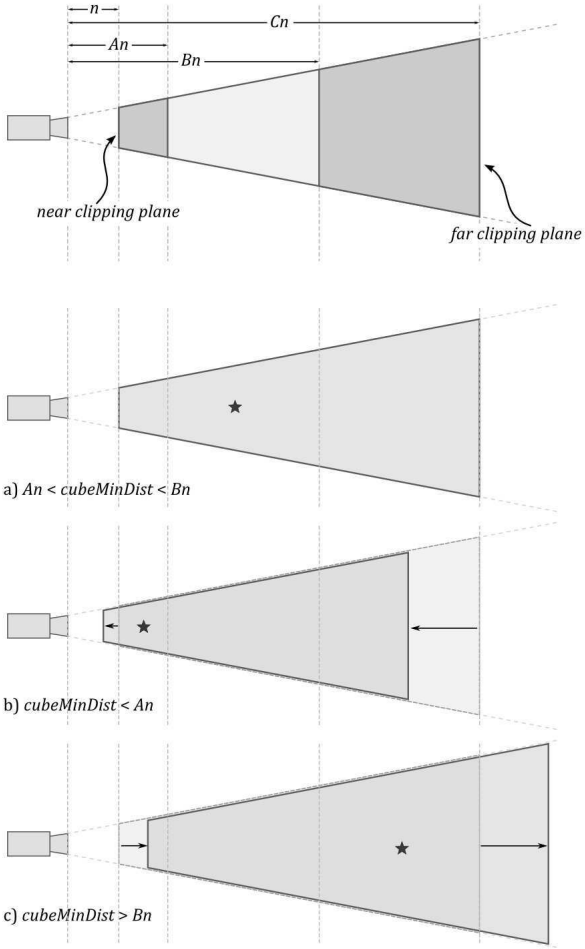

Figura 3.4: Ajuste automático dos planos de corte [19]

Consequentemente, *far* também assume um valor menor e os objetos distantes podem acabar sendo cortados. A Figura 3.5 mostra um caso onde isso acontece. Na Figura 3.5(a), a câmera está localizada perto de uma plataforma e é possível ver à esquerda uma segunda plataforma, perto da linha do horizonte. Na Figura 3.5(b), a câmera foi colocada ainda mais próxima da plataforma. O resultado disso é o desaparecimento da segunda plataforma, uma vez que essa foi cortada pelo plano *far*.

Uma possível solução para esse problema seria usar $maxDist$, o maior valor presente no cubo de distâncias, para ajustar o plano *far* da mesma forma como é feito com o plano *near*. Essa estratégia, entretanto, faz com que a relação $r = \frac{near}{far}$ fique muito alta em alguns momentos, a ponto de produzir artefatos (pontos que ficam piscando na tela). Esse efeito não é aceitável e, portanto, essa abordagem não é usada.

Durante a condução de testes com alguns usuários, percebeu-se que o problema ilustrado na Figura 3.5 não é tão crítico. Ao navegarem pelo ambiente

PUC-Rio - Certificação Digital Nº 0812547/CA

3.5(a): 3.5(b):

Figura 3.5: Problema com o ajuste automático dos planos de corte.

virtual, as pessoas naturalmente têm sua atenção voltada para aquilo que está mais próximo, que por sua vez está sendo exibido corretamente devido ao ajuste automático dos planos de corte. Observamos que, mesmo em momentos em que tal problema ocorria, nenhum usuário notou sua existência.

Por último, é importante observar que o funcionamento correto das equações 3-6 e 3-7 dependem da existência de um número de quadros suficientes que permitam a convergência para os valores ótimos de planos de corte. Por exemplo, se a câmera se move rápido demais na direção de um objeto localizado a 1000 metros a ponto de cobrir essa distância em apenas 3 quadros, o ajuste será feito apenas 3 vezes. Considerando-se um valor inicial de *near* igual a 500 metros, o valor final ajustado será de $near = 500 \times 0.75 \times 0.75 \times 0.75 = 211$ metros, o que acaba provocando o corte do objeto em questão. Dessa forma, os movimentos realizados pelas ferramentas de navegação devem ser os mais suaves possíveis.

3.4 Detecção e Tratamento de Colisão

Impedir que a câmera possa ultrapassar objetos em um ambiente virtual pode ser essencial em determinadas situações. Por exemplo, quando em ambientes imersivos, uma colisão com algum objeto pode resultar em quebra de imersão, fazendo ainda com que o usuário possa ficar desorientado. Outra situação que pode ser citada ocorre quando a visualização envolve o efeito de estereoscopia: nesse caso, a colisão com algum objeto da cena pode causar uma sensação física desagradável nos olhos do usuário.

O ajuste de velocidade descrito na Seção 3.2 produz por si só um comportamento que deveria impedir a ocorrência de colisões: uma vez que a

PUC-Rio - Certificação Digital Nº 0812547/CA

velocidade da câmera é reduzida de acordo com $minDist$ (distância ao ponto mais próximo), no momento em que $minDist = 0$ a câmera deveria parar naturalmente. Entretanto, não é isso que ocorre pois $minDist$ nunca chega a assumir tal valor. Como dito na seção anterior, o menor valor para o plano *near* é 0.1 e, dessa forma, esse valor determina quão próximo a câmera pode chegar perto de um objeto sem ultrapassá-lo. Portanto, mesmo com o ajuste de velocidade é possível ainda ultrapassar os modelos.

Como solução, [19] usam as informações do cubo de distâncias para obter um fator de colisão, correspondente a uma "força" que tem por efeito fazer com que a câmera desvie suavemente dos objetos mais próximos. A ideia é que cada ponto no cubo de distâncias localizado a uma distância menor do que um dado raio r produza uma força de repulsão, dada por:

$$F(x, y, i) = penal(dist(x, y, i))\ norm(pos(x, y, i) - camPos) \tag{3-8}$$

Na equação 3-8, $F(x, y, i)$ é a força de repulsão produzida pelo ponto p referente ao pixel (x, y) da imagem i do cubo de distâncias. O valor $dist(x, y, i)$ é a distância de p à câmera, e está armazenado no canal a do pixel (x, y). O termo $pos(x, y, i)$ é a posição de p no espaço do mundo e $camPos$ é a posição da câmera. A função $norm(v)$ indica o vetor normalizado de v. Logo, na equação 3-8, o termo $norm(pos(x, y, i) - camPos)$ indica o vetor unitário que aponta de p para a câmera. Esse vetor é obtido diretamente dos canais rgb do pixel (x, y). A função *penal* calcula um fator de penalidade de colisão cuja a intensidade depende da distância d que p se encontra da câmera, e é dada por:

$$penal(d) = e^{\frac{(r-d)^2}{\sigma^2}} \tag{3-9}$$

Na equação 3-9, σ é um parâmetro que indica a suavidade do fator de colisão. Quanto maior σ, mais suave é o fator calculado. Considerando a região esférica de raio r e centro na posição da câmera, a equação 3-9 tem como efeito atribuir uma penalidade de colisão que cresce exponencialmente a partir do momento em que o ponto p ultrapassa essa região.

As forças de colisão referentes à equação 3-8 são calculadas uma vez a cada atualização do cubo de distâncias, no momento em que é feita a varredura para determinar o valor de $minDist$. Para cada posição do cubo de distâncias é testado se a distância armazenada no canal a é menor do que r. Se isso for verdade, então o ponto correspondente a essa posição está dentro da região de colisão e uma força de penalidade é calculada para ele. Uma vez que todos os pontos do cubo de distâncias foram testados, as forças são combinadas em uma única, dada pela equação a seguir:

PUC-Rio - Certificação Digital Nº 0812547/CA

$$F_{colisao} = \frac{1}{6\ cuboRes^2} \sum_{x,y,i} F(x,y,i) \tag{3-10}$$

Na equação 3-10, *cuboRes* é a resolução do cubo de distâncias. Por exemplo, para uma resolução de 64×64, $cuboRes = 64$.

A força calculada pela equação 3-10, quando aplicada à câmera faz com que esta desvie suavemente dos objetos que se encontram em seu caminho (Figura 3.6). Em conjunto com a ferramenta *voar*, essa força pode ser usada para obter um comportamento similar ao modo de navegação assistida descrito por [5], [33] e [17]: o usuário pode navegar pelo ambiente sem ter que se preocupar em manter um caminho livre de colisão, uma vez que o próprio sistema se encarrega dessa tarefa. Essa abordagem é diferente da usada por [19], visto que esses utilizam $F_{colisao}$ em conjunto com a técnica de *POI*. Essa é mais restritiva, não permitindo que o usuário tenha o controle total da câmera enquanto essa se movimenta.

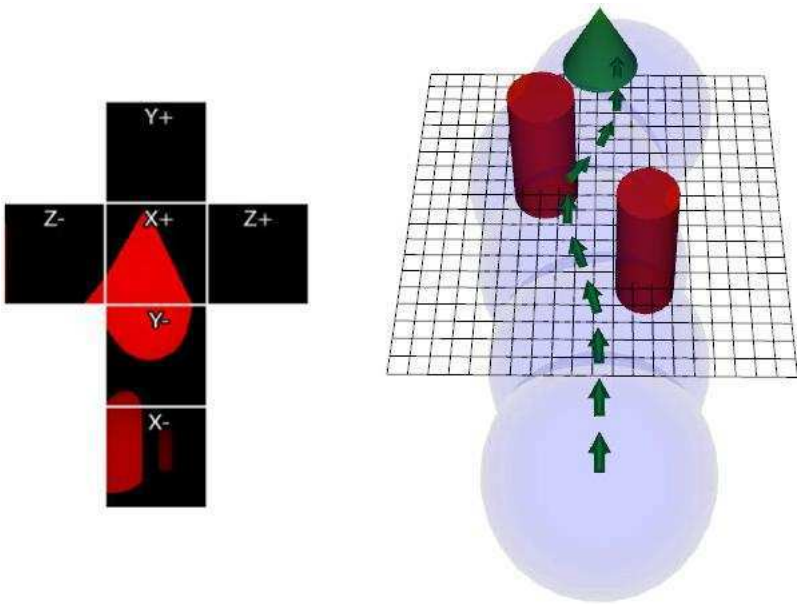

Figura 3.6: Efeito da aplicação de $F_{colisao}$ sobre a câmera [19]

$F_{colisao}$ é aplicada à câmera a cada quadro e influencia na posição da câmera de acordo com a seguinte equação:

$$camPos_t = camPos_{t-1} + dir\ V_{nav}\ \Delta t + F_{colisao} \tag{3-11}$$

Na equação 3-11, *camPos* é a posição da câmera, *dir* é o vetor unitário que indica a direção especificada pelo usuário, V_{nav} é a velocidade de navegação calculada como descrito na Seção 3.2 e Δt é o intervalo entre dois quadros da aplicação.

O modo como $F_{colisao}$ influencia no comportamento da câmera depende de dois parâmetros: o raio r da esfera que define a região do espaço sujeita ao

PUC-Rio - Certificação Digital Nº 0812547/CA

tratamento de colisão e o fator de suavidade σ. Valores altos de r permitem uma margem de segurança maior com relação à colisão, mas acabam impedindo a câmera de atingir pontos de menores escalas. Valores muito baixos de σ podem fazer com que a câmera seja repelida mais bruscamente, enquanto que valores muito altos podem fazer com que a câmera seja repelida com pouca intensidade, vindo a colidir com o ambiente. Dessa forma, a escolha desses parâmetros deve ser feita com cuidado. Para o SiVIEP foram testadas diversas combinações e observou-se que, para os tipos de modelos que esse se propõe a visualizar, os valores $r = 6$ m e $\sigma = 2$ forneceram um efeito aceitável.

3.5 Examinar com Centro de Rotação Automático

A ferramenta *examinar* permite que um objeto ou local qualquer da cena possa ser inspecionado. Basicamente, o seu funcionamento corresponde a rotacionar a câmera em torno de um ponto, chamado de *centro de rotação*.

A localização do *centro de rotação* é fundamental para o funcionamento correto da ferramenta *examinar*. Se mal especificado, a câmera pode assumir comportamentos que do ponto de vista do usário, parecem confusos. [9] descrevem algumas situações em que isso ocorre. Por exemplo, a Figura 3.7 ilustra três desses casos.

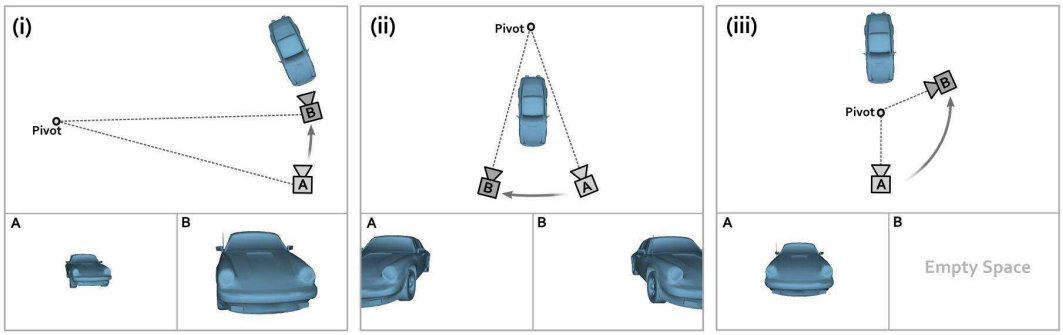

Figura 3.7: Problema relacionados ao centro de rotação [9].

No caso (i) da Figura 3.7, o centro de rotação, nomeado *pivot* na figura, está localizado fora do campo de visão do usuário, distante do objeto a ser inspecionado. Ao usar a ferramenta *examinar*, é feita uma rotação em torno de *pivot* que, do ponto de vista matemático é correta. Entretanto, para o usuário, essa operação resulta em um efeito similar ao de um *zoom* no objeto, quando na verdade o comportamento que esse espera é o de uma rotação em torno do mesmo. Esse problema é pior quando *pivot* se encontra a uma distância grande do modelo: quanto maior essa distância, maior a velocidade angular da câmera e, consequentemente, maior será o erro sentido pelo usuário. Observações informais mostraram que essa situação acontecia com certa frequência no

PUC-Rio - Certificação Digital Nº 0812547/CA

SiVIEP, em parte devido à característica multiescala e o tamanho elevado do ambiente que esse se propõe a visualizar.

Nos casos (ii) e (iii), *pivot* encontra-se no ângulo de visão da câmera, mas está localizado fora do objeto de interesse. Em (ii), *pivot* está além do modelo e a rotação em torno dele tem o efeito de uma operação de *pan*. No caso (iii), o *pivot* está posicionado em frente ao modelo e a rotação leva a um ângulo de visão onde esse não pode ser visto.

Analisando as situações descritas, pode-se concluir que elas ocorrem basicamente devido a dois motivos:

1. O centro de rotação está localizado totalmente fora do ângulo de visão da câmera. Mais especificamente, *pivot* não está no centro da tela.

2. O centro de rotação está localizado em um ponto mais distante e/ou que não correspondente ao objeto a ser examinado.

O funcionamento correto da ferramenta *examinar* depende de impedir que as situações 1 e 2 ocorram. Dessa forma é conveniente identificar em que momento e por que essas acontecem.

O SiVIEP conta com um botão na interface do aplicativo que, ao ser selecionado, permite que um novo centro de rotação seja escolhido. Essa é uma ferramenta de apoio e deve ser usada pelo usuário em conjunto com o modo de navegação *examinar*. Observou-se que algumas pessoas, mesmo aquelas com mais experiência no uso do aplicativo, acabavam em algum momento esquecendo de escolher o centro de rotação adequado antes de iniciar a rotação em torno do objeto de interesse. O motivo disso é que, após um certo tempo usando a aplicação, o usuário se despreocupa com as questões de interface e naturalmente transfere sua atenção para o ambiente virtual. A necessidade de trocar entre diferentes ferramentas de interação promove uma quebra momentânea dessa imersão. Em especial, esse efeito foi sentido por alguns usuários ao trocar da ferramenta *voar* para a de *examinar*. Após realizar essa operação, os usuários naturalmente tentavam primeiro rotacionar a câmera em torno do objeto localizado em sua frente, sem reajustar devidamente o centro de rotação. Mesmo os usuários que não esqueceram de realizar essa última operação relataram terem se sentido incomodados em ter de explicitamente fazê-la.

Como abordagem para esses problemas pensou-se inicialmente em criar uma forma de obrigar o usuário a ajustar o centro de rotação. Por exemplo, ao ativar a ferramenta *examinar*, o ponto correpondente ao primeiro clique com o mouse pode ser usado como novo *pivot*. A câmera teria então sua posição reajustada de modo que o novo *pivot* corresponda ao centro da tela, assim como

PUC-Rio - Certificação Digital Nº 0812547/CA

é feito na ferramenta manual de centro de rotação. Essa abordagem, entretanto, sofre de outro problema: nem sempre o ponto correspondente ao primeiro clique do mouse é um ponto válido de geometria ou mesmo está localizado sobre o objeto a ser examinado, o que poderia gerar situações confusas. Além disso, o reposicionamento da câmera, de forma a colocar o novo *pivot* como centro da tela, nesse caso é algo inesperado pelo usuário e pode acarretar em desorientação em alguns casos.

A solução final encontrada foi estabelecer automaticamente um centro de rotação no momento da ativação da ferramenta *examinar*. Usando o ponto correspondente ao centro da tela como novo *pivot*, é possível estabelecer um comportamento natural do ponto de vista do usuário. Tudo o que este precisa fazer é apontar a câmera para o objeto de interesse e então selecionar a ferramenta *examinar*. Esperar que isso aconteça é razoável, uma vez que na maioria das vezes os usuários só tomam a decisão de examinar um objeto quando este está visível na tela. Entretanto não é garantido que esse esteja localizado exatamente na direção do ponto central da tela. De fato, pode acontecer do centro de rotação ser mapeado para um objeto que está atrás daquele que realmente se quer examinar. Nesse caso, o usuário experimentaria o efeito de uma operação de *pan*, como mostrado em (ii) da Figura 3.7. Em outra situação, o ponto central da tela pode não corresponder a nenhum ponto válido de geometria e, dessa forma, é impossível determinar *pivot*. Para resolver esses problemas, a menor distância não normalizada presente na face frontal do cubo de distâncias, $minFront$, é usada.

No caso em que o ponto central não é válido, *pivot* é ajustado para o ponto que está a uma distância de $minFront$ na frente da câmera. Procedendo dessa forma, a velocidade ângular da rotação da ferramenta *examinar* será condizente com a escala em que a câmera se encontra, impedindo que essa realize movimentos muito rápidos.

A estimativa $minFront$ também é usada quando o centro de rotação é mapeado para um ponto distante, localizado atrás do objeto. Nesse caso, $minFront$ pode atuar como um limitador, no sentido de identificar e corrigir situações que possam levar a um comportamento estranho da câmera. A ideia é não permitir que *pivot* seja ajustado para um ponto cuja a distância seja maior do que $k \times minFront$. Essa solução não resolve o problema de maneira definitiva, mas minimiza de forma adequada seus efeitos. Para o SiVIEP, o valor de $k = 10$ forneceu resultados satisfatórios.

PUC-Rio - Certificação Digital Nº 0812547/CA

3.6 Restrições da Câmera

Algumas das técnicas apresentadas anteriormente correspondem à aplicação de certas restrições na câmera. Por exemplo, o ajuste automático de velocidade proposto na Seção 3.2 age no sentido de controlar a velocidade e a suavidade de movimentação da câmera, impedindo que essa acelere ou desacelere bruscamente. O tratamento de colisão descrito na Seção 3.4 impede que os modelos sejam atravessados ao mesmo tempo em que aplica uma força que desvia a câmera para fora dos obstáculos. No ajuste dos planos de corte da Seção 3.3, o valor para o plano *near* não pode ser menor do que 0.1 m a fim de que esse não seja ajustado para zero. Resumindo, o objetivo dessas restrições é impedir situações que possam resultar em desorientação para o usuário e, ao mesmo tempo, evitar que a câmera entre em um estado instável.

Além das mencionadas, foi criada uma última restrição cuja função é impedir que a câmera se afaste demais da cena. Isso é feito estabelendo-se uma caixa envolvente que engloba toda a cena e que é invisível ao usuário. O movimento da câmera fica então restrito aos limites dessa região.

As dimensões dessa caixa devem ser tais que o usuário possa visualizar toda a cena a partir de um ponto qualquer, localizado nos limites da caixa. Por esse motivo, o tamanho dessa deve ser maior do que a caixa envolvente mínima da cena. Atualmente, essa caixa é determinada manualmente pelo projetista responsável por criar a cena.

Do ponto de vista do usuário, essa nova restrição tem como objetivo principal impedir que esse se afaste demais da cena, a ponto de que não seja mais possível enxergar nenhuma geometria. Entretanto, a caixa envolvente desempenha também uma outra função, não menos importante: garantir o estado correto do cubo de distâncias descrito na Seção 3.1.

Como discutido na Seção 3.1, a placa gráfica é usada para calcular as distâncias de cada ponto desenhado na tela até a câmera. Dessa forma, se a câmera estiver localizada a uma distância muito grande a ponto da geometria da cena não gerar nenhum pixel na tela, então o cubo construído nesse momento não conterá nenhuma informação que possa ser usada pelas técnicas apresentadas anteriormente.

Esse problema é ainda pior quando se considera a diferença entre as resoluções da imagem final gerada na tela e das faces do cubo de distâncias. Como o objetivo desse último é fornecer apenas uma estimativa da escala da cena, sua resolução geralmente é bem menor do que aquela que é visível ao usuário. Por exemplo, no SiVIEP usa-se uma resolução de 64×64 para o cubo de distâncias, enquanto a da tela chega facilmente a 1024×1024. O efeito

PUC-Rio - Certificação Digital Nº 0812547/CA

disso é que o cubo de distância pode se tornar inválido mesmo quando ainda é possível visualizar a cena, uma vez que a distância necessária para que nenhum pixel seja gerado na construção do cubo é menor do que a necessária para que a mesma situação ocorra na tela. Ao definir uma região máxima de atuação da câmera, pode-se garantir que esses problemas não aconteçam e que o cubo de distâncias sempre contenha as informações necessárias.

3.7 Seta Indicadora

Observou-se que algumas pessoas ao usarem a ferramenta *voar* ficavam perdidas, sem saber onde a cena estava. Elas em algum momento apontavam a câmera para um local completamente vazio e depois não eram capazes de encontrar a direção correta novamente. Isso ocorria pois não havia qualquer informação que pudesse indicar a localização da cena.

Para contornar esse problema, foi introduzida uma seta que aponta na direção em que a cena se encontra. Essa seta é mostrada somente quando nenhuma geometria é mostrada na tela. A Figura 3.8 mostra uma situação onde isso acontece.

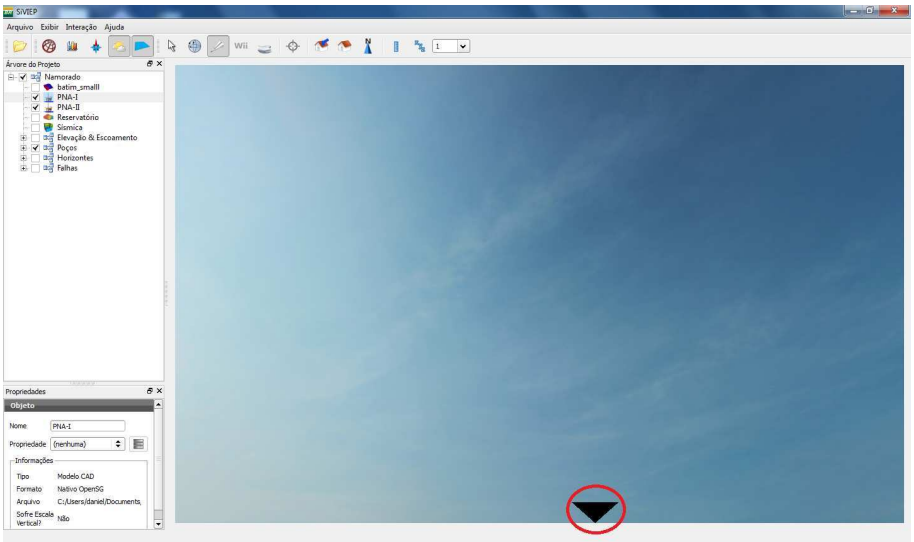

Figura 3.8: Seta indicadora

Na Figura 3.8 a câmera está sendo apontada para o céu, que no SiVIEP é tratado apenas como uma textura de fundo e, por isso, não representa uma geometria válida. A cena está localizada abaixo da câmera e, dessa forma, a seta indicadora, destacada na figura por um círculo, aponta nessa direção.

Para identificar o momento em que a seta indicadora deve ser exibida, verifica-se se a face frontal do cubo de distâncias não possui nenhum ponto de geometria válida, ou seja, se todos os pixels dessa face tem o valor 1.0. Se

PUC-Rio - Certificação Digital Nº 0812547/CA

isso for verdade, então a câmera está sendo apontada para um local onde não há nenhum objeto. A partir de então, é calculado o local da tela onde a seta deve aparecer. Essa é disposta de forma a sempre acompanhar a borda da tela, apontando para um ponto p da cena. Para p pode-se se usar qualquer ponto. Por exemplo, pode-se usar o ponto da cena que está mais próximo da câmera, o qual pode ser obtido através do cubo de distâncias, ou então o ponto central da caixa envolvente mínima da cena.

O uso de setas indicadoras é muito comum em jogos eletrônicos. Essa é uma maneira intuitiva de tentar indicar ao usuário a localização de alguma informação importante na cena. Durante testes de usabilidade feitos com o SiVIEP, a seta indicadora apareceu para alguns poucos usuários. Esses, ao notarem sua presença, foram capazes de rapidamente entender sua finalidade.

PUC-Rio - Certificação Digital Nº 0812547/CA