Daniel Ribeiro Trindade

Técnicas de Navegação 3D Usando o Cubo de Distâncias

DISSERTAÇÃO DE MESTRADO

DEPARTAMENTO DE INFORMÁTICA
Programa de Pós–graduação em Informática

Rio de Janeiro
Março de 2010

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Ribeiro Trindade

Técnicas de Navegação 3D Usando o Cubo de Distâncias

Dissertação de Mestrado

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática da PUC-Rio

Orientador: Alberto Barbosa Rapaso

Rio de Janeiro
Março de 2010

Pontifícia Universidade Católica
do Rio de Janeiro

Daniel Ribeiro Trindade

Técnicas de Navegação 3D Usando o Cubo de Distâncias

Dissertação apresentada como requisito parcial para obtenção do grau de Mestre pelo Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC-Rio. Aprovada pela Comissão Examinadora abaixo assinada.

Alberto Barbosa Rapaso
Orientador
Departamento de Informática — PUC-Rio

Marcelo Gattass
Departamento de Informática — PUC-Rio

Waldemar Celes Filho
Departamento de Informática — PUC-Rio

Luciano Pereira dos Reis
Petrobras

Rodrigo Penteado Ribeiro de Toledo
Petrobras

José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico — PUC-Rio

Rio de Janeiro, 26 de Março de 2010

Daniel Ribeiro Trindade

Graduou-se em Engenharia de Computação na Universidade Federal do Espírito Santo em 2008. Desde 2008 trabalha no laboratório de Computação Gráfica da PUC-RIO (TecGraf) desenvolvendo sistemas de realidade virtual e visualização científica.

Ficha Catalográfica

Trindade, Daniel Ribeiro

Técnicas de Navegação 3D Usando o Cubo de Distâncias / Daniel Ribeiro Trindade; orientador: Alberto Barbosa Raposo. — Rio de Janeiro : PUC-Rio, Departamento de Informática, 2010.

v., 81 f: il. ; 29,7 cm

1. Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática.

Inclui referências bibliográficas.

1. Informática – Tese. 2. Ambientes Multiescala. 3. Navegação 3D. 4. Detecção de Colisão. 5. 3DUI. 6. Realidade Virtual. I. Raposo, Alberto Barbosa. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Aos amigos, novos e antigos.

Agradecimentos

Ao TecGraf, por me proporcionar um ambiente agradável de trabalho e condições para que eu realizasse essa pesquisa.

Ao Rodrigo Marques, pessoa extremamente capaz e responsável, por ter me convencido a fazer mestrado na PUC, ter sido amigo e um agradável colega de apartamento e ter me ajudado com conversas, sugestões e idéias.

Ao Alberto, que foi um excelente orientador, sempre atencioso e disposto a tirar minhas dúvidas, além de um ótimo revisor.

Ao Pablo, Thiago, Henrique, Marcela, Eduardo e todos os outros integrantes das equipes do SiVIEP e do Environ, todos profissionais extremamente competentes e sempre dispostos a ajudar.

Aos amigos, novos e antigos, que tiveram a capacidade de compreender o fato de eu ter sumido por alguns meses.

À minha família, que apoiou minha vinda ao Rio e sempre esteve disposta a me ajudar.

À CAPES, pelo suporte financeiro.

Resumo

Trindade, Daniel Ribeiro; Raposo, Alberto Barbosa. **Técnicas de Navegação 3D Usando o Cubo de Distâncias**. Rio de Janeiro, 2010. 81p. Dissertação de Mestrado — Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

A utilização de visualizadores 3D é algo cada vez mais comum em diversos ramos de atividades. O surgimento de novas tecnologias, com o resultante aumento do poder de processamento dos computadores atuais, tornou possível a criação de ambientes virtuais 3D maiores e mais ricos em detalhes. No entanto, a navegação em ambientes 3D, especialmente os ambientes multiescala, ainda é um problema para muitos usuários. O objetivo deste trabalho é propor soluções para alguns problemas de navegação 3D, a fim de melhorar a experiência de uso nesse tipo de aplicação. Nesse sentido, são apresentadas técnicas que permitem ajustar automaticamente a velocidade de navegação, os planos de corte e o ponto de centro de rotação. É proposta também uma solução para a detecção e tratamento de colisão entre a câmera e os modelos da cena, além de uma técnica que visa impedir que os usuários fiquem perdidos quando nenhum objeto da cena é visualizado. Essas soluções são baseadas na construção e manutenção de uma estrutura chamada de cubo de distâncias (cube map, no original em inglês), que fornece informações sobre a localização espacial dos pontos da cena em relação à câmera. Atualmente em desenvolvimento no Tecgraf/PUC-Rio, o SiVIEP (Sistema de Visualização Integrado de Exploração e Produção) é um visualizador voltado para profissionais da área de exploração e produção de petróleo, que serviu para a detecção e entendimento dos problemas mencionados e para a validação das soluções implementadas.

Palavras–chave

Ambientes Multiescala. Navegação 3D. Detecção de Colisão. 3DUI. Realidade Virtual.

Abstract

Trindade, Daniel Ribeiro; Raposo, Alberto Barbosa. **3D Navigation Techniques Using the Cube Map**. Rio de Janeiro, 2010. 81p. MSc Thesis — Department of Computer Science, Pontifícia Universidade Católica do Rio de Janeiro.

The use of 3D viewers is becoming common in several activities. The appearance of new technologies, with the resulting increase in processing power, made possible the creation of larger and richer 3D virtual environments. However, the navigation in 3D environments, especially the multiscale ones, is still a problem for many users. The goal of this work is to propose solutions to some 3D navigation problems in order to improve the user experience with this kind of application. In this sense, techniques to automatically adjust the navigation speed, the clipping planes and the rotation center are presented. It is also proposed a solution for the detection and treatment of collision between the camera and the scene, and a technique that aims to prevent users from getting lost when no scene object is visualized. These solutions are based on the construction and maintenance of a structure called cube map, which provides information about the spatial location of the scene points relative to the camera. Currently in development at Tecgraf/PUC-Rio, the SiVIEP (Integrated Visualization System for Exploration and Production) is a viewer aimed at professionals in the area of oil exploration and production that was used to detect and understand the mentioned problems, and also for validating the implemented solutions.

Keywords

Multiscale Environments. 3D Navigation. Collision Detection. 3DUI. Virtual Reality.

Sumário

Lista de figuras

Lista de tabelas

A melhor maneira de se ser feliz é contribuir para a felicidade dos outros.

Confúcio, *Ditado Popular.*

1
Introdução

Diversos ramos da engenharia e da geociências fazem uso de aplicações de visualização 3D. Estas permitem que os profissionais daquelas áreas possam planejar melhor suas ações através da análise de objetos 3D que representem estruturas do mundo real. Essa etapa tem se tornado cada vez mais importante, pois possibilita reduzir custos e maximizar a produção.

Na área de exploração e produção de petróleo isso não é diferente. Atualmente existem diversas aplicações de visualização 3D, cada uma delas com o enfoque em uma etapa diferente do processo de extração de petróleo. Por exemplo, um determinado aplicativo pode ser capaz de mostrar modelos CAD de engenharia, como plataformas de petróleo ou navios, mas não permite a visualização de reservatórios ou poços de petróleo simultaneamente. Ainda não existe uma aplicação que permita a visualização integrada desses modelos.

Entretanto, há uma necessidade crescente de ferramentas desse tipo. Com elas seria possível ter uma visão gerencial integrada dos modelos usados nas diversas etapas do projeto, permitindo a realização de um planejamento mais preciso das ações que devem ser tomadas.

Para atender a essa demanda, o TecGraf e a Petrobras começaram a trabalhar conjuntamente no projeto SiVIEP (Sistema de Visualização Integrado de Exploração e Produção). Seu objetivo é criar uma aplicação para visualizar de forma integrada os diferentes modelos 3D presentes no processo de extração e produção de petróleo. Apesar de ainda estar em estágio de desenvolvimento, a aplicação já é capaz de mostrar vários tipos de modelos e possui algumas ferramentas de navegação (Figura 1.1).

Tal integração é necessária e representa uma grande vantagem em relação aos aplicativos presentes no mercado, mas ela acaba criando novos problemas que geralmente não existiam quando cada modelo era visualizado separadamente. O primeiro, e mais evidente, tem relação com o desempenho: agora uma quantidade muito maior de geometria deve ser carregada na placa gráfica e isso pode impactar negativamente na taxa de quadros por segundo que a aplicação é capaz de exibir. Nesse sentido, técnicas de aceleração em renderização devem ser implementadas.

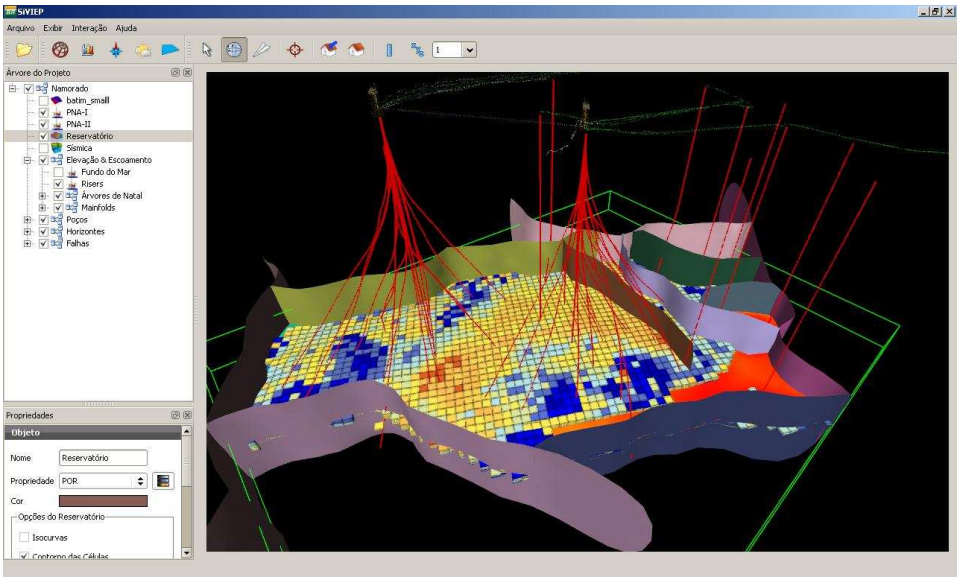


Figura 1.1: SiVIEP: vizualização de um campo de exploração de petróleo.

Outro problema não menos importante tem relação com a *navegação* nesse novo tipo de ambiente virtual. Por navegação, entende-se como o *processo cognitivo de determinar e seguir um caminho, baseado no conhecimento e informações existentes no ambiente* [Jul and Furnas, 1998].

No mundo real, a navegação é algo natural ao ser humano. As pessoas precisam a todo momento se deslocar de um ponto a outro e na maioria dos casos não há dificuldade em determinar como isso pode ser feito. Estudos mostram que os seres humanos são capazes de formar mapas cognitivos de seus ambientes para uso posterior na tarefa de navegação [Darken and Sibert, 1993; Goldin, 1982; Howard, 1981]. Diversas informações presentes no ambiente podem ser usadas para a construção desse mapa. Por exemplo, uma pessoa pode se basear no formato geométrico das construções ou mesmo em suas cores. Em um grande número de cidades, as ruas têm placas com os nomes das mesmas. Em alguns casos, é possível usar até mesmo o cheiro de um local como pista de sua localização.

Quando se trata de ambientes virtuais 3D, entretanto, as coisas podem ser bem diferentes e mais complicadas. O ato de navegar deixa de ser algo natural e, para alguns usuários, pode até mesmo resultar em experiências desagradáveis e frustrantes. Não é incomum ouvir frases do tipo 'onde eu estou?' ou 'o objeto sumiu da tela'.

As causas para esses problemas são várias e têm sido objeto de estudo de muitos pesquisadores ao longo dos últimos anos. Por exemplo, em [Jul and Furnas, 1998] é identificado um problema que os autores chamam de *desert fog*: condição na qual o ambiente imediatamente próximo não contêm informações suficientes que permitam tomar decisões de navegação. Esse tipo de situação

pode ocorrer em qualquer tipo de ambiente virtual, mas é mais comum em *ambientes multiescala (AMEs)*.

Ambientes multiescala são ambientes onde a informação pode existir em diferentes níveis de detalhe. Em outras palavras, em AMEs é possível visualizar simultaneamente objetos com diferentes escalas, indo desde um parafuso até um campo de petróleo com dezenas de quilômetros de extensão.

A integração de modelos 3D presente no SiVIEP proporciona justamente um ambiente virtual com essa característica. Por exemplo, um reservatório de petróleo é muito maior do que uma plataforma de extração, que por sua vez é bem maior que uma peça de maquinário localizada em seu interior. Como consequência, o SiVIEP acaba herdando alguns dos problemas de navegação inerentes a AMEs.

A característica multiescala, entretanto, não é a única fonte de problemas. Quando mal projetadas, as ferramentas de navegação podem acabar criando situações confusas. No artigo *Safe 3D Navigation* [Fitzmaurice et al., 2008], por exemplo, os autores chegam a algumas conlusões interessantes sobre o comportamento de algumas pessoas ao usarem aplicações 3D:

- Nem sempre elas conseguem compreender como uma determinada ferramenta de navegação funciona ou, pior, a entendem incorretamente e acabam esperando um outro tipo de comportamento ao usá-la.
- Geralmente são necessárias mais do que uma única ferramenta para explorar o mundo virtual. Entretanto, um número considerável de usuários usa apenas uma ferramenta para todos os fins.
- Uma vez perdidas na cena, as pessoas geralmente tentam resolver a situação usando a mesma ferramenta de quando se originou o problema. O resultado disso é elas acabam não conseguindo reverter a situação, e pioram o problema.

Nesse sentido, o objetivo deste trabalho é propor e implementar soluções para os problemas de navegação que foram identificados durante o desenvolvimento do SiVIEP. Esses serão abordados com mais detalhes nas próximas seções, assim como também será feita uma descrição do SiVIEP e de suas ferramentas de navegação.

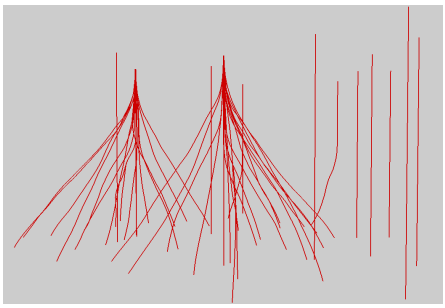
1.1 SiVIEP

A navegação em um determinado local depende fortemente do tipo do ambiente que esse possui [Darken and Sibert, 1993]. Por exemplo, as decisões de uma pessoa serão provavelmente bem diferentes se ela estiver se deslocando no meio de um deserto ao invés de uma cidade.

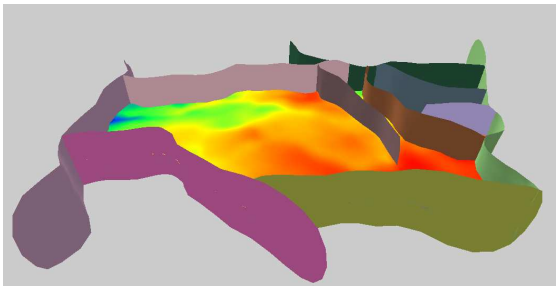
Por esse motivo, é conveniente mostrar e caracterizar alguns dos modelos que o SiVIEP é capaz de visualizar. Basicamente, existem dois tipos principais:

1. *Modelos de geociências*

   - **Poços**: são perfurações na superfície terrestre utilizadas para produzir petróleo ou gás natural. São representados por vários segmentos de linhas que formam a trajetória final do poço (Figura 1.2(a)).
   - **Superfície geológica**: ou falha geológica, é uma superfície em um volume de rocha onde é possível observar que houve deslocamento relativo dos blocos paralelos à falha. São representados por superfícies formadas pela composição de triângulos (Figura 1.2(b)).



1.2(a): Poços



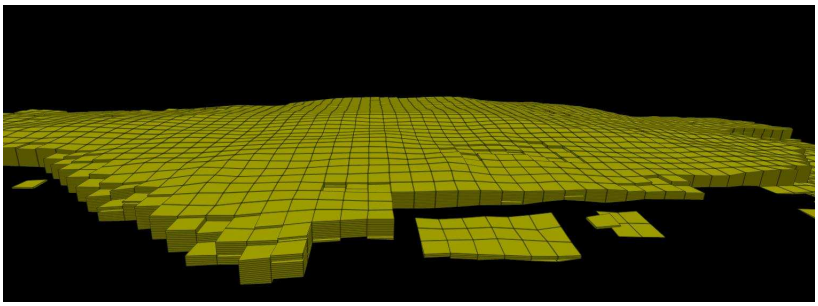
1.2(b): Superfícies Geológicas

Figura 1.2: Modelos de geociências

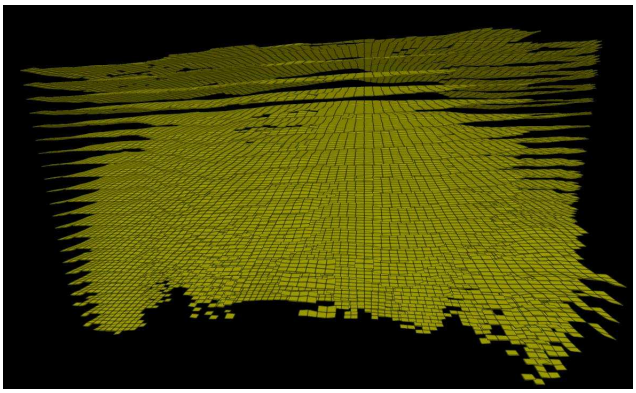
   - **Reservatório**: corpo subterrâneo de rocha com porosidade e permeabilidade suficientes para armazenar e transmitir fluidos. Pode ser representado por uma malha discreta de células em formato de hexaedros, dispostos em grades e geralmente com mais de uma camada [Calomeni and Celes, 2006]. É possível estabelecer uma separação entre as camadas para permitir uma melhor inspeção (Figura 1.3).

2. *Modelos de engenharia*

   - **Plataforma**: é representada por modelos CAD e é um dos objetos mais complexos (Figura 1.4(a)). É possível inspecionar esses modelos tanto externamente quanto internamente.



1.3(a): Sem separação de camadas



1.3(b): Com separação de camadas

Figura 1.3: Modelo de reservatório

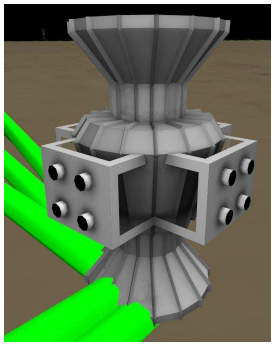
- **Árvore-de-natal**: sistema posicionado no fundo do mar, composto por válvulas conectadas ao poço e à unidade de produção na superfície (Figura 1.4(b)). Estas válvulas permitem o fluxo de produção de petróleo e gás, do poço para a superfície, assim como a injeção de líquido e gás da superfície para o poço.
- **Manifold**: estrutura metálica apoiada no fundo do mar, que acomoda válvulas e acessórios que permitem que este seja conectado às árvore-de-natal e a outros sistemas de produção, indo de tubulações a risers (Figura 1.4(c)).
- **Riser**: conjunto de tubos flexíveis que conectam a unidade de produção e a árvore-de-natal e/ou manifold de produção, permitindo a injeção de gás ou a retirada de óleo e gás.

As definições acima foram retiradas de [TNPetroleo, 2010] e [Schlumberger, 2010].

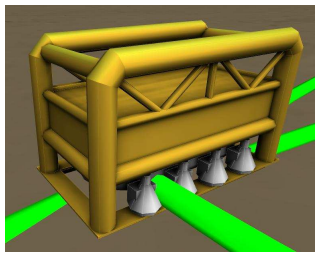
Para permitir que os usuários naveguem por esses modelos, o SiVIEP conta com duas ferramentas principais de navegação: *voar* e *examinar*.

A ferramenta *voar*, como o próprio nome diz, permite que a câmera 'voe' pela cena, em uma determinada velocidade, e o usuário pode dessa forma explorar livremente o ambiente. As setas direcionais do teclado transladam a câmera, enquanto o mouse é usado para orientar a direção do movimento.

1.4(a): Plataforma



1.4(b): Árvore de natal



1.4(c): Manifold

Figura 1.4: Modelos de Engenharia

A ferramenta *examinar* tem como objetivo possibilitar que o usuário examine um determinado objeto. Movimentos de arraste usando o botão esquerdo do mouse rotacionam a câmera em torno de um centro de rotação que pode ser fixado pelo usuário. A operação de *pan*, que permite que o usuário desloque o ambiente lateralmente como se esse fosse uma imagem 2D, é realizada através do 'drag' com o botão central do mouse. O 'drag' vertical/horizontal com botão direito do mouse tem efeito *zoom-in*/*zoom-out* respectivamente. O botão de scroll do mouse, quando presente, também é usado para operações de *zoom*.

1.2 Problemas de Navegação

Nas seções a seguir, são discutidos alguns problemas referentes a navegação em ambientes virtuais. Esses foram identificados a partir da observação do comportamento de diferentes usuários do SiVIEP.

1.2.1 Velocidade de Navegação

Níveis diferentes de escala exigem que a câmera tenha velocidades de navegação diferentes, compatíveis com a escala em que ela se encontra.

Para entender porque isso é necessário, suponha que um usuário esteja explorando o interior de uma plataforma usando a ferramenta *voar*. Nesse caso pode ser conveniente que a velocidade de navegação da câmera seja compatível com a de um ser humano andando, por exemplo. Mas se em um dado momento for necessário navegar até outra plataforma, que está distante 5 km da primeira, a velocidade deve ser reajustada. Caso contrário o tempo necessário para chegar ao destino seria muito grande. Conclui-se então que é necessária alguma maneira de se ajustar a velocidade de navegação.

Como tentativa de encontrar uma solução para esse problema, pode-se tentar observar como essa situação poderia ser resolvida no mundo real. Provavelmente, para ir de uma plataforma a outra se usaria algum tipo de veículo, por exemplo um barco motorizado. Com este seria possível acelerar e desacelerar, permitindo que a velocidade fosse ajustada quando fosse necessário.

Nesse exemplo, o controle está nas mãos do usuário. Logo, uma solução imediata para o problema no mundo virtual seria fornecer também alguma maneira que o permitisse controlar a velocidade de navegação. O scroll do mouse pode ser usado para esse fim: dependendo do sentido de rolamento, a velocidade da ferramenta *voar* é aumentada ou diminuída.

Apesar de funcionar bem na maioria da vezes quando em situações reais, deixar o controle a cargo do usuário se revelou ineficaz no mundo virtual. Ao observar algumas pessoas enquanto usavam o SiVIEP, notou-se que um comportamento acontecia com certa frequência: ao perceberem que estavam em uma escala maior, elas aumentavam exageradamente a velocidade. Uma vez que objeto de interesse ainda estava longe, mesmo que a velocidade fosse dobrada, o usuário não tinha imediatamente a percepção de estar se aproximando do destino. Para compensar, os usuários então aumentavam ainda mais a velocidade, até sentirem o efeito esperado. Quando se aproximavam do objeto, entretanto, a velocidade estava alta demais e aquele acabava sendo ultrapassado. Isso acontecia mesmo com as pessoas que tentavam diminuir a velocidade usando o scroll do mouse, pois não conseguiam realizar a operação a tempo. Como resultado, o que se via era um processo iterativo, onde os usuários iam e voltavam até atingirem o destino. Esse problema também é descrito em [Mackinlay et al., 1990], onde o autor relata a dificuldade das pessoas em controlar altas velocidades.

Conclui-se então que é necessário uma forma de se ajustar automaticamente a velocidade de navegação, onde a necessidade de intervenção por parte do usuário seja a mínima possível.

1.2.2 Ajuste Correto dos Planos de Corte

Os planos de corte *near* e *far*, quando mal configurados também podem se tornar uma fonte de desorientação para o usuário. Esses planos, juntamente com os outros quatro planos que formam o *frustum de visão* (Figura 1.5), definem a região do espaço que pode ser vista. Qualquer objeto localizado fora desses limites não será renderizado pela placa gráfica. Dessa forma, a atribuição de valores incorretos pode fazer com que objetos não sejam exibidos na tela ou, ainda, que sejam parcialmente cortados, como mostra a Figura 1.6.

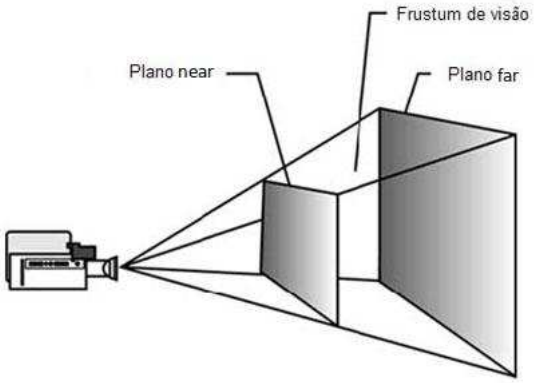
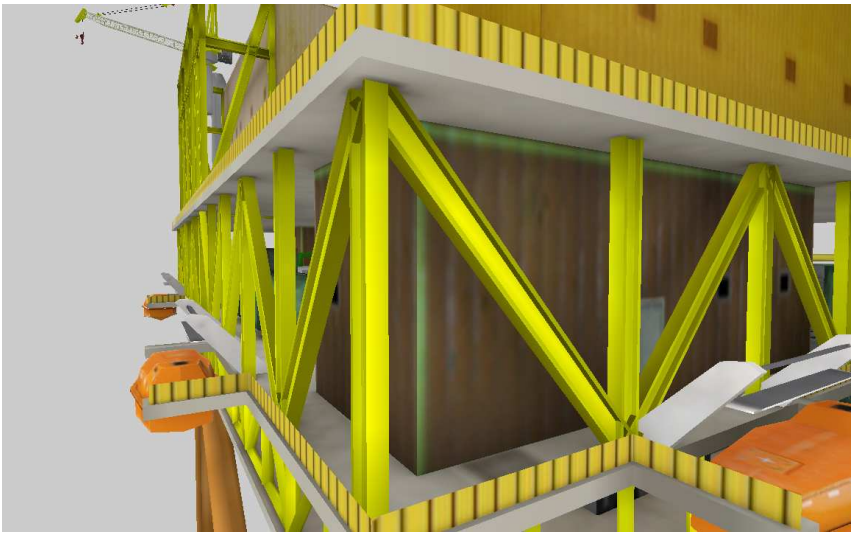


Figura 1.5: Frustum de visão



1.6(a): Plano *near* definido corretamente.

1.6(b): Plano *near* incorreto cortando do modelo.

Figura 1.6: Ajuste dos planos de corte.

Os valores *near* e *far* também influenciam fortemente no uso eficiente do *buffer de profundidade*. Devido ao modo como o cálculo de perspectiva é feito,

o *buffer de profundidade* possui uma precisão maior para objetos localizados perto do plano *near* do que para os localizados perto do plano *far*. De acordo com [OpenGL Architecture Review Board, 1997], sendo $r = \frac{near}{far}$, então $log_2\ r$ bits de precisão do *buffer de profundidade* são perdidos. Dessa forma, quanto maior for r, menor será a capacidade de se distinguir duas superfícies próximas entre si. O resultado disso é o aparecimento de artefatos (pontos que ficam piscando na tela) em objetos que estão mais distantes da câmera.

As soluções para esses dois problemas são conflitantes: por um lado, deseja-se aumentar o valor de r a fim de garantir que nenhum objeto seja cortado indevidamente. Por outro, o valor de r deve ser pequeno o suficiente para que não ocorram efeitos indesejados na visualização dos modelos. Esse dilema ocorre principalmente quando se deseja exibir cenas muito grandes, o que é justamente o caso do SiVIEP.

1.2.3 Tratamento de Colisão

Quando não há tratamento de colisão, torna-se possível atravessar objetos presentes na cena. Apesar de conveniente em alguns casos, essa situação geralmente configura um erro do ponto de vista do usuário.

Identificou-se que algumas pessoas, ao usarem o SiVIEP, sentiam desconforto ou ficavam desorientadas quando por algum motivo atravessavam paredes ou equipamentos presentes em uma plataforma. Outra situação em que esse efeito ocorria era ao navegarem entre duas camadas de um reservatório: os usuários não conseguiam se manter no espaço entre camadas e acabavam colidindo com essas.

Por fim, no futuro pretende-se que o SiVIEP tenha suporte a imersão com o uso de *caves*. Neste caso, colisões com o ambiente podem resultar em situações bem desconfortáveis [Calomeni and Celes, 2006], e o tratamento daquelas torna-se imprescindível. Ainda, atualmente o SiVIEP conta com suporte ao efeito de estereoscopia. Quando este está ativado, o ato de atravessar um objeto provoca uma sensação física nos olhos do usuário que pode ser bem desconfortável.

1.2.4 Senso de Orientação e Localização

Saber se orientar em um determinado ambiente é fundamental para o processo de navegação. Segundo conclui [Darken and Sibert, 1996], sem indicações de direção, a desorientação inibe uma navegação eficiente e esse

problema é pior em mundos virtuais maiores ou com pouca organização estrutural.

A desorientação também pode ser causada pelo uso incorreto das ferramentas de navegação. Um problema identificado no SiVIEP revelou que alguns usuários têm dificuldade em lidar com a ferramenta de centro de rotação. Essa permite escolher um novo centro de rotação que, por sua vez, é usado pela ferramenta *examinar*.

O problema é que a maioria das pessoas esqueciam de escolher um novo centro de rotação antes de iniciar o uso da ferramenta *examinar*. Como exemplo, considere inicialmente que o centro de rotação está localizado em um objeto $p1$. O usuário decide então navegar até outro objeto $p2$ usando a ferramenta *voar*. Ao chegar lá, tenta examinar $p2$ rotacionando a câmera em torno desse, mas é surpreendido por um movimento que considera estranho. Isso ocorre pois a rotação está sendo feita em torno de $p1$, não $p2$, uma vez que o centro de rotação não foi devidamente reconfigurado.

Concluindo, deve-se fornecer ao usuário alguma indicação que o permita se localizar em relação ao ambiente. É necessário também que haja uma forma de se garantir o estado correto do centro de rotação.

1.3 Objetivo

O objetivo deste trabalho é propor e implementar soluções para os problemas que foram abordados anteriormente. Tais soluções devem atender da melhor forma a dois requisitos principais:

1. Serem o mais automatizadas possíveis, de modo que exijam o mínimo de intervenção por parte do usuário.

2. Serem independentes dos tipos de modelos a serem visualizados. Ou seja, as soluções não devem se basear em características únicas de cada modelo. Isso permite que novos tipos de objetos sejam visualizados no futuro sem que haja necessidade de criar novas soluções.

Nesse sentido, foram estudados vários trabalhos de outros pesquisadores visando encontrar possíveis soluções. Dentre esses, o trabalho de [McCrae et al., 2009] se revelou como a melhor das alternativas.

[McCrae et al., 2009] abordam as questões do controle de velocidade de navegação em AMEs, do ajuste dos planos de corte e da detecção de colisão entre a câmera e o ambiente virtual. Para isso, eles criaram uma estrutura de dados chamada de *cubo de distâncias* (*Cube Map*, no original em inglês), que serve como base de todas as técnicas apresentadas por eles.

Neste trabalho foram implementadas algumas técnicas descritas por [McCrae et al., 2009] e são propostas também outras soluções baseadas na estrutura *cubo de distâncias*.

1.4 Estrutura da Dissertação

Este trabalho está organizado em 5 capítulos. No próximo capítulo são discutidos outros trabalhos relacionados ao tema desta dissertação. No Capítulo 3 são apresentadas as soluções implementadas para resolver os problemas descritos nas seções anteriores. O Capítulo 4 apresenta resultados de desempenho e testes com usuários. Por fim, o Capítulo 5 conclui este trabalho e apresenta algumas propostas de trabalhos futuros.

2
Trabalhos Relacionados

Neste capítulo são discutidos alguns trabalhos relacionados ao tema dessa pesquisa e que serviram como base para as soluções propostas neste trabalho.

2.1
Navegação em Ambientes Virtuais

A navegação em ambientes virtuais tem sido foco de estudo de vários pesquisadores. Boa parte desses trabalhos tem como objetivo principal propor e analisar técnicas que facilitem a tarefa de explorar esses ambientes.

Por exemplo, [Ware and Osborne, 1990] analisam três diferentes ferramentas de navegação: *eyeball in hand*, *scene in hand* e *flying vehicle control*. Essas técnicas, apesar de inicialmente terem sido implementadas fazendo uso de dispositivos de controle para ambientes imersivos, podem ser alteradas para usar dispositivos mais comuns como mouse e teclado.

A primeira ferramenta, *eyeball in hand*, consiste em mapear os movimentos do dispositivo de entrada de forma a alterar a orientação da câmera virtual. A técnica *scene in hand* faz o oposto: movimentos realizados com o controle tem o efeito de alterar a orientação da cena, não da câmera. Esse comportamento é similar ao da ferramenta *examinar*, descrita na Seção 1.1. A última, *flying vehicle control*, corresponde à ferramenta *voar*, também descrita na Seção 1.1.

[Ware and Osborne, 1990] concluem que nenhuma dessas três técnicas eram capazes de suprir, sozinhas, todas as necessidades dos usuários. Por exemplo, eles observaram que ao explorar ambientes internos, a ferramenta *flying vehicle control* é mais adequada, enquanto que *scene in hand* é mais indicada para ambientes externos.

[Fitzmaurice et al., 2008] chegam à mesma conclusão ao analisar o comportamento de vários usuários. Ainda, observam também que boa parte das pessoas não têm compreensão de que geralmente são necessárias mais de um tipo de ferramenta para garantir uma navegação eficiente. Elas acabam sempre tentando usar uma mesma ferramenta para todos os fins. Como solução, os autores propõem que os menus de seleção de ferramentas sejam organizados

de forma a dar uma ênfase maior para aquelas que julgam ser mais importantes (Figura 2.1).

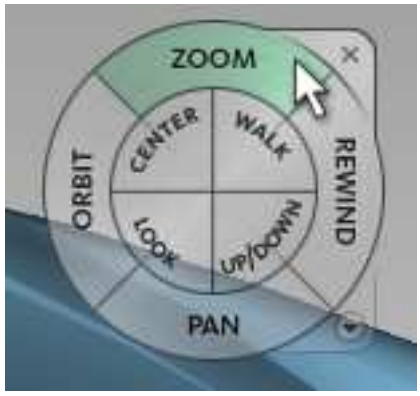


Figura 2.1: Organização do menu de ferramentas de interação [Fitzmaurice et al., 2008].

[Mackinlay et al., 1990] introduzem a técnica de *ponto de interesse (POI)*. Essa consiste em escolher um ponto da cena, onde o usuário deseja chegar. A partir daí a câmera realiza uma animação, seguindo de sua posição atual até o *POI*. A cada quadro, a posição da câmera é atualizada de acordo com a seguinte equação:

$$camPos = camPos - k(camPos - poi) \tag{2-1}$$

Na Equação 2-1 $camPos$ é a posição atual da câmera, poi é o ponto de interesse e k é uma constante que influencia na velocidade de aproximação. O efeito dessa equação é que a câmera parece se aproximar do poi a uma taxa proporcional à constante k. Segundo o autor, análises informais revelaram que esse tipo de movimento é mais natural ao usuário.

Em [Tan et al., 2001] é proposta uma solução híbrida, em que as ferramentas *voar* e *examinar* são compostas em uma única técnica de interação. O controle é feito com base no movimento de *dragging* do mouse: se o usuário inicia um *dragging* clicando em um ponto em que não há objetos, o comportamento resultante é o de *voar*. Se, ao contrário, o usuário clica em algum objeto, esse é posto no centro de tela e, a partir desse ponto, é possível examiná-lo.

[Drucker and Zeltzer, 1994] desenvolveram um sistema de controle automático de câmera baseado em restrições. Nele, a câmera sofre influência de várias restrições, definidas previamente com base no tipo de tarefa que se deseja realizar. Por exemplo, pode-se especificar que o vetor *up* da câmera sempre fique alinhado com o vetor *up* do mundo, ou que a câmera sempre aponte para um ponto específico do mundo. Um dos módulos desenvolvidos pelos autores permite determinar o caminho que a câmera deve seguir para ir de um ponto

a outro. Para isso, um grafo de conectividade da cena é construido em tempo de pré-processamento e o caminho entre os dois pontos é calculado usando o algoritmo A*.

O sistema proposto por [Drucker and Zeltzer, 1994], entretanto, apresenta instabilidade no posicionamento de câmera. Como solução, [Hermann and Celes, 2005] propõem que a câmera seja modelada por um sistema físico de partículas sujeitas a restrições de barras rigidas [Jakobsen, 2001]. Isso permite movimentos mais suaves.

2.2 Ajuste dos Parâmetros de Navegação em Ambientes Multiescala

Ambientes multiescala foram inicialmente introduzidos pelos trabalhos de [Perlin and Fox, 1993] e [Bederson et al., 1994; Bederson and Hollan, 1994], com a criação das interfaces *Pad* e *Pad++*. Esses sistemas permitiam visualizar informações em um plano de projeção 2D usando uma *lupa virtual*: o local da tela que é apontado por essa lupa é mostrado com mais detalhes. É possível também realizar operações de *zoom* nesses sistemas. Quando isso acontece, novas informações são exibidas, baseadas no contexto dos objetos de interesse. Por exemplo, no caso de um sistema de arquivos, inicialmente seriam vistas várias pastas. Ao dar *zoom* em uma dessas, seria possível ver os arquivos que estão dento dela e, no fim, o conteúdo de um determinado arquivo. Cada um desses níveis pode ser considerado uma escala diferente de visualização.

[Furnas and Bederson, 1995] desenvolveram o conceito de *diagramas de espaço-escala*, com o objetivo de facilitar a criação e análise de interfaces 2D multiescalas. Esses diagramas têm o formato de uma pirâmide de vários níveis, onde cada um desses representa uma escala referente a um ponto de vista do ambiente.

Apesar de serem basicamente interfaces 2D, os conceitos presentes nesses trabalhos podem ser aplicados também para a criação de ferramentas para navegação em ambientes virtuais 3D. [Kopper et al., 2006], por exemplo, utilizam a mesma ideia de *lupa virtual* de [Perlin and Fox, 1993].

O objetivo de [Kopper et al., 2006] é permitir a exploração das diferentes escalas de um ambiente virtual que representa o corpo humano. Assim, o usuário deve ser capaz de visualizar o corpo, membros, orgãos, células e até o DNA. Para isso, os autores desenvolveram o conceito de *nível de escala* (*LoS*, do inglês *level of scale*).

Um *nível de escala* pode ser comparado com a ideia de *nível de detalhe* (*LoD - level of detail*). Em algoritmos de *LoD*, objetos próximos à câmera são mostrados com maior nível de detalhe, enquanto objetos distantes têm sua

geometria simplificada, com o objetivo de melhorar a performance. Da mesma forma, um determinado *LoS* também mostra mais informações quando próximo da câmera. Entretanto, em um *LoS* não é somente o visual dos objetos que sofre alteração, mas também a forma como o usuário deve interagir e navegar nessa parte do ambiente.

[Kopper et al., 2006] definem uma hierarquia de *LoSs*. Por exemplo, o corpo humano inteiro é a raiz da hierarquia e configura o primeiro *LoS*. Os membros do corpo vêm a seguir, cada um correspondendo a um novo *LoS*. Esse processo segue até o menor nível de escala possível, o DNA das células.

Para navegar entre esses diferentes níveis de escala, os autores usam uma *lupa virtual* para enxergar a existência de *LoSs* que estejam em hierarquias mais baixas (Figura 2.2). A troca de hierarquia é feita quando o usuário executa um comando e uma animação é então criada para indicar essa mudança. Uma vez que a câmera está no *LoS* de interesse, é possível usar a ferramenta *voar* para explorar essa parte do ambiente. A velocidade de navegação e os planos de corte são ajustados com a aplicação de um fator de escala dado pela seguinte equação:

$$fatorEscala = \sqrt[3]{\frac{Volume_{novoLoS}}{Volume_{antigoLoS}}} \tag{2-2}$$

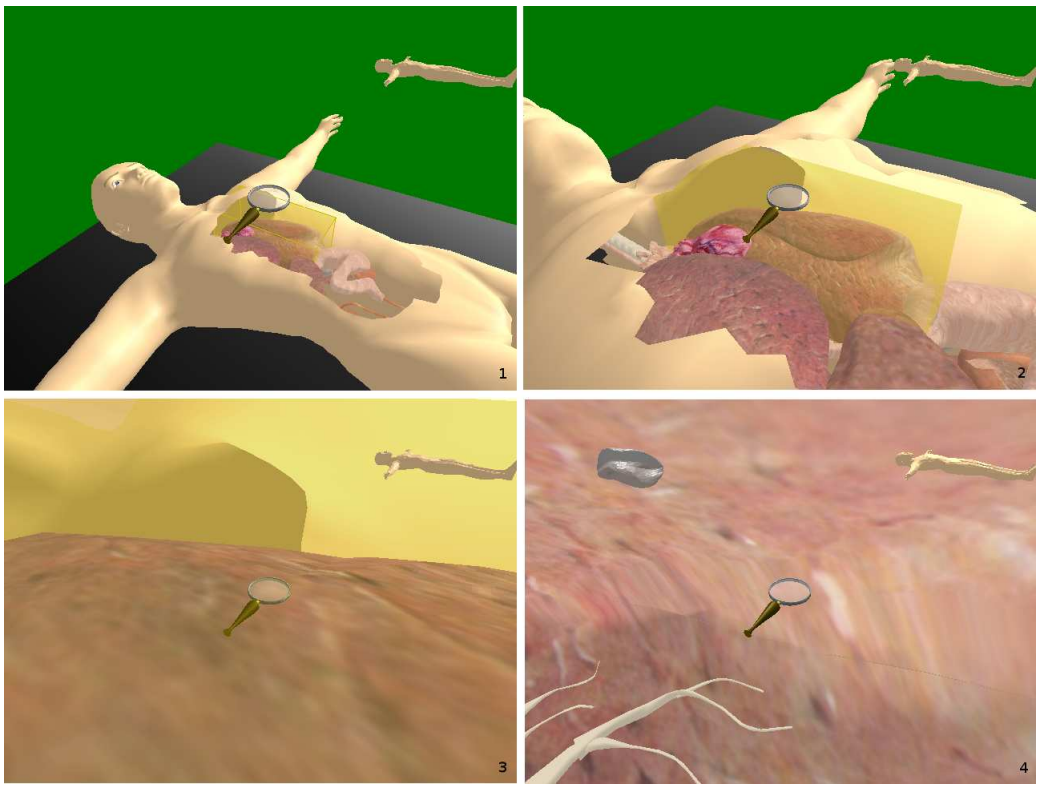


Figura 2.2: Visualização dos diferentes *níves de escala* do corpo humano com o auxílio da *lupa virtual* [Kopper et al., 2006].

Na equação 2-2, $Volume_{novoLoS}$ e $Volume_{antigoLoS}$ são, respectivamente, os volumes das caixas envolventes do *LoS* que se deseja ir e do *LoS* anterior.

A técnica de [Kopper et al., 2006] fornece uma maneira simples e funcional de exploração de diferentes níveis de escala. Entretanto, ela não permite a navegação livre entre dois diferentes *LoSs*. O usuário deve obrigatoriamente executar um comando para realizar essa mudança. Por último, o ajuste dos parâmetros de navegação é feito somente no momento da mudança de escala. Dessa forma, pode-se dizer que o método de [Kopper et al., 2006] é discreto no sentido de que os diferentes níveis de escala são bem definidos, tanto na sua forma quanto na sua localização na hierarquia de *LoSs*.

Os trabalhos de [Ware and Fleet, 1997] e [McCrae et al., 2009], ao contrário, realizam ajustes contínuos nos parâmetros de navegação e, dessa forma, não necessitam que o ambientes virtuais tenham uma hierarquia de níveis de escala bem definida.

[Ware and Fleet, 1997] propõem ajustar a velocidade de navegação da ferramenta *voar* usando a informação de profundidade presente no *Z buffer*. Este buffer possui a mesma resolução da tela e guarda as distâncias de cada ponto dessa até a câmera. Por motivos de desempenho, o autor verifica apenas 15 linhas horizontais e igualmente espaçadas. Foram testados quatro diferentes formas de se ajustar a velocidade de navegação: usando o ponto mais próximo (menor valor de profundidade encontrado), o ponto mais distante (maior valor de profundidade), a média dos valores amostrados e por último, a média de valores amostrados considerando uma região de interesse. A conclusão a que os autores chegaram é que o ponto mais próximo é o mais adequado para realizar esse tipo de ajuste.

Essa também foi a conclusão de [McCrae et al., 2009], que propõem uma ferramenta de navegação baseada em *POI* [Mackinlay et al., 1990]. Nesse trabalho, a distância da câmera ao ponto mais próximo da cena é usada para controlar a velocidade de navegação de acordo com a seguinte equação:

$$camPos = camPos + (poi - camPos)t_{\Delta}\frac{minDist}{2} \tag{2-3}$$

Na equação 2-3, $camPos$ é a posição da câmera, poi é o ponto de destino, t_{Δ} é o intervalo de tempo entre dois quadros de animação e $minDist$ é a distância da câmera ao ponto mais próximo da cena.

Para determinar $minDist$ é construído um *cubo de distâncias*, cujas faces correspondem aos 6 planos de projeção resultantes da orientação da câmera nas 6 direções canônicas. Cada face guarda informação de profundidade, similar ao *Z buffer*. O cálculo dessa estrutura é feito da seguinte forma: toda vez que a câmera se move, as faces do cubo são renderizadas em 6 imagens diferentes,

usando a nova posição da câmera. O ângulo de abertura da câmera é de 90^0, de forma a cobrir todo o espaço do ambiente virtual, sendo limitado apenas pelos planos de corte. Através de um programa que roda na placa gráfica, a distância de um ponto visível até câmera é escrito no canal alpha do pixel referente ao fragmento gerado pelo ponto. Ao final, *minDist* é determinado fazendo uma varredura pelo menor valor presente nas imagens. Por uma questão de desempenho, a renderização do cubo de distâncias é feita usando uma resolução menor do que a da aplicação.

Além de ajustarem a velocidade de navegação, [McCrae et al., 2009] também utilizam *minDist* para determinar os planos de corte. No Capítulo 3, o modo como isso é feito é explicado, assim como a construção do cubo de distâncias é descrito com mais detalhes.

2.3 Detecção e Tratamento de Colisão

[Baciu et al., 1998] classificam as técnicas de detecção de colisão entre objetos em ambientes virtuais em duas linhas principais de pesquisa, com base no tipo de teste de interseção que é feito: *testes de interseção feitos no espaço do objeto* e *testes de interseção feitos no espaço da imagem.*

As técnicas baseadas em testes feitos no espaço do objeto usam diretamente a informação de geometria dos modelos para verificar a existência de colisão entre esses. Geralmente é necessário pré-computar estruturas de dados especiais, como árvores BSP [Thibault and Naylor, 1987], que permitam detectar regiões de possível colisão. O desempenho desses métodos é relacionado com a quantidade de geometria existente: quanto maior o número de modelos, maior será o custo envolvido no cálculo de colisão.

As técnicas baseadas no espaço da imagem consistem em projetar a geometria no plano de imagem usando o hardware gráfico e, então, são realizados testes de interferência geralmente através da análise do *Z buffer* [Baciu and Wong, 1997; Baciu et al., 1998]. Uma vez que os testes são feitos no espaço da imagem, a quantidade de geometria tem uma influência bem menor sobre o custo para computar a colisão. Outro ponto positivo é que esses métodos fazem uso maior do processamento das placas gráficas, reduzindo o número de cálculos realizados na CPU.

Os trabalhos mencionados anteriormente se preocupam com a detecção de colisão entre todos os objetos da cena. Entretanto, o problema que se quer tratar aqui é um pouco mais simples: deseja-se detectar se há colisão somente entre a câmera e os modelos da cena, com o objetivo de melhorar a experiência de navegação para o usuário.

Diversas técnicas foram propostas nesse sentido. Por exemplo, para permitir uma navegação eficiente em modelos de reservatórios de petróleo, [Calomeni and Celes, 2006] controem um grafo de conectividade em tempo de pré-processamento que os permite traçar um caminho livre de colisão para a câmera. Nesse grafo, os nós representam locais em que a câmera pode ir sem que haja colisão. As arestas indicam a existência de um caminho, também livre de colisão, entre os nós conectados por essas.

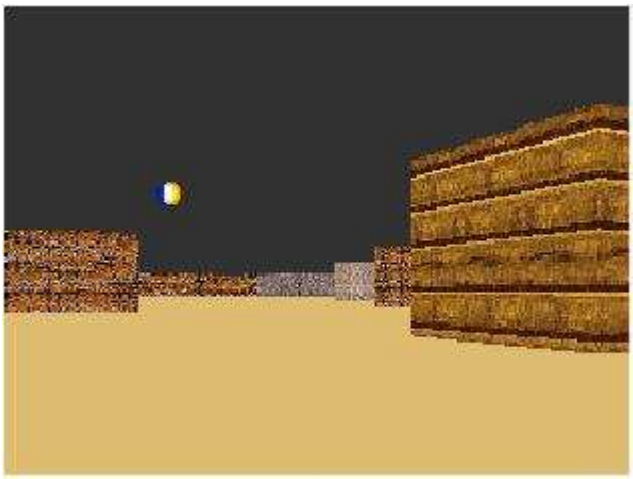
Com base nessa estrutura, [Calomeni and Celes, 2006] desenvolveram dois modos de navegação:

- *Modo de navegação automático*: nesse modo, o usuário seleciona um nó de destino e a câmera realiza uma animação livre de colisão entre o ponto atual e o de destino. A escolha do caminho é feita usando uma versão modificada do algoritmo A*.
- *Modo de navegação assistida*: nesse modo o usuário pode navegar livremente pelo ambiente usando a ferramenta *voar*. Entretanto, o grafo de conectividade é consultado diversas vezes durante a navegação para verificar a existência de colisão. Com base na direção de movimento da câmera, um nó de destino é inferido. O nó de origem é determinado a partir da posição atual da câmera. O algoritmo A* é então executado para achar um caminho entre esses dois nós, que por sua vez será usado para corrigir a trajetória da câmera de forma a evitar colisões com o ambiente.

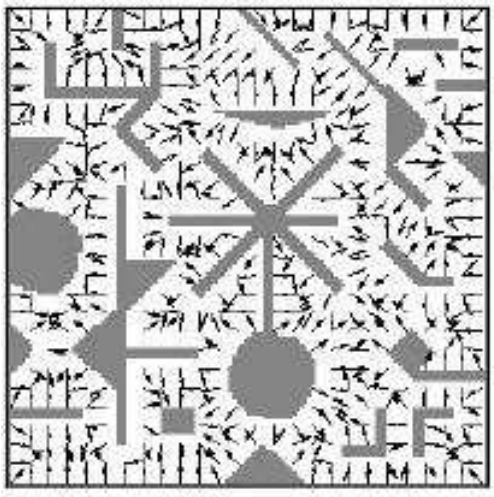
Esse tipo de navegação assistida também foi usada por [Xiao and Hubbold, 1998]. Em sua pesquisa, ele introduz o uso de *campos de forças artificiais* no processo de navegação. Nessa abordagem, os objetos são envoltos por forças de repulsão. Essas são calculadas durante a navegação, através de raios lançados contra a geometria, a fim de encontrar os objetos mais próximos. Ao se aproximar de um modelo, a câmera é repelida por uma dessas forças. Quanto maior a proximidade da câmera, maior é a força de repulsão. Como efeito, a câmera sofre um desvio no sentido de ser jogada para fora da região de colisão.

[Li and Chou, 2001] estendem a ideia de [Xiao and Hubbold, 1998] e transferem a computação das forças para uma etapa de pré-processamento. Para isso, eles assumem que o ambiente virtual pode ser representado em um formato 2D (Figura 2.3). Como resultado, é possível obter um mapa de campo de forças com maior resolução, se comparado a técnica de [Xiao and Hubbold, 1998]. Durante a navegação, a amplitude desses vetores de forças são alterados

com base na posição da câmera, de forma o obter o mesmo efeito de [Xiao and Hubbold, 1998].



2.3(a):



2.3(b):

Figura 2.3: Mapa de campo de forças: em (a), o ambiente virtual de um labirinto. Em (b), a representação 2D desse labirinto, juntamente com os vetores do campo de força. Figuras retiradas de [Li and Chou, 2001].

Ainda na linha de pesquisa baseada em campos de forças artificial, [McCrae et al., 2009] propõem o uso da placa gráfica para auxiliar na construção do mapa de forças. Para isso, ele usa os canais RGB do cubo de distâncias para guardar o vetor que aponta na direção da câmera, vindo do ponto referente ao fragmento gerado no processo de renderização. Essa abordagem permite obter uma mapa de forças com resolução maior, além de eliminar a etapa de pré-processamento proposta por [Li and Chou, 2001]. A desvantagem é que a construção do cubo de distâncias aumenta o processamento necessário durante a navegação, uma vez que são realizadas mais 6 passadas de renderização.

O mapa de forças presente no cubo de distâncias é utilizado para calcular uma força de repulsão resultante, que por sua vez é usada em conjunto com a ferramenta estilo *POI* descrita na Seção 2.2. Essa força faz com que a câmera seja repelida pelos objetos encontrados no meio do caminho.

2.4 Auxílio para Orientação e Localização

Além dos problemas discutidos anteriormente, outro motivo frequente de navegação ineficiente tem relação com a dificuldade que alguns usuários têm em se orientar e se localizar em ambientes virtuais.

Dentre as diversas causas, [Mine, 1995] cita a inexistência de informações suficientes e o projeto ruim das ferramentas de navegação. Essa última causa também é apontada por [Fitzmaurice et al., 2008], que identifica a ocorrência do problema descrito na Seção 1.2.4, relativo ao posicionamento do centro de rotação da ferramenta *examinar*.

Os autores observaram que os usuários tinham dificuldade em lidar com o centro de rotação pois muitos ignoravam, ou mesmo não percebiam, sua existência. Como solução, eles decidiram mostrar uma pequena esfera para indicar a localização desse. Além disso, eles estabeleceram que o centro de rotação sempre deve estar visível na tela.

A técnica anterior ilustra um tipo de solução muito comum em ambientes virtuais: se naturalmente não há informação visual sufuciente, então ela deve ser criada no sentido de guiar o usuário. Por exemplo, [Darken and Sibert, 1993] permitem que os usuários deixem marcas pelo caminho enquanto estão navegando, a fim de que possam refazer o caminho de volta e não fiquem perdidos.

[Jul and Furnas, 1998] introduzem o conceito de *resíduo*, que possibilita que o usuário perceba a existência de um objeto, mesmo que este não esteja visível por estar muito pequeno. A ideia é colocar marcas onde os objetos deveriam estar aparecendo. Essa também foi a abordagem aplicada por [Pierce and Pausch, 2004] em seu trabalho, no qual é descrito um sistema de navegação baseado em marcas com formato dependentes dos objetos que elas representam (Figura 2.4). Por exemplo, um farol é representado por um ícone na forma de um farol em miniatura. Esses ícones só se tornam visíveis ao usuário quando este está afastado a uma certa distância. Para evitar que essas marcas se sobreponham, criando assim um emaranhado confuso, os autores também propõem técnicas para decidir quando e quais ícones devem ser visíveis em um dado instante.

[Stoakley et al., 1995] criaram uma versão em miniatura do mundo

2.4(a):

2.4(b):

Figura 2.4: Visualização de uma fazenda antes (a) e depois (b) da adição de marcas para os objetos da cena [Pierce and Pausch, 2004].

virtual (Figura 2.5). Através dela é possível ao usuário saber sua localização e orientação, e este pode planejar melhor qual caminho seguir, uma vez que é possível visualizar todo o ambiente através da miniatura. O modelo também pode ser rotacionado, a fim de se conseguir ter vários pontos de vista do ambiente. As ações realizadas no modelo em miniatura não afetam o de escala real. Dessa forma, ações incorretas executadas pelo usuário podem ser facilmente revertidas.

Por último, a indústria de jogos eletrônicos também fornece vários exemplos de técnicas que visam facilitar a orientação e localização em mundos virtuais. Muitos jogos de estratégia usam mapas 2D e marcam neste a posição do jogador. Jogos de corrida também fazem uso dessa técnica e, além da posição do carro, indicam também a direção que deve ser seguida através de setas. Essas também são usadas para indicar a posição de um determinado objeto da cena, ou mesmo outro jogador no caso de jogos *multiplayers*. Essa ideia

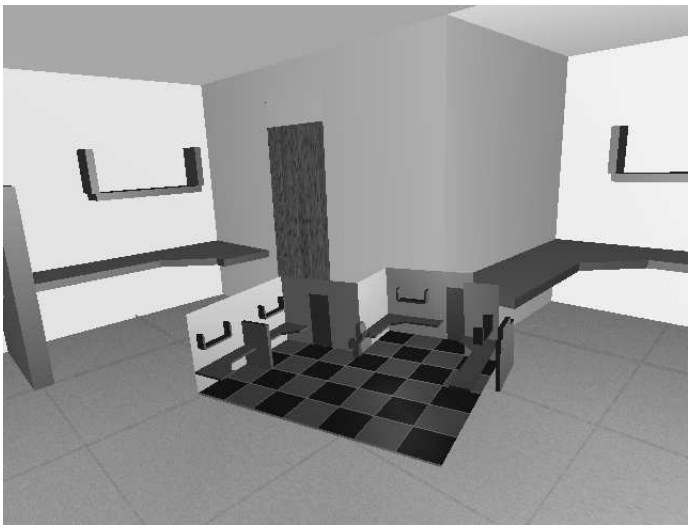


Figura 2.5: Visualização da versão em miniatura (centro da figura) de um cenário virtual [Stoakley et al., 1995].

foi incorporada ao SiVIEP com a criação de uma *seta indicadora*, a qual é explicada com mais detalhes no próximo capítulo.

2.5 Análise Final

Analisando os diferentes trabalhos mencionados anteriormente é possível estabelecer um conjunto de técnicas que melhor se adequam aos requisitos do SiVIEP.

A solução proposta por [McCrae et al., 2009] é interessante pois trata de três dos problemas apresentados aqui: velocidade de navegação, ajuste dos planos de corte e detecção e tratamento de colisão. As técnicas são baseadas em uma estrutura chamada de *cubo de distâncias*, cuja construção independe do formato ou das características dos objetos da cena. Também não é necessário que nenhuma etapa de pré-processamento seja realizada.

As técnicas discutidas em [Kopper et al., 2006] também não requerem pré-processamento, mas necessitam que uma hierarquia de níveis de escala possa ser bem definada. Isso gera um novo problema, no qual a questão é como a hierarquia deve ser estabelecida. Ainda, a navegação exige que o usuário conheça e compreenda as relações existentes entre os diferentes níveis de escala.

O modelo de navegação assistida de [Xiao and Hubbold, 1998], [Li and Chou, 2001] e [Calomeni and Celes, 2006] pode ser implementado no SiVIEP em conjunto com a ferramenta *voar* já existente e com a detecção de colisão. O grafo de conectividade descrito em [Calomeni and Celes, 2006] permite também determinar a trajetória entre dois pontos quaisquer do ambiente. Entretanto, a construção desse grafo exige pré-processamento e é dependente da forma da

geometria que os autores se dispõem a visualizar.

Como conclusão observou-se que, dentre as técnicas estudadas, aquelas propostas por [McCrae et al., 2009] são as que melhor atendem ao SiVIEP. Em especial, o *cubo de distâncias* se revelou útil também na criação de uma solução automática para o problema do centro de rotação, e de uma técnica cujo objetivo é auxiliar o usuário a se localizar e orientar no ambiente virtual. Os detalhes do funcionamento e de como essas técnicas foram implementadas no SiVIEP são tema do próximo capítulo.

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 [McCrae et al., 2009], ilustra o funcionamento do cubo de distâncias. Em seu trabalho, [McCrae et al., 2009] 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

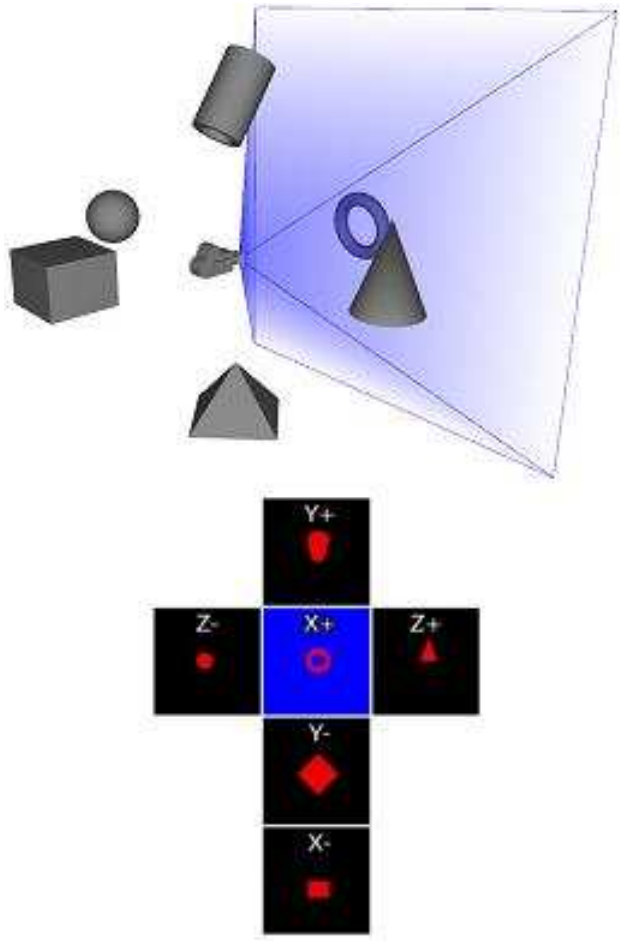


Figura 3.1: Cubo de Distâncias [McCrae et al., 2009].

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
```

```
 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;
```

```
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

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 }
```

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. [McCrae et al., 2009] 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.

[McCrae et al., 2009] 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:

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

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.

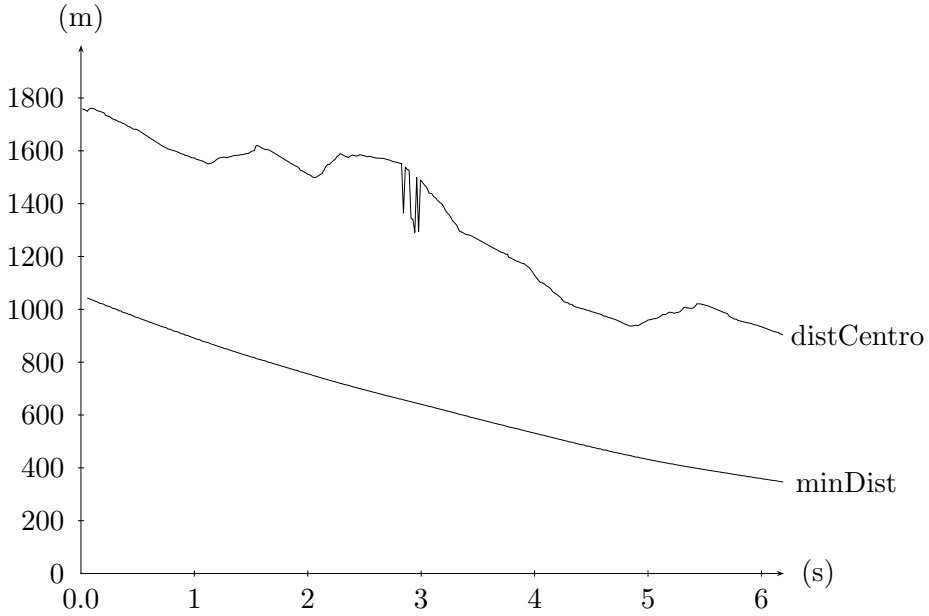
Na técnica proposta por [Mackinlay et al., 1990], a velocidade de navegação é ajustada com base na distância em que a câmera se encontra do ponto de destino escolhido, como pode ser visto na equação 2-1. Com isso, os autores conseguem um ajuste que eles creem se adaptar melhor às necessidades dos usuários.

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 [Mackinlay et al., 1990].

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.



Distância x Tempo

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* 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.

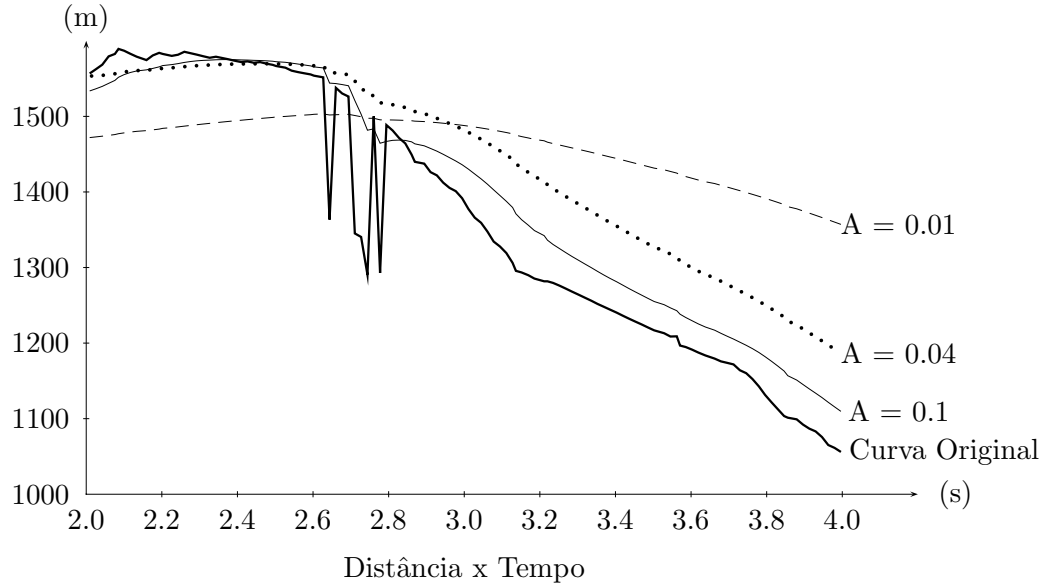


Figura 3.3: Efeito da aplicação da *média exponencial móvel* sobre um curva.

Isso ocorre pois a percepção de proximidade fornecida por *minDist* não 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 indevido de objetos até o aparecimento de artefatos em objetos distantes da câmera.

[McCrae et al., 2009] 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 [McCrae et al., 2009], 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 [McCrae et al., 2009], 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

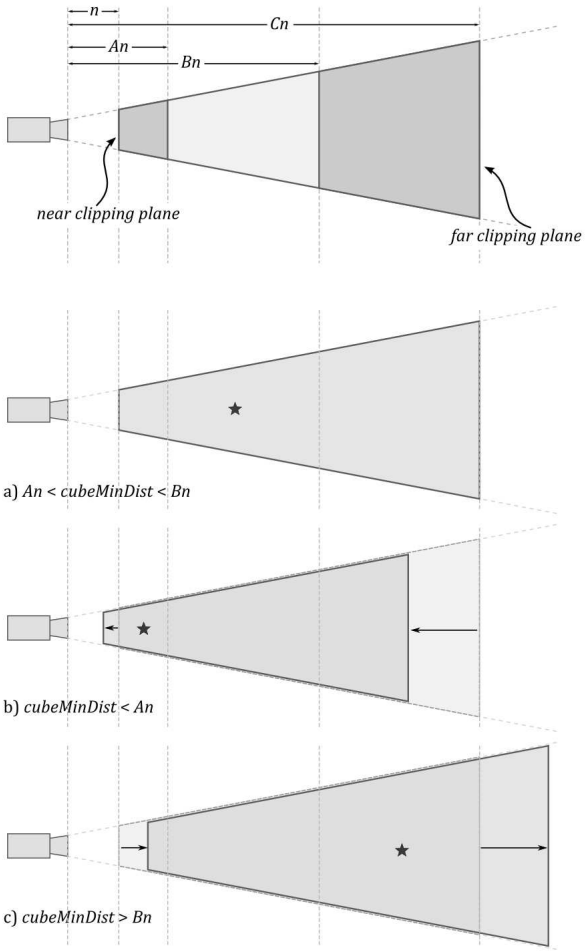


Figura 3.4: Ajuste automático dos planos de corte

distantes. Devido à proximidade da geometria, o valor de *near* é reduzido. 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 $r = \dfrac{near}{far}$ 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

3.5(a): 3.5(b):

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

problema ilustrado na Figura 3.5 não é tão crítico. Ao navegarem pelo ambiente 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 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, [McCrae et al., 2009] 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:

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

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 [Calomeni and Celes, 2006], [Xiao and Hubbold, 1998] e [Li and Chou, 2001]: 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 [McCrae et al., 2009], 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.

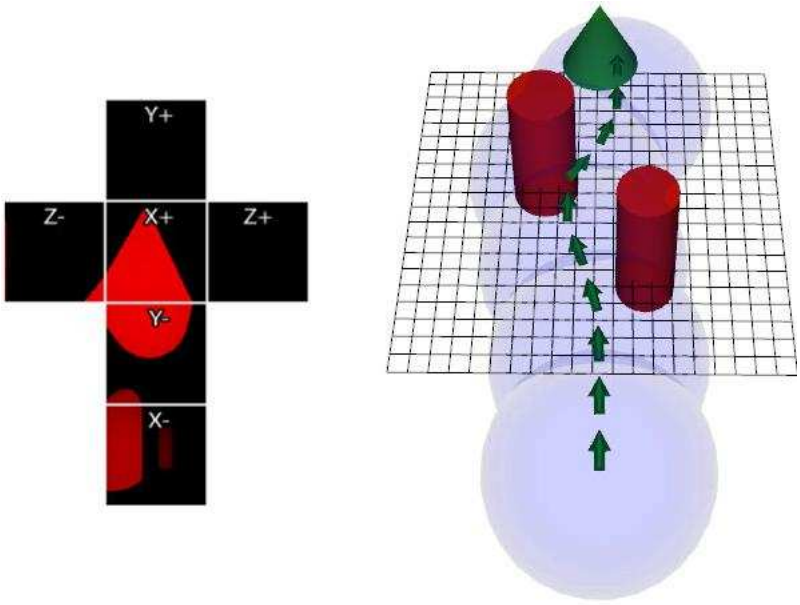


Figura 3.6: Efeito da aplicação de $F_{colisao}$ sobre a câmera [McCrae et al., 2009]

$F_{colisao}$ 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 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. [Fitzmaurice et al., 2008] descrevem algumas situações em que isso ocorre. Por exemplo, a Figura 3.7 ilustra três desses casos.

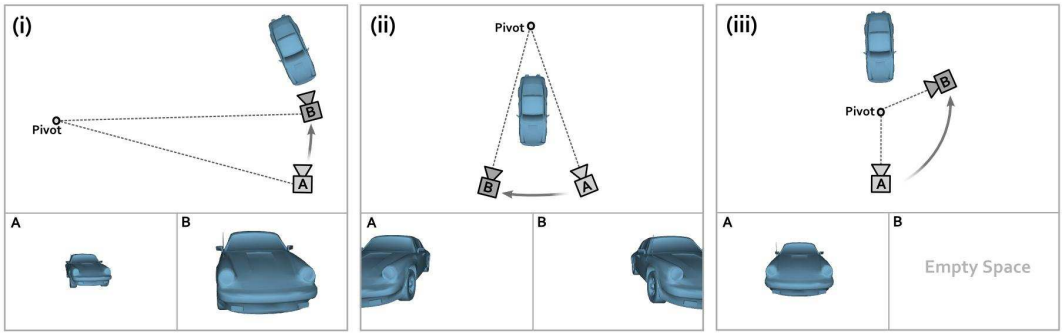


Figura 3.7: Problema relacionados ao centro de rotação [Fitzmaurice et al., 2008].

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 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 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.

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 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.

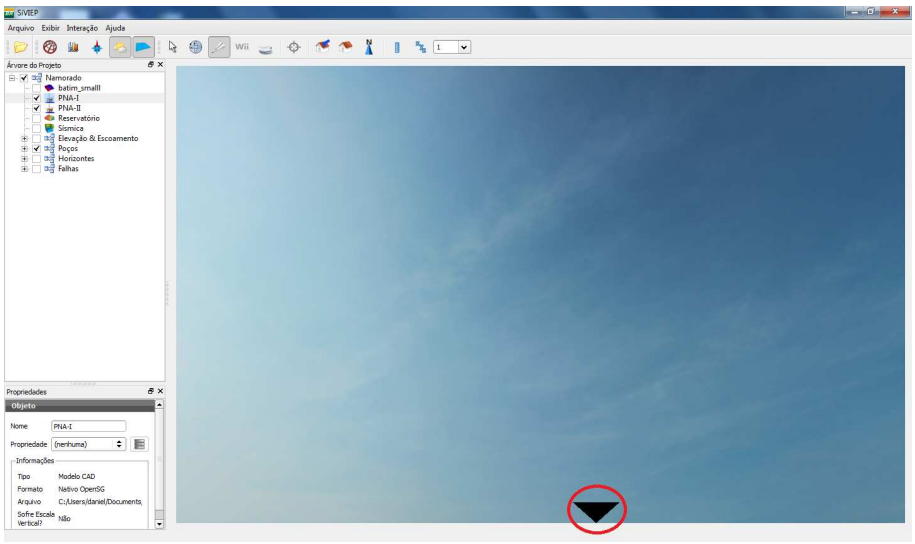


Figura 3.8: Seta indicadora

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 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.

4
Testes e Resultados

Este capítulo apresenta e discute os resultados dos testes realizados. Foram feitos dois tipos de testes:

- *Testes de Desempenho*: têm como objetivo verificar o impacto das técnicas apresentadas no desempenho da aplicação.
- *Testes de Usuário*: têm como objetivo testar junto aos usuários a solução apresentada, colhendo destes informações que permitam verificar a eficácia daquela.

4.1
Testes de Desempenho

Como explicado no capítulo anterior, as técnicas de navegação desenvolvidas neste trabalho dependem basicamente da estrutura cubo de distâncias. A construção e a atualização desse é custosa do ponto de vista computacional: é necessárias que mais seis passadas de renderização sejam feitas para cada quadro exibido na tela. Para cenas com grande quantidade de geometria, isso pode representar uma perda significativa de desempenho, a ponto disso ser sentido também pelo usuário.

Outro fator que pode influenciar no desempenho é a precisão que se deseja obter nas estimativas fornecidas pelo cubo de distâncias. Quanto maior a resolução das imagens referentes às faces do cubo, maior é a precisão. Entretanto, isso também aumenta o tempo necessário para que a placa gráfica renderize todo o cubo, além de também aumentar o tempo necessário para realizar a varredura desse.

Dessa forma, os testes descritos nesta seção têm como objetivo verificar como a presença do cubo de distâncias influencia no desempenho da aplicação.

4.1.1 Ambiente de Desenvolvimento

O desenvolvimento do visualizador SiVIEP segue atualmente o modelo de orientação a objetos. A linguagem usada é C++, sendo a motivação principal o desempenho que esta permite alcançar em aplicações gráficas.

A renderização dos modelos é feita usando o grafo de cena OpenSG [OpenSG, 2010]. Esse foi escolhido por ser *multi-thread safe* e prover técnicas para renderização distribuída. Além disso, ele é extensível, permitindo o uso de outras bibliotecas de renderização mais especializadas.

Várias outras bibliotecas são usadas na construção do SiVIEP, mas não são mencionadas aqui uma vez que elas não influem no objetivo deste trabalho e nem nos resultados dos testes mostrados aqui.

4.1.2 Ambiente de Testes

Para a realização dos testes foi utilizado um computador com a seguinte configuração:

- Hardware
  - Processador Intel Core 2 Duo E7500, composto de dois núcleos de 2.93 GHz
  - Placa Mãe Intel DG41TY
  - Memória RAM de 2 Gb, DDR2 800 MHz
  - Placa de vídeo NVIDIA GeForce 9800 GT com 512 Mb de memória
- Software
  - Sistema Operacional Windows 7 32 bits
  - Compilador Microsoft Visual Studio 2005
  - OpenSG 2.0

4.1.3 Resultados

Como mencionado anteriormente, os testes de desempenho descritos aqui visam responder basicamente a duas questões:

1. De que maneira a construção e a atualização do cubo de distâncias influi na taxa de quadros por segundo (FPS) que o SiVIEP consegue exibir.

2. De que maneira a resolução adotada no cubo de distâncias, ou seja, a precisão com que este representa o ambiente, impacta no FPS da aplicação.

Para obter a resposta da questão 1 foi feito o seguinte procedimento: várias cenas, cada uma delas com uma quantidade diferente de geometria, foram carregadas no SiVIEP. Para cada uma delas, durante um intervalo de 60 segundos, foi medido a cada quadro o FPS corrente e esse valores foram armazenados em um arquivo. Ao final, esses foram usados para calcular o FPS médio durante o intervalo. Isso é feito duas vezes, sendo uma sem o processamento do cubo de distâncias e a outra onde isso é incluído. Nesse último caso a resolução do cubo foi fixada em 64×64. A resolução de tela é fixada em 740×655.

Por simplicidade, a criação de cenas com quantidades diferentes de geometria foi feita carregando-se várias vezes o modelo de uma plataforma de petróleo. Por exemplo, a primeira cena possui apenas uma plataforma, a segunda possui duas, a terceira três e assim por diante. Uma vez que se conhece o número de triângulos de uma plataforma, é possível determinar facilmente o tamanho em geometria das diferentes cenas usadas nos testes.

A Tabela 4.1 mostra os resultados obtidos com esse teste. Cada linha da tabela corresponde a uma cena diferente. A primeira coluna indica o número de modelos de plataforma carregados, enquanto que a segunda mostra o número total de triângulos correspondente. As terceira e quarta colunas mostram respectivamente o FPS resultante da renderização da cena com (Auto) e sem (Manual) a presença das soluções propostas. A última coluna contém a razão entre os valores da quarta e terceira colunas, e permite verificar mais facilmente o quanto de desempenho é perdido por conta do cubo de distâncias.

A análise da Tabela 4.1 mostra que o processamento do cubo de distâncias provoca uma queda de desempenho na aplicação que varia entre aproximadamente 3 e 2 vezes, para uma resolução de 64×64. Para cenas relativamente pequenas (de 1 a 10 plataformas), a diferença de desempenho é mais acentuada. Entretanto, considerando-se 30 fps como a taxa de quadros por segundo

	N. de triângulos	Auto (fps)	Manual (fps)	Manual / Auto
1	1.03779×10^5	236.923	728.029	3.071
2	2.07558×10^5	175.890	433.816	2.46
3	3.11337×10^5	130.512	323.661	2.48
4	4.15116×10^5	106.902	252.215	2.36
5	5.18895×10^5	89.831	207.759	2.31
10	1.03779×10^6	50.903	109.388	2.15
20	2.07558×10^6	29.944	57.866	1.93
30	3.11337×10^6	19.913	39.395	1.97
40	4.15116×10^6	15.388	29.945	1.95
50	5.18895×10^6	12.781	24.694	1.93
60	6.22674×10^6	10.757	20.815	1.93
70	7.26453×10^6	9.334	17.945	1.92
80	8.30232×10^6	8.165	15.765	1.93
90	9.34011×10^6	7.289	14.105	1.93
100	1.03779×10^7	6.574	12.748	1.94

Tabela 4.1: Resultados de desempenho com (Auto) e sem (Manual) o processamento do cubo de distâncias.

suficiente para a renderização em tempo real, nessas cenas pode-se dizer que essa queda não é perceptivel ao usuário. Isso é verdade também até o número de 20 plataformas carregadas, quando o desempenho cai para aproximadamente 30 fps. Após esse número, a razão entre Manual e Auto se estabiliza em aproximadamente 2 vezes e o usuário já é capaz de perceber os efeitos da queda de desempenho.

Na obtenção da resposta para a questão 2, o tamanho da cena foi fixado em uma única plataforma e foram verificados os FPSs resultantes de seu carregamento para diferentes resoluções do cubo. Para uma dada resolução, executou-se o SiVIEP com e sem a varredura feita ao fim da atualização do cubo. Dessa forma, é possível verificar também o impacto que essa varredura tem sobre a aplicação. A resolução de tela foi mantida em 740×655.

O resultado desse teste é mostrado na Tabela 4.2. A primeira coluna dessa tabela indica as diferentes resoluções de cubo testadas. As segunda e terceira colunas indicam respectivamente os FPSs médios da aplicação com e sem a varredura realizada no fim da atualização do cubo de distâncias. Na quarta coluna, R1 é a razão entre os valores da terceira e segunda colunas e permite identificar qual o peso da varredura sobre a queda de desempenho gerada. Por fim, na última coluna, R2 é a razão entre o FPS da aplicação sem as técnicas apresentadas no capítulo anterior (da Tabela 4.1, esse valor é 728.029 fps) e o FPS com a presença dessas.

Analisando-se os valores referentes a R2, percebe-se que o aumento da resolução do cubo de distâncias produz uma queda significativa no desempenho

Resolução	Com Varredura (fps)	Sem Varredura (fps)	R1	R2
32×32	270.644	272.323	1.01	2.69
64×64	238.413	256.103	1.07	3.05
128×128	179.443	198.978	1.11	4.06
256×256	88.926	99.852	1.12	8.19
512×512	30.974	35.826	1.16	23.53
1024×1024	8.765	10.404	1.19	83.17

Tabela 4.2: Resultados de desempenho considerando diferentes resoluções para o cubo de distâncias.

do sistema. No fim de cada uma das seis passadas de renderização necessárias para atualizar o cubo de distâncias, uma imagem, correspondente a uma das faces desse, deve ser enviada da placa gráfica para a CPU. Logo, quanto maior a resolução dessas imagens, maior será a quantidade de dados que devem ser enviados e, portanto, maior a queda de desempenho.

Da mesma forma, pode-se ver através dos valores de R1 que o peso da varredura do cubo na queda do desempenho cresce de acordo com a resolução escolhida. Isso ocorre pois, quanto maior a resolução, maior o número de posições que devem ser verificadas. Deve-se levar em conta ainda que essa etapa de varredura é feita em CPU e, portanto, não tira vantagens da paralelização fornecida pela placa gráfica.

Considerando-se os resultados obtidos, conclui-se que o cubo de distâncias deve ter a mínima resolução possível a fim de garantir a precisão requerida. Para o SiVIEP, a resolução de 32×32 não foi suficiente e foi preciso usar uma resolução maior, de 64×64. Por exemplo, para os testes de usuários realizados, um cubo de resolução 32×32 não foi capaz de representar as plataformas da Figura 4.1, considerando o ponto de vista inicial mostrado nessa figura.

4.2 Testes de Usuário

As técnicas apresentadas no capítulo anterior têm como objetivo principal auxiliar os usuários na tarefa de exploração dos ambientes virtuais. Do ponto de vista do usuário isso deveria resultar numa experiência de navegação mais confortável e menos sujeita a erros.

Para comprovar que isso acontece, foram feitos testes de usabilidade com dois grupos de usuários, um com 5 pessoas e o outro com 7 pessoas, a fim de colher suas opiniões sobre as soluções desenvolvidas. O planejamento e a condução desses testes foi feita com base nas orientações apresentadas em [Tullis and Albert, 2008].

4.2.1 Perfil do Usuários

Para a realização dos testes foram recrutadas doze pessoas, divididas em dois grupos:

- Usuários Avançados
  - Têm experiência no uso de aplicações de visualização 3D.
  - Possuem conhecimento de modelagem 3D.
  - Usam aplicações de visualização 3D pelo menos 1 vez por mês.
- Usuários Não-Avançados
  - Têm pouca experiência com aplicações de visualização 3D, sendo essa limitada ao uso de alguns jogos eletrônicos.
  - Não possuem conhecimento sobre modelagem 3D
  - Não usam aplicações 3D com frequência

Das doze pessoas, sete pertencem ao primeiro grupo enquanto as outras cinco pertencem ao segundo. Em comum, todas essas pessoas possuem as seguintes características:

- Possuem idades entre 20 e 30 anos.
- Pertencem ao sexo masculino.
- Não tinham contato com o SiVIEP antes da realização do teste.

Por último, as pessoas do grupo de usuários avançados são nomeadas aqui por PA1, PA2, PA3, PA4, PA5, PA6 e PA7. Para o grupo não-avançado, os usuários de teste são identificados por PN1, PN2, PN3, PN4 e PN5.

4.2.2 Procedimentos Adotados

Para cada pessoa, pediu-se primeiramente que elas lessem e assinassem um termo de compromisso (ver Apêndice A), dando seu consentimento para a realização do teste. Esse documento é uma garantia para a pessoa de que sua identidade não será revelada e que os dados colhidos serão usados unicamente para fins de pesquisa e desenvolvimento. Em seguida foi feita uma descrição geral do SiVIEP, com a apresentação do aplicativo em si, assim como de suas funcionalidades.

O teste consistiu basicamente em colocar as pessoas para usar duas versões diferentes do SiVIEP:

- *Automática*: essa versão tem suporte às soluções discutidas no capítulo anterior. O usuário não precisa se preocupar com o ajuste de velocidade e há tratamento de colisão. Também não é necessário o uso explícito da ferramenta de centro de rotação. A seta indicadora também é incluída nessa versão, assim como o ajuste automático dos planos de corte.
- *Manual*: essa versão não contem nenhuma das técnicas mencionadas, com exceção do ajuste automático dos planos de corte. O ajuste de velocidade na ferramenta *voar* deve ser feito manualmente através do uso do scroll do mouse e o usuário deve tomar cuidado para não colidir com os modelos. A ferramenta de centro de rotação deve ser usada sempre antes do início da inspeção do objeto com a ferramenta *examinar*.

Antes da pessoa começar a usar uma determinada versão foram passadas algumas instruções do funciomento das ferramentas de navegação daquela versão. Por exemplo, na versão manual, foi pedido aos usuários que evitassem atravessar os modelos da cena. Foi explicado também para esses como proceder com o ajuste manual de velocidade e com a ferramenta de centro de rotação. Já no caso da versão automática, os usuários foram informados que não havia necessidade de se preocuparem com os aspectos mencionados.

O ambiente de teste para as duas versões consistiu de uma cena composta por duas plataformas de extração de petróleo A e B, dispostas entre si de uma certa distância (ver Figura 4.1). A câmera é colocada inicialmente em uma posição de onde é possível visualizar as duas plataformas. Foi pedido que os usuários navegassem até a plataforma A usando a ferramenta *voar*. Uma vez lá, a pessoa deveria explorar a parte interna da plataforma a fim de escolher três objetos quaisquer para serem examinados com a ferramenta *examinar*. Por último, foi pedido que o usuário navegasse dessa plataforma até a plataforma B.

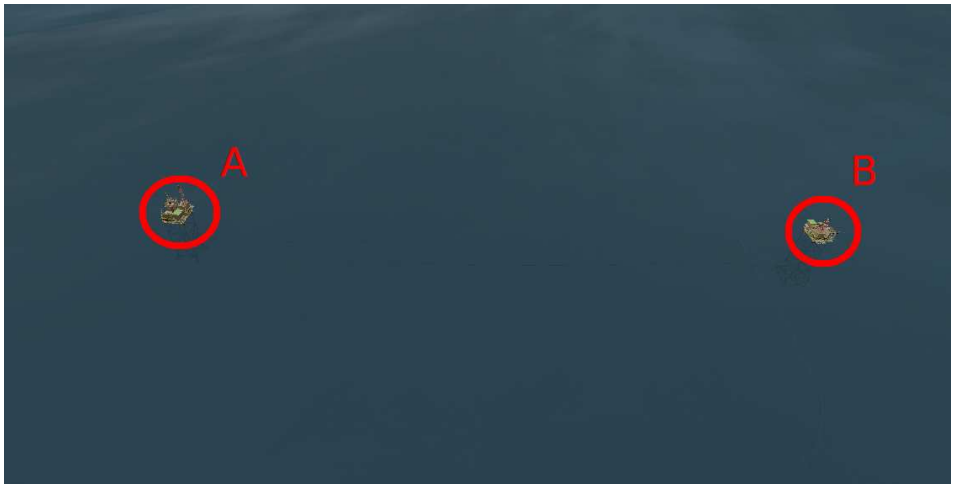


Figura 4.1: Cena do ambiente de teste

O tempo para completar o teste não foi fixado. O objetivo disso foi evitar que as pessoas se sentissem pressionadas em realizar as tarefas em um curto espaço de tempo, modificando assim suas decisões com relação a navegação no ambiente. Dessa forma, também a definição do término do teste fica a cargo do usuário.

Ao fim do uso de cada versão foi pedido que as pessoas preenchessem um formulário (ver Apendice B) com o objetivo de colher suas impressões sobre as ferramentas usadas. Esse formulário é composto das seguintes afirmações, identificadas aqui por A1, A2, A3, A4 e A5:

- A1: *"Eu não tive dificuldades com o ajuste de velocidade da ferramenta voar"*
- A2: *"Eu consegui realizar as tarefas sem colidir com o ambiente."*
- A3: *"Eu não tive dificuldades com a ferramenta de centro de rotação."*
- A4: *"Eu não me senti desorientado em nenhum momento ao navegar pelo ambiente virtual."*
- A5: *"Eu me senti confortável usando as ferramentas de navegação."*

Embaixo de cada uma dessas afirmações há uma escala composta de dez números, indo de 1 a 10, onde 1 significa que o usuário discorda completamente da afirmação e 10 que ele concorda plenamente com essa. No fim do formulário é deixado um espaço onde o usuário pode descrever textualmente suas impressões gerais e justificar as notas dadas.

Depois de usadas as duas versões, foi pedido aos usuários que respondessem ainda um último questionário, composto por apenas duas questões discursivas, identificadas por Q1 e Q2:

– Q1: *"Qual das duas abordagens você gostou mais: das técnicas de navegação automatizadas ou as manuais? Por quê?"*

– Q2: *"Com relação a abordagem escolhida como preferida, existe na sua opinião algo que possa ser melhorado? Se sim, o que é e por que precisa ser melhorado?"*

A pergunta Q1 é considerada uma das mais importantes do teste, devido ao seu caráter conclusivo. Nela, a pessoa é incentivada a refletir acerca das duas versões usadas, estabelendo assim uma comparação mental entre as duas experiências.

Por último, a ordem em que as versões são apresentadas aos usuários não foi igual para todos. Por exemplo, a primeira pessoa a realizar o teste usou inicialmente a versão manual e em seguida a automática. Já a segunda, fez o teste na ordem inversa. Esse padrão foi seguido até o último usuário. Esse procedimento foi realizado com o objetivo de minimizar o efeito de aprendizado provocado pelo uso da primeira versão sobre a segunda.

4.2.3 Resultados

Grupo Não-Avançado

As Tabelas 4.3 e 4.4 mostram os resultados obtidos da aplicação dos testes de usabilidade para o grupo de usuários não-avançados. Elas contêm as notas dadas por cada usuário às 5 afirmações mencionadas na seção anterior. A Tabela 4.3 indica as notas referentes ao uso da versão Manual, enquanto a Tabela 4.4 contem as notas para a versão automática. A última coluna de cada tabela indica a nota média para cada afirmação. Para permitir uma melhor visualização dos resultados gerais, essas médias são colocadas lado a lado na Figura 4.2. O intervalo de confiança usado na geração desse gráfico é de 90 %.

Afirmativas	PN1	PN2	PN3	PN4	PN5	Média
A1	7	8	10	9	5	7.8
A2	6	5	7	7	2	5.4
A3	6	9	8	7	3	6.6
A4	4	8	7	9	1	5.8
A5	5	8	9	10	4	7.2

Tabela 4.3: Resultados do teste de usabilidade para a versão Manual (grupo de usuários não-avançados).

A análise desses resultados mostra que a versão automática obteve notas superiores em relação à versão manual, como já era esperado. Ao usarem

Afirmativas	PN1	PN2	PN3	PN4	PN5	Média
A1	9	10	10	10	9	9.6
A2	10	10	10	10	10	10
A3	10	8	10	10	10	9.6
A4	10	10	9	10	10	9.8
A5	10	9	10	10	10	9.8

Tabela 4.4: Resultados do teste de usabilidade para a versão Automática (grupo de usuários não-avançados).

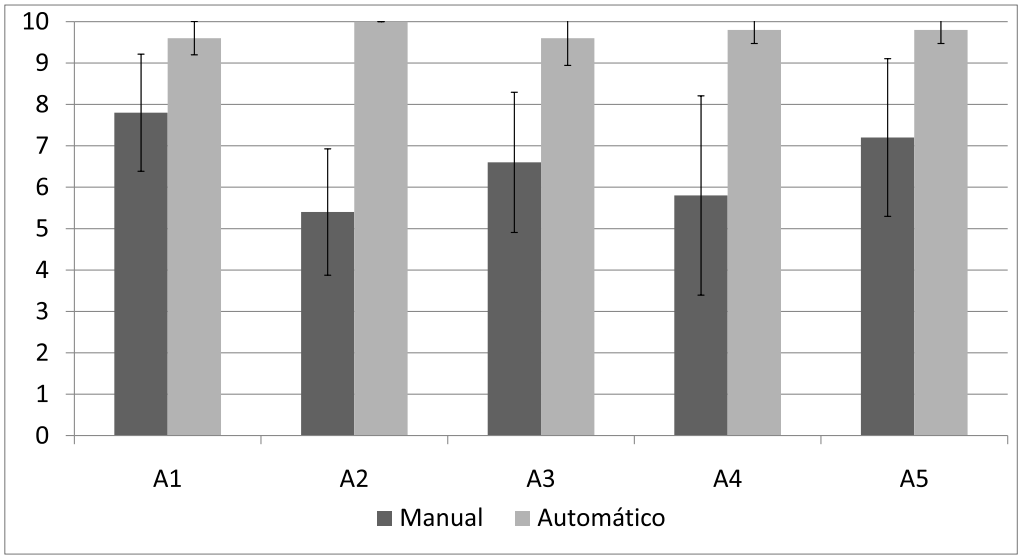


Figura 4.2: Resultados comparativos entre as versões Manual e Automática (grupo de usuários não-avançados).

a versão manual, os usuários reclamaram principalmente da dificuldade em controlar a velocidade da câmera de modo a não colidir com o ambiente. Um dos usuários, por exemplo, disse que "o programa é um pouco brusco e dificulta a utilização de quem não tem hábito ou destreza com jogos e programas em 3D". A falta de tratamento de colisão provocou situações em que algumas pessoas se sentiram perdidas. Nesse último caso, dois dos cinco usuários chegaram a mencionar que gostariam de desistir da tarefa.

Com relação à questão $Q1$ presente no último questionário, todos os usuários desse grupo responderam preferir a versão automática em relação à manual. Dentre as justificativas, a maioria dos usuários disse que as técnicas fornecidas pela versão automática tornaram a navegação mais simples, menos propensa a erros e mais fácil de controlar. Para a questão $Q2$, nenhum desses usuários achou necessário algum tipo de melhoria para a versão escolhida.

Grupo Avançado

As Tabelas 4.5 e 4.6 mostram os resultados obtidos da aplicação dos testes de usabilidade para o grupo de usuários avançados. O formato de apresentação dos resultados é o mesmo da seção anterior. Também é fornecida a Figura 4.3 mostrando um comparativo entre as notas dadas para versão manual e as da versão automática. O intervalo de confiança é de 90 %.

Afirmativas	PA1	PA2	PA3	PA4	PA5	PA6	PA7	Média
A1	10	9	7	7	10	9	9	8.71
A2	3	7	7	7	10	8	7	7
A3	7	8	10	3	5	5	7	6.42
A4	3	9	6	5	10	6	8	6.71
A5	6	8	8	8	10	8	7	7.85

Tabela 4.5: Resultados do teste de usabilidade para a versão Manual (grupo de usuários avançados).

Afirmativas	PA1	PA2	PA3	PA4	PA5	PA6	PA7	Média
A1	7	5	10	5	9	9	9	7.71
A2	10	9	9	10	10	9	10	9.6
A3	10	6	6	10	10	9	10	8.71
A4	6	9	7	10	10	8	9	8.42
A5	10	6	9	9	10	9	9	8.86

Tabela 4.6: Resultados do teste de usabilidade para a versão Automática(grupo de usuários avançados).

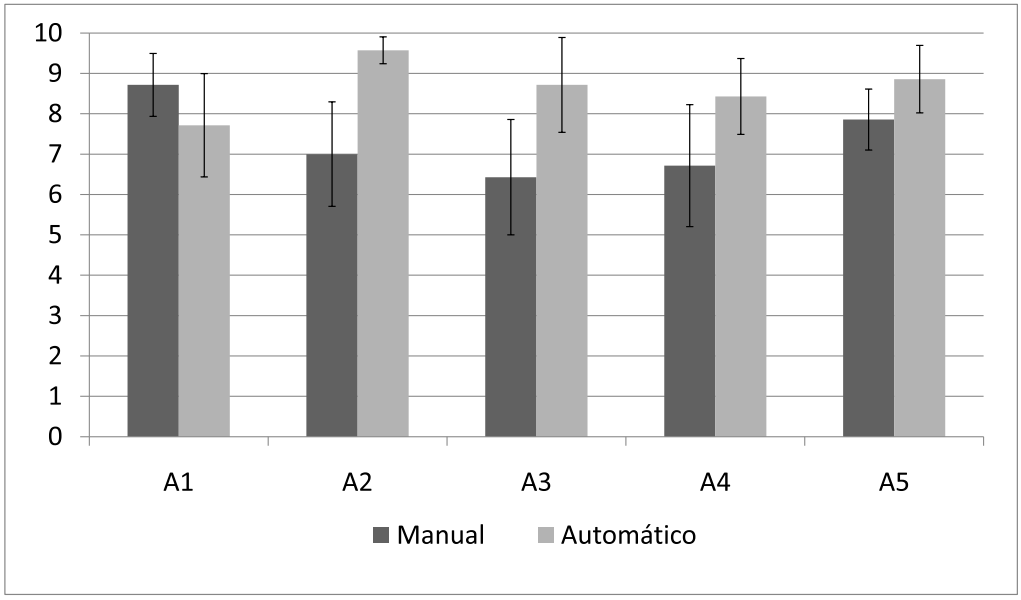


Figura 4.3: Resultados comparativos entre as versões Manual e Automática (grupo de usuários avançados).

Das cinco afirmativas, todas tiveram notas maiores na versão Automática, com exceção da afirmativa A1. Essa afirmação tem por objetivo avaliar o ajuste automático de velocidade da ferramenta *voar*. Dessa forma, os resultados acima poderiam levar a crer que tal ajuste não trouxe os benefícios esperados para esse grupo de usuários. Entretanto, a análise das justificativas das notas e os comentários gerais feitos por cada uma dessas pessoas no fim do formulário revelam alguns pontos interessantes.

Praticamente todos os usuários que deram uma nota menor para o ajuste de velocidade automático reclamaram que, ao se aproximar demais de algum objeto, a câmera ficava muito lenta e demorava até que eles conseguissem se afastar novamente do objeto. Isso provocou nesses usuários uma sensação de impaciência, que foi agravada pelo fato de não ter sido dada nenhuma opção de controle que os permitisse *momentaneamente* aumentar a velocidade. Ao mesmo tempo, esses mesmos usuários disseram que a existência do ajuste automático era boa pois permitiu que eles se preocupassem menos com os controles, além de ter evitado em alguns momentos a ocorrência de erros. Isso confere com os comentários gerais feitos pelo grupo de usuários não-avançados. Resumindo, os usuários avançados sentiram na verdade a falta de algum controle que os possibilitasse realizar um ajuste mais "personalizado" em determinados momentos, e que ao mesmo tempo considerasse o ajuste automático de velocidade.

Outro fator que contribuiu para as notas maiores da versão Manual em A1 é o modo como os testes foram conduzidos. Antes de iniciar o teste com a versão Manual, foi pedido aos usuários que evitassem ultrapassar (colidir com) os objetos da cena. Isso fez com que eles fossem bem mais cuidadosos com relação ao ajuste da velocidade. Esse comportamento resultou em notas maiores para a versão Manual, uma vez que as possíveis dificuldades que seriam normalmente sentidas não ocorreram devido ao maior cuidado com o ajuste. Isso, entretanto, acabou influenciando negativamente no conforto sentido pelos usuários: no sentido de evitar colisões com os modelos, eles eram obrigados a parar várias vezes a fim de reajustar a velocidade de navegação. Isso é incômodo e acaba tirando a atenção do usuário ao que realmente interessa, que é o próprio ambiente a ser explorado.

Os maiores ganhos da versão Automática foram com relação às afirmações A2 e A3 que tem, respectivamente, o objetivo de avaliar a eficácia do tratamento de colisão e do centro de rotação automático. Nos comentários gerais, essas duas técnicas foram as que mais receberam comentários positivos. Os usuários perceberam facilmente os efeitos criados por essas técnicas e ficaram bastante satisfeitos com seus resultados. Em especial, alguns usuários afirma-

ram que por conta da existência dessas técnicas, eles deixaram de cometer erros e que isso permitiu que eles se preocupassem menos com as questões de interface. Por fim, as notas dadas pelos usuários desse grupo para as afirmações A4 e A5 mostram que esses se sentiram mais confortáveis e passaram por menos situações de desorientação quando no uso da versão Automática, assim como aconteceu com o grupo de usuários não-avançados.

Com relação ao último questionário respondido por cada usuário, seis dos sete usuários responderam a questão Q1 afirmando que preferem a versão Automática em relação à Manual. Apenas o usuário PA2 optou por essa última versão. Em sua justificativa, ele cita o problema com relação ao ajuste automático de velocidade e a falta de retorno ao usuário sobre onde o ponto de centro de rotação está localizado quando no ajuste automático.

Por fim, na questão Q2, que pede que os usuários deem sugestões sobre o que precisa ser melhorado na versão escolhida, todos eles pediram a existência de algum controle que os permita momentaneamente aumentar a velocidade ou mesmo, atravessar objetos.

4.3 Análise final

Dos resultados acima pode-se concluir que as técnicas apresentadas neste trabalho melhoram a experiência de navegação dos usuários. Dentre os doze participantes dos testes de usabilidade, onze preferiram a versão da aplicação que continha as técnicas apresentadas nesse trabalho.

O uso de dois grupos de participantes com perfis distintos permite algumas observações interessantes. O grupo de usuários avançados foi mais crítico em seus comentários, além de ter sido capaz de lidar melhor com os controles na versão manual. Isso já era esperado, dada a experiência desses em aplicações 3D. Também por esse motivo, esse grupo sugeriu melhorias para alguns aspectos da navegação na versão automática, algo que não aconteceu no caso dos usuários não-avançados.

O ganho da versão automática em relação à manual é maior para o grupo de usuários não-avançados, e revela que esses tiveram naturalmente uma dificuldade maior com relação aos controles manuais. Esse perfil de usuário revelou precisar de um apoio maior para a realização das tarefas de navegação.

Concluindo, os resultados permitem supor que usuários avançados têm uma tendência maior a preferir soluções que facilitem seu uso, mas que não sejam completamente automatizadas a ponto de não permitirem um certo nível de controle. Quando usuários não-avançados são considerados, o caso é o oposto: esses, devido à sua menor experiência, preferem abordagens

que minimizem a necessidade de ajustes nos parâmetros das ferramentas de navegação.

Considerando os pedidos feitos pelos grupo de usuários avançados com relação ao ajuste automático de velocidade, conclui-se também que esse deve ser modificado no sentido de dar ao usuário um controle maior da velocidade da câmera. Isso, entretanto, deve ser feito de forma a não influenciar negativamente nas vantagens providas pelo ajuste atual, uma vez que este foi preferido pelos usuários não-avançados.

Por último, os dois perfis de participantes escolhidos são equivalentes aos de usuários do SiVIEP: o objetivo desse é servir de apoio tanto para tarefas de análise e planejamento, sendo assim usado por usuários mais avançados como engenheiros, como também para apresentações de empreendimentos na área petrolífera onde é possível que pessoas com menos experiência usem a aplicação.

5
Conclusão

Este trabalho propôs técnicas com o objetivo de auxiliar e facilitar a tarefa de navegação em ambientes virtuais 3D. A motivação dessa pesquisa foram os problemas de interação identificados no visualizador SiVIEP. Apesar desse ter servido como base para o desenvolvimento e testes das soluções apresentadas, essas podem ser utilizadas em qualquer aplicação que envolva navegação em ambientes virtuais 3D.

As técnicas baseiam-se na construção e manutenção de uma estrutura de dados chamada de *cubo de distâncias*, proposta inicialmente por [McCrae et al., 2009]. Através dessa estrutura é possível obter informações sobre a proximidade dos objetos em relação à câmera, que por sua vez possibilitam o funcionamento das técnicas discutidas nos capítulos anteriores.

Foram propostas e implementadas as seguintes técnicas:

1. *Ajuste automático da velocidade de navegação ao utilizar a ferramenta Voar*: a velocidade da câmera é dependente da escala em que a mesma se encontra. Esse ajuste é feito de forma suave e contínua e permite que o usuário navegue em diferentes escalas sem ter que manualmente reajustar a velocidade de navegação.

2. *Ajuste automático dos planos de corte near e far*: os planos de corte são continuamente ajustados de forma a impedir cortes indesejados e a otimizar o uso do *buffer de profundidade*.

3. *Detecção e tratamento de colisão*: ao navegar pelo ambiente usando a ferramenta *Voar*, a câmera tem sua trajetória suavemente alterada de forma a evitar colisões com objetos da cena.

4. *Determinação do centro de rotação utilizado na ferramenta Examinar*: não é necessário que o usuário tenha que explicitamente escolher um novo ponto de centro de rotação toda vez que for usar a ferramenta *Examinar*. A determinação do novo ponto é feita de forma automática.

5. *Seta indicadora*: tem como objetivo impedir que os usuários fiquem perdidos no caso em que nada é visto na tela. Quando isso acontece, uma seta aparece na tela indicando o local onde a cena se encontra.

Como principais contribuições desse trabalho podem ser citadas:

- A construção do cubo de distâncias de forma que esse sempre fique orientado com a câmera, permitindo assim agregar novas informações que possiblitaram o desenvolvimento de algumas das técnicas apresentadas.
- O uso da *media exponencial móvel* com o objetivo de diminuir os efeitos negativos provocados pelo ajuste automático da velocidade de navegação quando esse é feito somente com base na distância do ponto mais próximo à câmera.
- O uso da função de colisão suave proposta por [McCrae et al., 2009] em conjunto com a ferramenta *Voar*, que resultou em um modo de navegação assistido.
- Um método que visa garantir o estado correto do ponto de centro de rotação da ferramenta *Examinar*, com base nas informações providas pelo cubo de distâncias.

A eficácia dessas técnicas foi verificada através da condução de testes de usabilidade. Esses consistiram basicamente em apresentar duas versões diferentes para os usuários: uma com suporte às técnicas mencionadas (automática) e outra sem (manual). Dos doze participantes, onze preferiram a versão automática. Segundo essas pessoas, a soluções fornecidas simplificaram o uso da aplicação e permitiram que eles se concentrassem na tarefa de exploração do ambiente virtual sem a preocupação constante em ajustar parâmetros de navegação ou interface.

Também foram realizados testes com o objetivo de verificar qual o impacto do processamento do cubo de distâncias no desempenho da aplicação. Os resultados mostram que, para um cubo com resolução de 64×64, a perda de desempenho varia entre 3 e 2 vezes em relação a versão sem o cubo de distâncias. Apesar de significativa, essa queda não impede o uso confortável da aplicação, mesmo para cenas relativamente grandes. Por exemplo, para uma cena com 20 plataformas (mais de 2 milhões de triângulos), o FPS da aplicação foi de aproximadamente 30, que ainda é um valor aceitável do ponto de vista do usuário.

5.1 Trabalhos Futuros

Um problema muito frequente em aplicações 3D e que não foi explorado em profundidade aqui consiste em como prover aos usuários maneiras de se localizarem no ambiente virtual. Aplicativos como o *Google Earth*, por exemplo, tentam resolver esse problema através da exibição de marcas ou até mesmo de anotações que se tornam visíveis em determinados momentos. [Pierce and Pausch, 2004] também seguem essa linha e criaram um sistema onde os objetos da cena são representados por ícones que simbolizam uma característica específica do modelo apontado. A existência de uma solução desse tipo em conjunto com as técnicas propostas aqui poderiam melhorar ainda mais a experiência de uso de ferramentas de navegação para ambientes virtuais.

Outra linha de pesquisa que pode ser seguida é o desenvolvimento de novas técnicas baseadas no cubo de distâncias. Por exemplo, uma ideia que ainda não foi explorada seria tentar usar as informações presentes no cubo para ajustar os parâmetros do efeito de estereoscopia.

Por último, pode-se pensar em técnicas que permitam diminuir o impacto da construção do cubo de distâncias no desempenho da aplicação. No trabalho de [McCrae et al., 2009], por exemplo, a geometria utilizada na construção do cubo é uma simplificação da original. A implementação dessa solução pode permitir que cenas maiores sejam carregadas sem que o usuário sinta os efeitos da queda de desempenho provocada pelo processamento do cubo de distâncias.

Referências Bibliográficas

[Baciu and Wong, 1997] Baciu, G. and Wong, W. S.-K. (1997). "Rendering in object interference detection on conventional graphics workstations". In *PG '97: Proceedings of the 5th Pacific Conference on Computer Graphics and Applications*, page 51, Washington, DC, USA. IEEE Computer Society. 2.3

[Baciu et al., 1998] Baciu, G.; Wong, W. S.-K.; and Sun, H. (1998). "RECODE: an image-based collision detection algorithm". In *Computer Graphics and Applications, 1998. Pacific Graphics '98. Sixth Pacific Conference on*, pages 125–133. 2.3

[Bederson and Hollan, 1994] Bederson, B. B. and Hollan, J. D. (1994). "Pad++: a zooming graphical interface for exploring alternate interface physics". In *UIST '94: Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 17–26, New York, NY, USA. ACM. 2.2

[Bederson et al., 1994] Bederson, B. B.; Stead, L.; and Hollan, J. D. (1994). "Pad++: advances in multiscale interfaces". In *CHI '94: Conference companion on Human factors in computing systems*, pages 315–316, New York, NY, USA. ACM. 2.2

[Calomeni and Celes, 2006] Calomeni, A. and Celes, W. (2006). "Assisted and automatic navigation in black oil reservoir models based on probabilistic roadmaps". In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 175–182, New York, NY, USA. ACM. 1, 1.2.3, 2.3, 2.5, 3.4

[Darken and Sibert, 1993] Darken, R. P. and Sibert, J. L. (1993). "A toolset for navigation in virtual environments". In *UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 157–165, New York, NY, USA. ACM. 1, 1.1, 2.4

[Darken and Sibert, 1996] Darken, R. P. and Sibert, J. L. (1996). "Wayfinding strategies and behaviors in large virtual worlds". In *CHI '96: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 142–149, New York, NY, USA. ACM. 1.2.4

[Drucker and Zeltzer, 1994] Drucker, S. M. and Zeltzer, D. (1994). "Intelligent Camera Control in a Virtual Environment". In *Proceedings of Graphics Interface 94*, pages 190–199. 2.1

[Fitzmaurice et al., 2008] Fitzmaurice, G.; Matejka, J.; Mordatch, I.; Khan, A.; and Kurtenbach, G. (2008). "Safe 3D navigation". In *I3D '08: Proceedings of the 2008 symposium on Interactive 3D graphics and games*, pages 7–15, New York, NY, USA. ACM. 1, 2.1, 2.4, 3.5

[Furnas and Bederson, 1995] Furnas, G. W. and Bederson, B. B. (1995). "Space-scale diagrams: understanding multiscale interfaces". In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 2.2

[Goldin, 1982] Goldin, Sarah E., T. P. W. (1982). "Simulating Navigation for Spatial Knowledge Acquisition". *Human Factors*, 24(4), pp. 457–471(15). 1

[Hermann and Celes, 2005] Hermann, R. and Celes, W. (2005). "Posicionamento automatico de cameras em ambientes virtuais dinamicos". In *Proceedings of the IV Workshop on Games and Digital Entertainment of the Brazilian Symposium on Computer Games and Digital Entertainment*. 2.1

[Howard, 1981] Howard, James H., K. S. M. (1981). "Memory and Perception of Cartographic Information for Familiar and Unfamiliar Environments". *Human Factors*, 23(4), pp. 495–503(9). 1

[Jakobsen, 2001] Jakobsen, T. (2001). "Advanced Character Physics". In *Proceedings of Game Developers Conference*, pages 383–401. 2.1

[Jul and Furnas, 1998] Jul, S. and Furnas, G. W. (1998). "Critical zones in desert fog: aids to multiscale navigation". In *UIST '98: Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 97–106, New York, NY, USA. ACM. 1, 2.4

[Kopper et al., 2006] Kopper, R.; Ni, T.; Bowman, D. A.; and Pinho, M. (2006). "Design and Evaluation of Navigation Techniques for Multiscale Virtual Environments". In *VR '06: Proceedings of the IEEE conference on Virtual Reality*, pages 175–182, Washington, DC, USA. IEEE Computer Society. 2.2, 2.2, 2.5

[Li and Chou, 2001] Li, T. and Chou, H. (2001). "Improving Navigation Efficiency with Artificial Force Field". In *Proceedings of 2001 14th IPPR Conference on Computer Vision, Graphics, and Image Processing*, Taiwan. 2.3, 2.3, 2.5, 3.4

[Mackinlay et al., 1990] Mackinlay, J. D.; Card, S. K.; and Robertson, G. G. (1990). "Rapid controlled movement through a virtual 3D workspace". *SIGGRAPH Comput. Graph.*, 24(4), pp. 171–176. 1.2.1, 2.1, 2.2, 3.2

[McCrae et al., 2009] McCrae, J.; Mordatch, I.; Glueck, M.; and Khan, A. (2009). "Multiscale 3D navigation". In *I3D '09: Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 7–14, New York, NY, USA. ACM. 1.3, 2.2, 2.2, 2.3, 2.5, 3.1, 3.1, 3.2, 3.3, 3.3, 3.4, 3.4, 5, 5, 5.1

[Mine, 1995] Mine, M. (1995). "Virtual Environment Interaction Techniques". Technical report, UNC Chapel Hill CS Dept. 2.4

[OpenGL Architecture Review Board, 1997] OpenGL Architecture Review Board, C. (1997). *OpenGL reference manual (2nd ed.): the official reference document to OpenGL, Version 1.1*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 1.2.2

[OpenSG, 2010] OpenSG (2010). "http://www.opensg.org". 4.1.1

[Perlin and Fox, 1993] Perlin, K. and Fox, D. (1993). "Pad: an alternative approach to the computer interface". In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 57–64, New York, NY, USA. ACM. 2.2

[Pierce and Pausch, 2004] Pierce, J. S. and Pausch, R. (2004). "Navigation with Place Representations and Visible Landmarks". In *VR '04: Proceedings of the IEEE Virtual Reality 2004*, pages 173–180, Washington, DC, USA. IEEE Computer Society. 2.4, 5.1

[Schlumberger, 2010] Schlumberger (2010). "The Oilfield Glossary: Where the Oil Field Meets the Dictionary, http://www.glossary.oilfield.slb.com/default.cfm". 1.1

[Stoakley et al., 1995] Stoakley, R.; Conway, M. J.; and Pausch, R. (1995). "Virtual reality on a WIM: interactive worlds in miniature". In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 265–272, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 2.4

[Tan et al., 2001] Tan, D. S.; Robertson, G. G.; and Czerwinski, M. (2001). "Exploring 3D navigation: combining speed-coupled flying with orbiting". In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 418–425, New York, NY, USA. ACM. 2.1

[Thibault and Naylor, 1987] Thibault, W. C. and Naylor, B. F. (1987). "Set operations on polyhedra using binary space partitioning trees". In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 153–162, New York, NY, USA. ACM. 2.3

[TNPetroleo, 2010] TNPetroleo (2010). Sala de aula - equipamentos submarinos, http://www.tnpetroleo.com.br/sala_de_aula/equipamentos-submarinos. 1.1

[Tullis and Albert, 2008] Tullis, T. and Albert, W. (2008). *Measuring the user experience: collecting, analyzing, and presenting usability metrics*. Elsevier, Amsterdam. 4.2

[Ware and Fleet, 1997] Ware, C. and Fleet, D. (1997). "Context sensitive flying interface". In *SI3D '97: Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 127–130, New York, NY, USA. ACM. 2.2

[Ware and Osborne, 1990] Ware, C. and Osborne, S. (1990). "Exploration and virtual camera control in virtual three dimensional environments". *SIGGRAPH Comput. Graph.*, 24(2), pp. 175–183. 2.1

[Xiao and Hubbold, 1998] Xiao, D. and Hubbold, R. (1998). "Navigation guided by artificial force fields". In *CHI '98: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 179–186, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co. 2.3, 2.5, 3.4

A
Termo de Compromisso

Termo de Consentimento para Avaliação de Técnicas de Navegação no SiVIEP

Você foi convidado(a) pelo Tecgraf – *Tecnologia em Computação Gráfica* – um laboratório de pesquisas em Computação Gráfica do Departamento de Informática da PUC-Rio, para participar de um teste de avaliação das técnicas de navegação presentes no aplicativo SiVIEP (Sistema de Visualização Integrado de Exploração e Produção)

Neste teste, buscamos verificar a satisfação dos usuários em relação a essas técnicas. Por esta razão, solicitamos seu consentimento para a realização deste teste. Para tanto, é importante que você tenha algumas informações:

1. Os dados coletados durante o teste destinam-se **estritamente** a atividades de pesquisa e desenvolvimento.

2. Os resultados dese teste serão usados unicamente para fins acadêmicos. A divulgação destes resultados pauta-se no **respeito a sua privacidade** e o **anonimato** dos mesmos é preservado em quaisquer documentos elaborados.

3. O consentimento para o teste é uma escolha livre, feita mediante a prestação de todos os esclarecimentos necessários sobre a pesquisa.

4. A realização do teste pode ser interrompida a qualquer momento, segundo a disponibilidade do participante. Neste caso, a equipe se compromete a descartar o teste para fins da avaliação a que se destinaria.

De posse das informações acima, gostaríamos que você se pronunciasse acerca
do teste.

( ) Dou meu consentimento para sua realização.
( ) Não autorizo sua realização.

Rio de Janeiro, _____ de Janeiro de 2010.

Participante

Nome: _____
Assinatura: _____

Avaliador:

Nome: Daniel Ribeiro Trindade
Assinatura:_____

B
Questionários Usados no Teste de Usabilidade

Nome:_____

Eu não tive dificuldades com o ajuste de velocidade da ferramenta voar.

Discordo plenamente	1	2	3	4	5	6	7	8	9	10	Concordo plenamente

Eu consegui realizar as tarefas determinadas sem colidir com o ambiente.

Discordo plenamente	1	2	3	4	5	6	7	8	9	10	Concordo plenamente

Eu não tive dificuldades com a ferramenta de centro de rotação.

Discordo plenamente	1	2	3	4	5	6	7	8	9	10	Concordo plenamente

Eu não me senti desorientado em nenhum momento ao navegar pelo ambiente virtual.

Discordo plenamente	1	2	3	4	5	6	7	8	9	10	Concordo plenamente

Eu me senti confortável usando as ferramentas de navegação.

Discordo plenamente	1	2	3	4	5	6	7	8	9	10	Concordo plenamente

Comentários gerais. Use as linhas a seguir para descrever as suas impressões sobre as ferramentas testadas e o que o motivou a dar as notas acima.

_____

_____

_____

_____

_____

_____

_____

_____

_____

Questionário Final

Nome:_____

Qual das duas abordagens você gostou mais: as técnicas de navegação automatizada ou as manuais? Por quê?

_____
_____
_____
_____
_____
_____
_____
_____
_____

Com relação a abordagem escolhida como preferida, existe na sua opinião algo que possa ser melhorado? Se sim, o que é e por quê precisa ser melhorado?

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____