UNIVERSIDADE FEDERAL DO MARANHÃO UFMA
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CIÊNCIA DA COMPUTAÇÃO

MÉRCIO PIO DE CARVALHO

UM ALGORITMO PARA VISUALIZAÇÃO 3D DE CONTORNO DE TENSÃO

São Luís
2012

MÉRCIO PIO DE CARVALHO

UM ALGORITMO PARA VISUALIZAÇÃO 3D DE CONTORNO DE TENSÃO

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos para a obtenção parcial do grau de bacharel em Ciência da Computação.

Orientador: Prof.º Dr. Anselmo Cardoso de Paiva.

São Luís
2012

Carvalho, Mércio Pio de

Um algoritmo para visualização 3D de contorno de tensão / Mércio Pio de Carvalho. – 2012.

75 f.

Orientador: Anselmo Cardoso de Paiva.

Monografia (Graduação em Ciência da Computação) - Universidade Federal do Maranhão, 2012.

1. Sistema elétrico de potência 2. Realidade virtual 3. DB4O 4. Papervision3D 5. Técnicas de visualização tridimensional I. Título

CDU 004: 621.311

UM ALGORITMO PARA VISUALIZAÇÃO 3D DE CONTORNO DE TENSÃO

MÉRCIO PIO DE CARVALHO

Monografia aprovado em 04 de dezembro de 2012.

BANCA EXAMINADORA

Prof. Dr. Anselmo Cardoso de Paiva (Orientador)

Ma. Simara Vieira da Rocha(Examinadora)

Me. Geraldo Braz Junior(Examinador)

À minha família, e em especial, Négile Ribeiro Carvalho *(in memorian)*.

AGRADECIMENTOS

Agradeço a Deus;

Meus pais e irmãs;

Familiares;

Ao Prof.º Dr. Anselmo Cardoso de Paiva, pela dedicação nas correções e orientações neste período de aprendizado;

Ao amigo Daniel Lima Gomes Júnior pela ajuda durante o longo período de desenvolvimento deste trabalho;

E a todos aqueles que torceram pela conclusão deste trabalho.

“Quem sabe concentrar-se numa coisa e insistir nela como único objetivo, obtém, ao fim e ao cabo, a capacidade de fazer qualquer coisa”.

Mahatma Gandhi

RESUMO

Este trabalho tem como objetivo o desenvolvimento de algoritmos de visualização tridimensional para contornos de tensão procedentes dos sistemas elétricos de potência que resultou na implementação das técnicas sugeridas mediante a utilização de tecnologias alternativas para as camadas de armazenamento e apresentação de dados. Apresenta-se também a arquitetura utilizada na construção de um ambiente de visualização tridimensional que possibilitou a execução dos algoritmos baseados nas técnicas propostas resultando: na conciliação das técnicas elaboradas com as tecnologias alternativas sugeridas e também na análise de informações inerentes aos sistemas elétricos de potência sob vários aspectos.

Palavras-chave: Sistema Elétrico de Potência. Realidade Virtual. DB4O. Papervision3D. Técnicas de Visualização Tridimensional.

ABSTRACT

This work aims to develop algorithms for three-dimensional visualization of voltage's contours from the electric power systems presenting as results: the implementation of the suggested techniques through the use of alternative technologies for storage layer and data presentation. It also presentes the architecture used in the construction of a three-dimensional viewing environment that enables the implementation of this algorithms based on the proposed techniques making possible: the conciliation of elaborated techniques with the suggested alternative technologies; and also the analysis of information about electric power systems in several differents aspects.

Keywords: Electric Power System. Virtual Reality. DB4O. Papervision3D. Three-Dimensional Visualization Techniques.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento e Durabilidade
AMF	*Action Message Format*
C/S	Cliente/Servidor
CPU	*Central Processing Unit*
CRUD	*Create, Read, Update e Delete*
CSG	Geometria sólida construtiva
DB4O	*DataBase4Objects*
DDA	*Digital Differential Analyzer*
EJBs	*Enterprise JavaBeans*
GPL	*General Public License*
HTML	*HyperText Markup Language*
J2EE	*Java 2 Platform Enterprise Edition*
JAR	*Java Archive*
JEE	*Java Enterprise Edition*
JSP	*Java Server Pages*
MVC	*Model-View-Controller*
PV3D	Papervision3D
RIA	*Rich Internet Application*
RV	Realidade Virtual
SD	Sistema Distribuído
SDK	*Software Development Kit*
SQL	*Structured Query Language*
SRC	Sistema de Referência de Câmera
SRD	Sistema de Referência do Dispositivo
SRO	Sistema de Referência do Objeto
SRP	Sistema de Referência de Projeção
SRU	Sistema de Referência do Universo
VRLM	*Virtual Reality Modeling Language*
X3D	*eXtensible 3D*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Constituindo um dos maiores e mais complexos sistemas criados pelo homem e considerada uma das grandes heranças da história recente da humanidade, a indústria de energia elétrica é um dos ícones da civilização moderna. Apesar de sua história secular, alguns de seus problemas persistem até hoje como é o caso da dificuldade de gerenciar informações por ela geradas, ora por quantidade, ora por complexidade. Por isso, aprimorar a visualização de informações é de extrema importância para tomada de decisões por repercutir diretamente na ordem social (GOMES JR, 2010) e constituir um caminho para tais melhorias.

Em um sistema elétrico de potência, a tarefa de aplicar procedimentos de correção quando equipamentos falham ou limites de operação são violados, entre outras casualidades exige uma resposta imediata do responsável por esta ação. Responder em tempo hábil depende então de uma rápida assimilação dos dados que se tem em mãos. A visualização de informações dentro deste escopo, por si só, motiva um análise mais aprofundada.

O ato de decidir o procedimento adequado a ser tomado compete à figura do operador do sistema que sempre espera ter todas as informações necessárias e suficientes para fazê-lo. Um sistema elétrico de potência possui uma quantidade grande de variáveis analisadas em um contexto temporal ou espacial. Quando a localização torna-se um fator a ser considerado, o uso de mapas ajuda a compreensão por parte do operador uma vez que dados geo-referenciados[1] os correlacionam ao mundo real. Por último, um conjunto de dados dependendo de seu arranjo, fornece percepções distintas de um mesmo elemento e isso interfere diretamente no tipo de visualização adequada.

Dentre as alternativas de auxílio no suporte à decisão estão aquelas que utilizam meios computacionais. Há muito que o homem se utiliza da interface do computador para compreender uma determinada realidade. Contudo, o avanço tecnológico possibilitou a introdução de técnicas de visualização utilizando ambientes tridimensionais. A utilização de menus e botões das interfaces 2D é substituída pela manipulação direta de elementos no mundo tridimensional.

[1] Geo-Referenciado: Representação da uma localização em sistema de coordenadas ou outro tipo de identificação no contexto geográfico.

Na vanguarda da comunicação envolvendo o homem e computador encontram-se aqueles que utilizam realidade virtual. Dentre as vantagens do seu emprego estão o favorecimento ao entendimento humano, pois o cérebro compreende o mundo físico pelo aspecto tridimensional além de proporcionar uma experiência com interação e sensação de imersão em um grau maior (WIEGMANN et al, 2003).

A introdução de mapas geográficos no desenvolvimento de ambientes tridimensionais tem a capacidade de atender algumas necessidades básicas citadas a respeito de sistemas elétricos de potência. Uma é a visualização das informações em si, realizada agora em 3D; a outra é que este tipo de recurso gera uma compreensão mais rápida, pois é intuitiva.

Este trabalho explora o potencial que há no uso da realidade virtual a partir da criação de ambientes tridimensionais no suporte de sistemas elétricos de potência.

1.1 Objetivo Geral

Este trabalho tem como objetivo o desenvolvimento de técnicas especializadas na exibição de contorno de tensão através de visualizações tridimensionais com a introdução de mapas e elementos geo-referenciados neste contexto.

1.2 Objetivos específicos

Os objetivos específicos deste trabalho são:

a) elaborar as técnicas de visualização;
b) implementar as técnicas de visualização para contorno de tensão;
c) criar um sistema de suporte para exibição das técnicas desenvolvidas.

A elaboração das técnicas é o primeiro passo a ser executado. Serão levadas em consideração as informações relativas a duas entidades presentes em um sistema elétrico de potência: subestações e linhas de força. Com base nelas, estruturar-se-ão as alternativas para determinar a visualização.

Definidas as técnicas, a ideia seguinte diz respeito a sua implementação. Depois de concebidas, a próxima etapa será transcrevê-las na forma de linguagens de programação apropriadas e tal conveniência explica-se pelo interesse de posterior utilização atribuindo um caráter mais prático às técnicas e não somente teórico.

A praticidade dos algoritmos criados só poderia vir por meio de sua execução. O terceiro objetivo específico é a elaboração de um sistema de suporte para a visualização das técnicas desenvolvidas para apreciação e consideração de viabilidade de incorporação a um sistema real, voltado para um operador de sistema elétrico de potência possibilitando o acesso à visualização de uma considerável quantidade de dados em um ambiente com características especiais inerentes ao mundo virtual.

1.3 Organização do trabalho

Este trabalho está organizado em mais cinco capítulos descritos ordenadamente a seguir. No segundo capítulo serão discutidos alguns conceitos importantes relacionados ao sistema elétrico de potências e realidade virtual. O terceiro capítulo destina-se à descrição e organização dos recursos tecnológicos empregados ao longo do processo de desenvolvimento do trabalho. O quarto, diz respeito ao desenvolvimento e implementação das técnicas de visualizações tridimensionais. O capítulo cinco apresenta o ambiente criado como suporte para a exibição das visualizações tridimensionais para contorno de tensão, abordadas no capítulo quatro. Insere-se no contexto deste capítulo uma visão geral do comportamento e da arquitetura deste ambiente de visualização e também uma breve avaliação do mesmo. Finalizando, compete ao sexto capítulo as considerações finais a respeito do trabalho, uma análise geral, destacando possíveis contribuições para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo abordaremos de forma específica os conceitos básicos necessários para compreensão do trabalho desenvolvido.

2.1 Sistema de potência

Um sistema moderno elétrico de potência é constituído de forma genérica por três entidades primárias. Cada entidade por sua vez também é formada por outros elementos. A primeira é o sistema de geração, responsável pela produção da energia. Em seguida, a energia é transferida para subestações responsáveis pela redistribuição local através de linhas de alta voltagem, tal tarefa é de competência do sistema de transmissão – a segunda entidade. De uma subestação local até o cliente, todo o aparato que realiza esta atividade é parte integrante do sistema de distribuição – terceira entidade (CROW; SHETTY, 2004).

Quando falamos da potência, em geral, referimo-nos à rapidez com que certa quantidade de energia é transformada ou com que rapidez realiza-se o trabalho. O potencial elétrico é nada mais que a capacidade de um corpo eletrificado de realizar trabalho. Uma tensão elétrica ocorre quando há uma diferença de potencial elétrico entre dois pontos. Como a potência está condicionada ao instante, chamamos de potência instantânea. Entende-se por potência ativa a parte da potência instantânea passível de utilização para consumo. Medimos a potência ativa em Watts (W). Existe uma parte da potência instantânea que é absorvida pelos componentes indutivos do próprio sistema de potência, denominada de potência reativa, sua medição é feita em Volt-Amperes Reativos (VAR). Além disso, importante, é considerar o somatório das potências ativas e reativas de todo o sistema, a potência aparente, medida em Volt-Amperes (VA).

A energia elétrica é produzida nas estações de energia e distribuída aos clientes através de extensas redes de distribuição. Cliente no sistema de potência é uma denominação geral para as entidades que necessitam da energia elétrica. Usuários domésticos, comerciais e industriais tanto de pequeno, médio e grande porte são exemplos de clientes. Tal classificação é necessária porque embora todos recebam energia e façam parte do sistema em si, é em função do usuário final que

se define a composição do subsistema de distribuição. Se por um lado o subsistema de geração e transmissão compartilhados pelos usuários presentes no mesmo sistema elétrico de potência é único, não se pode dizer o mesmo em relação ao subsistema de distribuição. São nas subestações de energia elétrica através de transformadores que acontece, por exemplo, a redução de altas voltagem alternadas para níveis próprios para consumo de usuários domésticos ou mudanças na frequência. Também estão presentes nas subestação equipamentos responsáveis no controle contra falhas, proteção contra raios e curto-circuitos.

Da geração de energia até seu consumo, o sistema moderno elétrico de potência é uma complexa estrutura que não só leva em conta o aparato para manipulação direta da energia, mas também agrega-se à infraestrutura um sistema de suporte para mantê-la em funcionamento. Tal complexidade torna necessária uma busca constante por tecnologias que permitam uma melhor visualização sobre dados para gerenciamento dos sistemas de potências.

2.2 Realidade virtual e interface homem-computador

O contínuo desenvolvimento e aperfeiçoamento tanto de hardware quanto de software influencia diretamente a forma com a qual nos comunicamos com computadores. Se antes uma maneira de validar a autenticidade por parte de um usuário em uma tela de login era através da inserção de senha, hoje, com implementação de algoritmos de análise de imagens é possível executar esta mesma tarefa através de reconhecimento facial. É notório que um caráter cada vez mais realístico da indústria de jogos tem impressionado o público graças ao aumento do poder computacional das chamadas “placas gráficas”. As duas situações mencionadas anteriormente correspondem a melhorias de software e hardware respectivamente e comprovam que o conjunto de capacidades das quais podemos hoje esperar de um computador está cada vez maior. Quando tratamos da relação homem-computador e de seu fluxo de informação bilateral obrigatoriamente admitimos a existência do elemento interface entre eles e este também é passível de desenvolvimento tecnológico. Dentre os mais avançados tipos de interface que dispomos, podemos citar a realidade virtual (RV) abordada aqui por ser umas das técnicas utilizadas neste trabalho.

A RV pode ser entendida como um meio do usuário visualizar, manipular

e interagir com computadores e dados complexos permitindo sintetizar tridimensionalmente aspectos ambientais do mundo real da maneira mais verissímil possível levando até mesmo o indivíduo a adotar essa interação como uma de suas próprias realidades temporais. Este última sensação diz respeito a capacidade da máquina de suprir, no mais alto grau possível, o usuário sob o ponto de vista do nível de imersão, interação e envolvimento (AUKSTAKALNIS; BLATNER,1992).

Analisar um sistema de RV pela óptica da imersão significa entender o quão profundo é o sentimento de presença do participante dentro do próprio ambiente virtual, são utilizados objetos como por exemplo capacetes de visualização e salas de projeções. É comum classificar um sistema de RV como simplesmente imersivo ou não-imersivo. Tem-se como exemplo uma caverna digital ou uma aplicação que utiliza monitor respectivamente, a mais comum e menos imersiva. Caso o sistema de RV tenha a capacidade de se modificar em resposta a alguma ação do usuário estamos afirmando que este proporciona algum tipo de interação oferecendo a possibilidade do usuário de interatuar com o sistema. Os sistemas que suportam interação são mais sofisticados que os sistemas de imersão e detêm maior êxito em oferecer a impressão ao usuário de se sentir inserido no sistema. Uma mão biônica e luvas são dispositivos utilizados para este propósito. A maneira como o usuário comporta-se em relação ao sistema e sua motivação equivalem ao grau de envolvimento e se resumem a fazê-lo adotar uma postura ativa ou passiva.

O desenvolvimento de sistemas de RV assim como qualquer outro sistema computacional deve atender a um ou alguns propósitos específicos e são em função destes propósitos que os sistemas de RV são categorizados (CASAS, 1996):

a) sistema de Telepresença: a imersão é percebida através de sons e respostas aos movimentos realizados no mundo real;

b) Realidade virtual em Segunda pessoa (*unencumbered systems*): envolve respostas em tempo real no qual o usuário vê a si mesmo, pois é colocado em frente a um monitor no qual é projetada sua imagem somada a outra imagem utilizada como fundo ou ambiente (*chromayed*);

c) sistema Desktop: englobam as aplicações que mostram um conjunto

de imagem 2D ou 3D na tela de um monitor de computador;

d) sistemas de imersão: como o próprio nome diz focam na submersão ou exploração do usuário de maneira mais estreita do mundo virtual, apresentam ao usuário imagens tridimensionais geradas através de telas como por exemplo visores HMD's (*Head Mounted Displays*).

Atualmente dispomos de muitas tecnologias para criação de ambientes virtuais como por exemplo o *OpenGL*, *Java3D*, *X3D*, *VRML*. A aplicação desenvolvida para atender a proposta de trabalho de visualização 3D para contornos de tensão utiliza uma tecnologia denominada *Papervision3D* (PV3D) e que mais tarde será abordada no terceiro capítulo.

Ainda tratando da aplicação, com base nos conceitos apresentados relacionados a RV, a mesma é definida como sistema não-imersivo, pouco interativo e de pouco envolvimento voltado para a exibição de informações de imagens 3D em um ambiente de rede de computadores e, por isso, também classificado como sistema *desktop*.

2.3 Visualização tridimensional

Como vimos anteriormente, é muito estreita a relação entre um sistema RV e visualizações tridimensionais, então é fundamental o domínio de meios que possibilitem visualização 3D a uma aplicação que tenha este pré-requisito. Tais meios fazem parte do processo de visualização e neste tópico serão abordados mais verticalmente.

Desenvolver aplicativos com visualização 3D requer criar e transformar geometricamente objetos 3D. No entanto, nossa aplicação, previamente definida como do tipo *desktop*, utiliza monitores de visualização que exibem apenas imagens bi-dimensionais. Assim, necessitamos de procedimentos que nos permitam representar objetos 3D nesse espaço 2D, ou seja, no sistema de coordenadas da tela. Outra tarefa está relacionada a capacidade de criar visualizações dos objetos 3D, a partir de qualquer ângulo, ou posição no espaço. O primeiro problema, a representação, refere-se aos métodos de modelagem e projeção e o segundo às definições dos parâmetros e transformações de visualização.

2.3.1 Modelagem 3D

Existe um conjunto de técnicas em computação gráfica que permitem transformar as informações a respeito de um modelo, contidas em uma estrutura de dados, em uma imagem formada na tela de um monitor. Um modelo de um objeto representantado computacionalmente tem sua descrição geométrica correspondente em estrutura de dados (COHEN; MANSSOUR, 2006).

Exitem três principais abordagens e métodos para executar a tarefa de modelagem (CARRETO, 2012):

a) representação de superfície: consiste na representação através de um conjunto superfícies que separam o interior do objeto do espaço exterior;

- aramado (*wire-frame*): representação de um objeto somente através de suas arestas;
- malha de polígonos: a estrutura do objeto é representada por faces planas, geralmente um triângulo;
- superfícies paramétricas: define as coordenadas dos pontos através de equações paramétricas;
- superfícies implícitas: representadas por superfícies nas quais são conhecidas as funções que as formam. Exemplo: esfera ,parabolóide, hiperbolóide etc.;

b) representação de volume: é a representação através de um conjunto de sólidos contíguos e menores que não se sobrepõem;

- voxels: o sólido é decomposto em células idênticas, pequenos cubos chamados de *voxels*, formando uma grade tridimensional;
- quadtrees/octrees: Assim como os *voxels*, também são métodos de enumeração de ocupação espacial , mas com a diferença que cada

cubo é armazenado em uma árvore com 4 ou 8 filhos para *quadtrees* ou *octrees* respectivamente além de poder variar de tamanho. Recursivamente podem ser subdivididos em cubos menores de igual tamanho até que uma condição de preenchimento total ou vazio seja encontrada tornando o método mais eficiente para representar objetos não uniformes;

- geometria sólida construtiva (CSG):utiliza sólidos mais simples para composição de sólidos mais complexos. Cada objeto é armazenado em uma árvore.Nas folhas encontram-se sólidos primitivos como esfera, cubo, cone e nos nós operadores booleanos como intersecção, união e diferença;

c) modelagem procedural: engloba métodos alternativos à modelagem geométrica tradicional, descreve os objetos de maneira algorítmica visando representa-los em forma e comportamento. Uso de fractais, sweeps, sistema de partículas entre outros para reproduzir elementos como vegetação, gases, líquidos, fogo etc.

2.3.2 Processo de visualização

A modelagem é somente o início do conjunto de atividades envolvidas no processo de visualização 3D. Tendo os objetos devidamente representados em uma forma computacional, outros elementos são levados em conta no processo e cada um precisa de um sistema de referência. Um sistema de referência não é nada mais que um sistema de coordenadas com a função de tornar o mais natural e convincente possível a representação. No processo de visualização são levados em conta os objetos contidos na cena, o próprio universo ou parte onde se localiza o objeto, a câmera, a imagem projetada e o dispositivo de exibição. Cada elemento deste possui um sistema de referência descritos a seguir (BADAN, 2012):

a) sistema de referência do objeto (SRO): define as coordenadas dos vértices de cada objeto contido em uma cena tridimensional objetivando definir geometricamente os objetos;

b) sistema de referência do universo (SRU): é o sistema de coordenadas global que define as coordenadas dos vértices dos objetos da cena incluindo a câmera. Neste espaço é possível saber a relação espacial entre todos os objetos do cenário;

c) sistema de referência da câmera (SRC): é o espaço definido por um sistema de coordenadas associado à câmera virtual. Define os parâmetros da câmera em relação ao mundo, como posição, orientação, distância focal, etc.. Pode ser associado à uma projeção perspectiva ou a uma projeção paralela;

d) sistema de referência de projeção (SRP): tem a função de definir as coordenadas dos vértices dos objetos de uma cena tridimensional em projeção bidimensional. Definido por um sistema de coordenadas 2D no plano de projeção, onde se localiza a tela virtual;

e) sistema de referência de dispositivo (SRD): associado à superfície de exibição do dispositivo de saída gráfica, é utilizado para transferência da imagem 2D criada no SRP, para um dispositivo de visualização bidimensional.

3 REFERENCIAL TECNOLÓGICO

No capítulo anterior tratamos dos conceitos básicos que fundamentam a proposta de trabalho cujos conhecimentos são essenciais para desenvolvimento de qualquer projeto que proveja visualizações 3D para sistemas de potência. Visualizações estas que neste trabalho serão realizadas a partir de um aplicativo também contido na proposta de trabalho e apresentado posteriormente. Se o capítulo antecessor alicerça o desenvolvimento de forma genérica, deste ponto em diante, cada vez mais, o foco será os aspectos de unicidade relacionados ao trabalho. Este capítulo destina-se a abordar tecnologias específicas utilizadas na composição do desenvolvimento do aplicativo proposto para a visualização do contorno de tensão.

3.1 Procedimentos e recursos

Desenvolver algoritmo para a visualização do contorno de tensão e demonstrá-lo em uma aplicação, isto é, significa acrescentar ao aplicativo uma funcionalidade que é nada mais que a de oferecer opção de visualizar o contorno de tensão. De forma implícita, a tarefa principal, na verdade, é fragmentada em sub-tarefas. Uma delas está relacionada ao algoritmo de visualização, mas somente em conjunto com as demais oferecerá a função principal. Em outras palavras, todas as tarefas presentes no contexto da visualização são parte do que chamamos de sistema. A partir de agora usaremos o termo sistema, bem mais conveniente, para o conjunto das tarefas que englobam todas as atividades que darão suporte a visualização final.

Continuando a abordar as funcionalidades, quando falamos somente das relativas à aplicação, do ponto de vista estrutural, referimos-nos à bem conhecida camada de aplicação cujo objetivo é definir regras específica para o sistema. O desenvolvimento em camadas é uma das técnicas mais comuns que os projetistas de softwares utilizam para quebrar em pedaços um sistema complexo de software (FOWLER, 2006).

Juntando-se à camada de aplicação temos a camada de dados, relacionada à persistência e gerenciamento da base de dados do sistema e a

camada de apresentação, atrelada à coleta e exibição de dados aos usuários.

Existe uma hierarquia obrigatória na composição em camadas. Nos extremos encontram-se as camadas de apresentação e dados e, entre elas, a de aplicação, uma comunicação entre a camada de apresentação e dados não é feita diretamente sem passar pela intermediária, a camada de aplicação. Cada camada pertencente a um sistema pode executar tarefas por meio de processos distintos que não necessariamente estão inseridos na mesma máquina, mas através de vários computadores interligados, portanto, distribuído. Segundo (TANENBAUM; STEEN, 2006, p. 23), um sistema distribuído (SD) é uma “[...] coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente”.

Agora que falamos de sistemas distribuídos, há uma frase associada ao cientista da computação Leslie Lamport com os seguintes dizeres: “Você sabe que você tem um sistema distribuído quando a falha de um computador do qual você nunca ouviu falar o impede de fazer qualquer coisa”. (HALDAR; ARAVIND, 2010, p. 27). Embora seja um meio “pessimista” de opinar sobre sistemas distribuídos, já que o define através de uma característica admissível, mas indesejável, a falha, traduz de forma bastante prática a disposição de um SD sob o ponto de vista de sua arquitetura. Se um computador impede alguém de fazer alguma coisa e este alguém não o utiliza diretamente, em condições de uso, o computado é um servidor e este alguém seria um cliente. Cliente/Servidor (C/S) é um tipo de arquitetura de SD e será utilizada na construção do sistema vinculado a proposta deste trabalho de desenvolvimento dos algoritmo para visualização 3D de contorno de tensão.

Como afirmado no início, este capítulo tem por finalidade explanar a respeito das tecnologias empregadas no processo de desenvolvimento do trabalho em questão. Foram discutidas apenas características teóricas. Contudo bastante pertinentes para compreensão do arranjo no qual cada tecnologia foi empregada para desempenhar cada papel, baseado na arquitetura C/S, como será descrito agora:

a) cliente: utilização da tecnologia Adobe Flex no ambiente de desenvolvimento Adobe Flex Builder 3 com adição do Papervision3D;

b) servidor: implementação em Java com utilização do servidor de aplicação (Apache Tomcat) através do ambiente de desenvolvimento

NetBeans IDE 6.7 em conjunto com BlazeDS e Database4Objects(Db4o).

O *Adobe Flex Builder 3* e o *NetBeans IDE 6.7* são dois ambientes de desenvolvimento integrado que oferecem suporte à programação "em Flex" e Java respectivamente. Quanto à linguagem Java, não há necessidade de comentários em resposta à decisão de sua participação neste projeto. Desde seu anúncio formal em 1995 pela Sun, em uma conferência, Java comprovou ser um recurso eficiente no meio comercial e também acadêmico devido a suas inúmeras características.

Adobe Flex, ou simplesmente *Flex*, é na verdade um conjunto de tecnologias que envolvem uma combinação da linguagem *ActionScript*, a de marcação MXML, similar ao HTML, um *framework* chamado de *Flex SDK* e o *Flash Player* que possibilita a visualização da aplicação.

O *BlazeDS* tem implementação feita em Java para utilização do protocolo AMF(*Action Message Format*) e efetua a comunicação entre o lado cliente(*client-side*) e servidor(*server-side*). Tipicamente uma aplicação *Flex*(escrita em *MXML* e *ActionScript*) é implantado(*deployed)* na forma de um arquivo SWF para uso no lado cliente. Os arquivos de classes Java (*Java class files*) ou conjunto de arquivos (*JAR*) são utilizados no lado servidor (*server-side*).

O *Apache Tomcat* é um servidor *web* Java, um contêiner de *servlets* de forma mais específica. É um servidor de aplicações JEE. Porém, não se enquadra na de servidor de EJBs. Em parte, ele suporta a especificação J2EE com tecnologias de *servlet* e JSP. Possui também a capacidade de funcionar integrado a um servidor *web* dedicado e provê um servidor web HTTP puramente em Java.

O desenvolvimento de sistemas utilizando o “trio”, formado pela tecnologia *Adobe Flex* na camada de apresentação, linguagem Java na camada de aplicação e *BlazeDS* como elo entre os dois primeiros, é comum devido a aspectos característicos ou mesmo estratégicos que se referem ao grupo de ferramentas conhecidas como RIA (*Rich Internet Application*) em cujo *Flex* insere-se. O termo RIA é um modelo de desenvolvimento de aplicações que separam o serviço "*back-end*" de um rica variedade de recursos no “*front-end*” na parte do cliente (MCCUNE, SUBRAMANIAM, 2009).

O foco a partir de agora é a introdução de duas tecnologias que atuam inseridas nos extremos da arquitetura Cliente/Servidor do sistema proposto: o *Db4o*, banco de dados escolhido e, o Papervision3D, "motor" das execuções das

visualizações 3D, situado na camada de apresentação.

3.2 Database4Objects (DB4O)

Discutiremos a seguir um pouco mais sobre o Database4Objects, banco escolhido para persistir os dados do sistema para a compreensão do seu manuseio e posterior integração com os demais itens que se utilizam deste.

3.2.1 Características

O Database4Objects é um banco de dados orientado a objetos de código aberto e flexível com desenvolvimento, licenças comerciais e suporte patrocinado pela Versant Corporation. O Db4o está disponível em duas licenças: a de código aberto, licença GPL[2] , que permite a fácil transferência, avaliação e uso em projetos compatíveis como GPL e uma licença de uso comercial para incorporar e distribuir Db4o em produtos não compatíveis com licença GPL.

Projetado para aplicações do tipo embarcada, cliente/servidor e desktop com bibliotecas para uso nativo em Java e .Net. Seu uso na plataforma Java requer a adição dos arquivos (.jar) para o *classpath* do projeto e importação dos pacotes a partir do com.db4o.Db4o. É um banco de dados projetado especificamente para fornecer persistência para programas orientados a objetos (PATERSON et al.,2006). Não requer nenhum tipo de mapeamento entre os objetos transientes e persistentes além de utilizar poucas linhas de código para ser manipulado independentemente da complexidade das estruturas (GUERRA, 2012).

O Db4o oferece recursos em acordo com as propriedades ACID[3], não se utiliza de consultas SQL na manipulação de objetos, eliminando a troca de orientação a objetos por performance (DB4O, 2012). Não requer muito recurso computacional, oferece rapidez de inserção, com breve tempo de aprendizado e ambiente de administração zero, pois rotinas de melhorias podem ser feitas durante a programação (GUERRA, 2012). Segundo (PARTNERS, 2012), empresas como a *Boing*, *Bosch*, *Hertz*, *BMW*, *Intel*, Seagate entre outras possuem sistemas que

[2] Licença Pública Geral: designação da licença para *software* livre idealizada por Richard Matthew Stallman em 1989

[3] Acrônimo de atomicidade, consistência, isolamento e durabilidade, características básicas desejáveis em um banco de dados.

utilizam o Db4o.

3.2.2 Operações básicas

Uma das tarefas básicas de um banco de dados é oferecer suporte as operações de inclusão, alteração, exclusão e consulta (CRUD)[4]. A seguir serão exibidas a maneira com a qual o Db4o as trata, além, é claro, de como iniciar e finalizar a comunicação com o banco.

3.2.2.1 Conexão com o banco

Para acessar ou criar um arquivo do banco de dados *Db4o* , evoca-se *Db4o.openFile()* fornecendo o caminho para o arquivo de base de dados como parâmetro, em troca, obtêm-se uma instância da classe *ObjectContainer*. Também é possível utilizar o método *Db4o.openServer()* indicando um endereço e porta onde o serviço está sendo executado.

Uma aplicação do tipo embarcada (*Embedded Server)* obrigatoriamente tem sua porta com valor 0 e maior que isto para uso via TCP. Caso haja necessidade de concessão de permissão de acesso, executa-se o comando *ObjectContainer.grantAccess()* informando usuário e a senha do banco. O ObjectContainer representa a interface primária do *Db4o*. O método *ObjectContainer close()* fecha a comunicação com o arquivo de banco de dados e libera todos os recursos a ela associados.

[4] Acrônimo de Create, Read, Update e Delete em língua inglesa.

Código fonte 1 - Conectando-se ao *bd4o.*

```
import com.db4o.Db4o;
import com.db4o.ObjectContainer;

public class Main {

   public static void main(String[]args){
      Integer porta=8000;
      String usuario="user";
      String senha="123";
      Object Containerdb=
      Db4o.openServer("//localhost/banco.yap",porta);
      db.grantAccess(usuario, senha);
      try{
      // algum código db4o
     }finally{
       db.close();
     }
   }
}
```

Fonte: Autor (2012).

3.2.2.2 Armazenando um objeto

Depois de receber o objeto da classe *ObjectContainer* definido como *db* e mostrado no exemplo anterior na Código fonte 1, simplesmente chamamos o método *ObjectContainer.set()* em nosso banco de dados, passando um objeto qualquer como um parâmetro.

Código fonte 2 – Armazenando um objeto no *Db4o*.

```
//cria um objeto do tipo cliente
Cliente cliente = new Cliente(123,"João");
//armazena o objeto cliente no banco
db.set(cliente);
```

Fonte: Autor (2012).

3.2.2.3 Recuperando objetos

O Db4o fornece três tipos de consultas para recuperação de objetos em Java:

a) Query by Example(QBE): cria-se um objeto como modelo, então o Db4o procura por objetos no banco de dados com os atributos correspondentes. Recomendado para consultas que não precisam de

operadores lógicos;

b) Native Queries(NQ): consulta na linguagem de programação nativa onde se cria um método que leva uma instância da classe que se deseja consultar como um parâmetro e retorna uma expressão booleana. O resultado da consulta é uma lista que contém todos os objetos dessa classe no banco de dados para o qual a expressão é tida como verdadeira;

c) Consulta SODA API (SODA): baseado em grafos de consulta que permite especificar os tipos de objetos a serem encontrados navegando pelas referências e impondo restrições aos campos.

Código fonte 3 – Recuperando um objeto no *bd4o* com consulta do tipo SODA.

```
//instancia query para requisitar os objetos
Query query = db.query();
//recupera todos os objetos da classe Cliente
query.constrain(Cliente.class);
```

Fonte: Autor (2012).

3.2.2.4 Atualizando objetos

Não existe um método pré-definido para atualizar um objeto. O processo é feito em etapas. Primeiro recupera-se o objeto, promove-se as alterações necessárias e por último, realiza-se uma chamada ao método *ObjectContainer.set()* novamente para armazenar o objeto modificado.

Código fonte 4 – Atualizando um objeto no *Bd4o*.

```
//seleciona o cliente João de código 123
ObjectSet<Cliente> lista = db.get(new Cliente(123,"João"));
//recupera o cliente
Cliente cliente = lista.next();
//atualiza o nome no objeto
cliente.setNome("João Pedro");
//atualiza o banco
db.set(cliente);
```

Fonte: Autor (2012).

3.2.2.5 Excluindo objetos

O processo é similar ao processo de atualização pela necessidade de

primeiro recuperar o objeto para garantir que este é o objeto a ser excluído para depois proceder à chamada do método *ObjectContainer.delete()*.

Código fonte 5 – Excluindo um objeto no *Bd4o*.

```
//seleciona o cliente João Pedro de código 123
ObjectSet<Cliente> lista =
db.get(new Cliente(123,"João Pedro"));
//recupera o cliente
Cliente cliente = lista.next();
//exclui o objeto do banco
db.delete(cliente);
```

Fonte: Autor (2012).

3.3 Papervision3D

Como já foi mencionado, utilizaremos o Papervision3D que por sua vez, oferece suporte para modelagem, visualização e transformações geométricas em um elevado grau de abstração. Por padrão, o PV3D modela objetos utilizando malha de polígonos, em forma de triângulos, chamadas de *faces*, e a projeção em perspectiva por criar um efeito semelhante à visão humana. Discutiremos, à exemplo do Db4o, apenas aspectos fundamentais e verdadeiramente necessários no desenvolvimento do sistema.

3.3.1 Características

Papervision3D é um *engine* 3D que trás o 3D para a plataforma *Adobe Flash* (WINDER, 2009) escrito e mantido por uma equipe pequena com contribuições de comunidades. Idealizado por Carlos Ulloa em novembro de 2005, tem atualmente sua versão 2.0 desde janeiro de 2009. Seu uso neste trabalho utilizou a linguagem *ActionScript* versão 3 e *Flash Player* na versão 9 e posteriormente 10. O PV3D é *open source* sob licença MIT[5] o que oferece ao utilizador autoridade para modificar e distribuí-lo para uso comercial, sem restrição.

[5] A licença MIT, também chamada de licença X ou de licença X11, é uma licença de programas de computadores, *softwares*, criada pelo Massachusetts Institute of Technology. Ela é uma licença não *copyleft* utilizada em *software* livre.

3.3.2 Conceitos e definições

Neste tópico falaremos de conceitos e significados importantes para o entendimento do PV3D para melhor compreender o processo de criação de um ambiente 3D e a maneira como o um objeto é composto.

3.3.2.1 Formas (Shapes) e objetos

As definições de formas e objetos no “mundo” do PV3D são bastantes simples. Formas estão associadas a superfícies, algo representado em 2D, um quadrado por exemplo. Agora, agregue mais cinco quadrados àquele primeiro formando um cubo. Não há possibilidade de entender que os seis quadrados são limítrofes do cubo sem considerar a existência de uma terceira dimensão. Um cubo é um objeto, algo 3D. Outras terminologias utilizadas no PV3D relativas à composição de objetos:

a) vértice (*vertex*): um ponto no espaço (2D ou 3D) que se conecta a outros sub-componentes formando uma linha ou superfície. Linhas e superfícies têm vértices(*verts*) associados a elas;
b) segmento: parte de uma linha definida por dois vértices;
c) spline: combinação de vértices e segmentos;
d) aresta (*edge*): um segmento tridimensional;
e) face: ou polígono(*polygon*), composto por arestas e vértices.

Triângulo é sem dúvida, a forma mais importante na criação de objetos 3D. A maioria dos objetos tridimensionais são criados por triângulos devido à sua fácil implementação. Muitas ferramentas de modelagem 3D criam objetos, chamados de *mesh*, a partir de triângulos, faces, utilizando a técnica *box modeling*[6]. Uma vez que o objeto está criado, o processo de renderização torna o objeto visível sendo que internamente, o Papervision3D o faz através da técnica chamada de “hidden surface drawing”.

[6] Técnica de modelagem 3D onde o modelo é criado modificando-se formas primitivas de modo a criar um rascunho do modelo final.

3.3.2.2 Hidden surface drawing

Esta técnica consiste em não desenhar superfícies que estão escondidas. Renderizar apenas as superfícies visíveis reduz o número de polígonos a serem desenhados economizando recursos da CPU e aumentando a velocidade de execução do processo. O Algoritmo do Pintor é um método muito popular e de fácil implementação usado na técnica de *hidden surface drawing*.

3.3.2.3 Algoritmo do pintor

O nome algoritmo do pintor se refere a maneira como a qual um pintor cria uma imagem. Primeiro ele pinta as partes mais distantes e depois as cobre com as partes mais próximas. Da mesma forma, o Algoritmo do Pintor classifica os polígonos por sua profundidade e então os pinta nessa ordem. Com a sobreposição, o problema das partes ocultas é resolvido e assim o faz atribuindo a cada polígono um valor de profundidade. Utilizar este método na prática, significa renderizar cada ponto de cada polígono no campo visual mesmo estando oculto. Imagine agora que alguns polígonos se sobreponham em partes diferentes. Falhas no Algoritmo do Pintor levaram ao desenvolvimento da técnicas de *Z-buffer*e*Quadrant Render Engine*, importada de outra ferramenta semelhante, chamada Away3D, presentes no PV3D.

3.3.2.4 View Frustum

Como já foi mencionado antes, a projeção perspectiva é a maneira utilizada pelo Papervision3D para criar visualizações. O view frustum corresponde ao volume que contem tudo o que é visível em uma cena 3D. Seu formato é um tronco de pirâmide, pois é limitado entre o plano traseiro e frontal tendo o centro de projeção como topo.

3.3.2.5 Culling

A visibilidade de um objeto é baseada na sua posição em relação ao view

frustum. O que está fora é invisível e não será desenhado. O culling otimiza o processo de renderização reduzindo o número de cálculos desnecessários na CPU. Quando o objeto está dentro ou fora do view frustum o processo é simples, mas quando ele encontra-se parcialmente inserido é necessário efetuar um procedimento um pouco mais complexo chamado *clipping*.

3.3.2.6 Clipping

Basicamente a operação de *clipping* consistem em calcular o produto escalar entre um vetor normal à face do polígono com um vetor da posição da câmera, no vértice do view frustum. O sinal do produto determina se o polígono será desenhado. Se o produto tem um valor negativo o polígono será desenhado, caso contrário, não o será. O Papervision3D manipula o *culling* e *clipping* de forma automática sem a necessidade de ajustes.

3.3.3 Elementos de uma cena 3D

Todas as cenas criadas com o Papervision3D tem os elementos abaixo em comum:

a) cena: é onde todos os objetos são colocados. Ela contem o ambiente 3D e gerencia todos os objetos a serem renderizados e exibidos na tela;

b) viewport: exibe a cena renderizada. É comparada à uma pintura instantânea de tudo contida no view frustum;

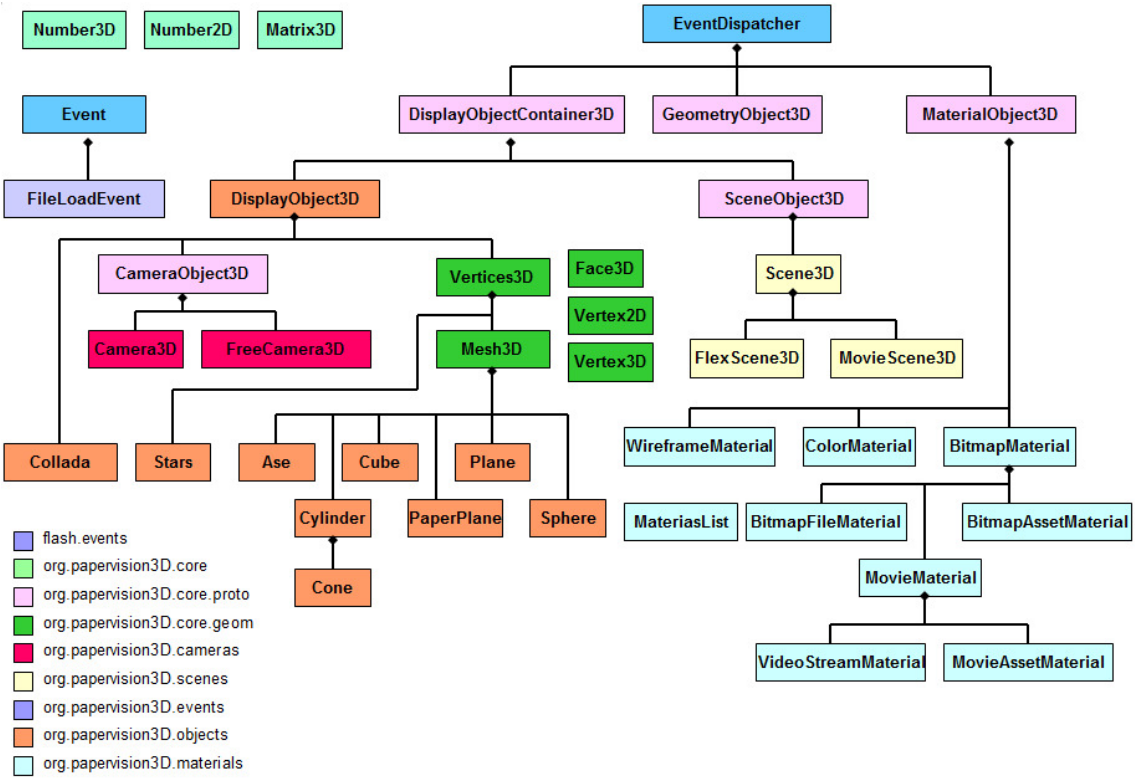
c) objeto: um objeto é uma combinação de vértices, arestas e faces que formam algo significativo para ser exibido. O Papervision3D cria objetos a partir de primitivas, estruturas embutidas ou importadas de softwares de modelagem 3D;

d) câmera: é o olho virtual dentro da cena 3D definido em relação ao ponto de vista na qual a cena será renderizada. Pode ter a posição modificada, rotacionada ou ter seu foco e zoom alterados;

e) Material: é a textura aplicada ao objeto. Pode ser uma cor, bitmap ou vídeo. Ele também interage com a luz ou sombras entre outros elementos;

f) Renderizador: desenha a sequencia de faces a partir dos dados da cena no viewport de modo a fazê-las terem um sentido visual.

Figura 1 – Diagrama de pacotes do Papervision3D em nível conceitual.



Fonte: JIDOSTAR (2007).

3.3.4 Criando uma cena 3D

Agora que falamos da teoria básica por trás do *Papervision3D*, a seguir, verificaremos, passo-a-passo, como cada elemento é utilizado para criar uma cena 3D através de um simples exemplo de animação que exibi um cone que rotaciona sobre seu eixo Y.

Neste exemplo, é utilizada uma classe que chamaremos de *AnimacaoSimples* derivada da classe Sprite para exibir o cone animado. Supondo que o projeto esteja devidamente criado assim como o *Papervision3D* inserido no

projeto, o arquivo também chamado de *AnimacaoSimples* possui a seguinte estrutura:

Código fonte 6 – Estrutura do exemplo.

```
package{
//escopo para importação de classes referenciadas
   public class AnimacaoSimples extends Sprite{
   //escopo para definição da classe
   }
}
```

Fonte: Autor (2012).

No escopo de importação de classe definimos todas as classes que serão utilizadas. A classe *AnimacaoSimples* define um novo tipo da classe *Sprite*, assim, inicialmente importamos a classe *Sprite*. Em seguida a classe *Event* para manipular um evento fundamental na animação e que será mostrado mais adiante. Por último, os elementos cena, câmera, *viewport*, renderizador e cone, todos eles devem ter suas classes inclusas. O código fica da seguinte forma:

Código fonte 7 – Declarando importações utilizadas.

```
//Importações da classe
//Superclasse de AnimacaoSimples()
Import flash.display.Sprite;
//Classe que manipula eventos diversos
Import flash.events.Event;
//classe modelo de cena
Import org.papervision3d.scenes.Scene3D;
//classe modelo de câmera
Import org.papervision3d.cameras.Camera3D;
//classe modelo de viewport
Import org.papervision3d.view.Viewport3D;
//classe modelo para o renderizador
Import org.papervision3d.render.BasicRenderEngine;
//classe modelo do tipo cone
Import org.papervision3d.objects.primitives.Cone;
```

Fonte: Autor (2012).

Todo os trechos de código a seguir encontram-se dentro da estrutura da classe AnimacaoSimples definindo-a. Começando pela declaração dos atributos básicos de cena e tendo como objeto o tipo primitivo cone.

Código fonte 8 – Declarando atributos da classe.

```
//declaração de atributos de classe
//para o elemento cena
Private var scene:Scene3D;
//para o elemento viewport
Private var viewport:Viewport3D;
//para o elemento câmera
Private var camera:Camera3D;
//para o elemento renderizador
Private var renderEngine:BasicRenderEngine;
//para o tipo primitivo cone
Private var cone:Cone;
```

Fonte: Autor (2012).

É possível notar que de todos os elemento de cena o material do objeto foi o único não declarado. Por padrão, os elementos de cena possuem uma configuração básica e não necessariamente exigem parâmetros nos construtores de suas classes. Como o material é um atributo pré-definido, foi utilizado o material original. Agora o construtor da classe:

Código fonte 9 – Implementando a cena.

```
//construtor da classe
Public function AnimacaoSimples(){
//instancia a cena
scene = new Scene3D();
//instancia a classe para câmera
camera = new Camera3D();
//instancia um tipo cone
cone = new Cone();
//insere o objeto cone na cena
scene.addChild(cone);
//instancia o viewport
viewport = new Viewport3D();
//torna o viewport visível
this.addChild(viewport);
//instancia o renderizador
renderEngine = new BasicRenderEngine();
//desenvolve a cena
renderEngine.renderScene(scene,camera,viewport);
//ação para o evento ENTER_FRAME
this.addEventListener(Event.ENTER_FRAME,render);
}
```

Fonte: Autor (2012).

Instanciamos cada elemento de cena. A vinculação dos objetos à cena é feita explicitamente para torná-los parte da cena, assim como o viewport à classe também. A cena de fato só é criada quando o método de renderização é evocado e,

por isso, importamos a classe *Event* e a utilizamos no último comando do construtor.Tal comando obriga a classe a realizar um procedimento chamado *render* cada vez que o evento *ENTER_FRAME* acontecer, assim definido:

Código fonte 10 – Definindo o tratamento para o evento.

```
//função render
private function render(e:Event):void{
   //adiciona um grau de rotação ao eixo-y
   cone.localRotationY +=1;
   //renderiza a cena novamente
   renderEngine.renderScene(scene,camera,viewport);
}
```

Fonte: Autor (2012).

O *render* é nada mais que um método que incrementa o valor do ângulo de rotação do eixo Y e refaz a renderização para exibir o cone em seu novo estado. Supondo que o evento *ENTER_FRAME* em geral aconteça em um número em torno de 30 vezes por segundo e que o evento seja tratado no mesmo período, o cone terá uma aparência bem natural de estar girando em torno de seu eixo, no caso o eixo Y. A execução do código descrito exibe algo parecido com a figura abaixo:

Figura 2 – Execução da classe *AnimacaoSimples*.

Fonte: Autor (2012).

O código do exemplo apresentado acima representa uma forma bem simples de utilizar o *Papervision3D* para compor uma cena 3D. À medida que a aplicação manipula uma cena com mais objetos ou precisa tratar mais eventos e se tornando mais complexa, novos arranjos no código serão realizados. É o caso da aplicação para a visualização 3D de contornos de tensão que será visto no próximo capítulo.

4 TÉCNICAS DE VISUALIZAÇÃO 3D UTILIZADAS

Este capítulo destina-se a demonstrar as técnicas implementadas para o trabalho proposto e suas seções explicam a heurística utilizada para resolver o problema da visualização dos dados da voltagem das subestações e linhas de tensão.

4.1 Mapeamento da superfície

A ideia inicial para dispor os elementos na visualização em 3D era criar um modelo cuja superfície representasse graficamente a localização geográfica das subestações e linhas de tensão, situadas na região nordeste do nosso país. Em relação aos aspectos geográficos, as informações relevantes seriam limitadas a representar as fronteiras estaduais excluindo itens como elevação do solo, hidrografia etc..

A solução encontrada foi utilizar no PV3D o objeto primitivo *Plane* cuja forma representa uma superfície por meio de malha de triângulos que podem ser manipulados separadamente, o que será fundamental para a representação da voltagem das subestações, além da inclusão de um mapa da região nordeste como textura para o objeto. No construtor da classe *Plane* é possível definir o número de triângulos no qual o objeto será composto. É requisitado o valor das dimensões e números de segmentos (horizontais e verticais). Para simplificar, neste caso específico, fica determinado que um único valor define a quantidade de segmentos tanto para horizontais quanto para verticais. Por exemplo, se um objeto da classe *Plane* possui valor 10 para segmentos, então este é composto 10 segmentos horizontais e 10 verticais resultando em 100 quadrados que por sua vez subdividem-se em 2 triângulos totalizando 200.

Embora a classe *Plane* esteja na camada de apresentação, a determinação das faces do objeto Plane que serão manipulados partem da camada de aplicação. Para garantir que os valores estejam em acordo, foi criado no lado servidor uma classe espelho de mesmo nome. A classe Plane (lado servidor) implementa os métodos descritos a seguir:

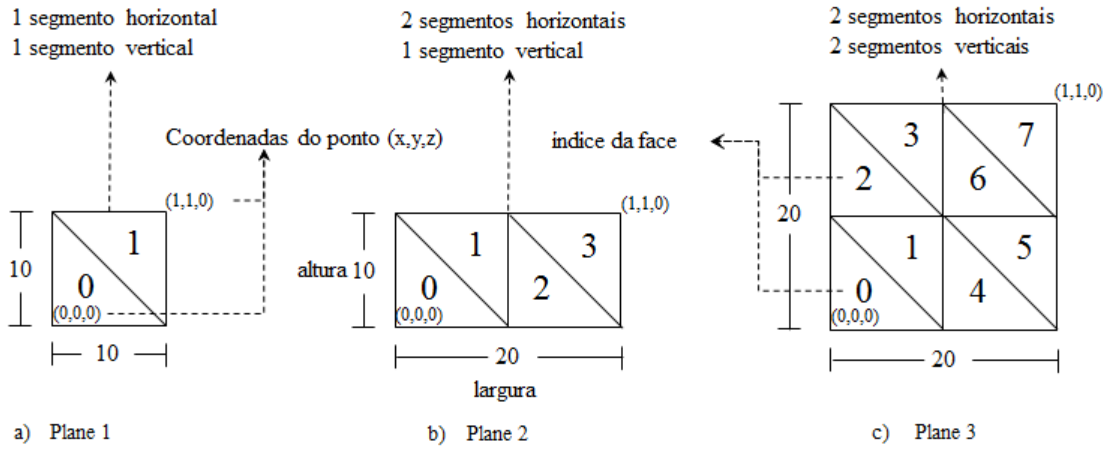
Tabela 1 – Métodos da classe Plane.

Método	Descrição
Plane	Construtor da classe. Cria a representação da malha do terreno segundo o número de segmentos, largura e comprimento da malha.
parserPoint	Calcula o índice da face onde se encontra um ponto segundo sua abscissa e ordenada.
calibra Latitude	Mapeia uma coordenada real de latitude para a grade.
calibraLongitude	Mapeia uma coordenada real de longitude para a grade.

Fonte: Autor (2012).

Antes de explicar o método *Plane.parserPoint()* devemos relembrar que a classe *Plane* implementa uma estrutura composta por faces, no caso triângulos. No *Papervision3D* cada face pode ser manipulada individualmente, basta referenciá-la em uma lista de faces contida na classe. A função do método *Plane.parserPoint()* é indicar qual seria a face desta lista cujo um determinado ponto estaria contido e o faria indicando apenas o índice desta lista onde esta face está armazenada. A figura abaixo exemplifica como é a ordem de composição de faces de um objeto tipo *Plane*:

Figura 3 - Representação gráfica da classe *Plane do Papervision3D*.



Fonte: Autor (2012).

Uma forma bastante simples de determinar qual é a face cuja área contem o ponto seria buscar na lista cada triângulo de face e aplicar algum algoritmo de intersecção, e caso houvesse algum, retornar seu índice na lista. Uma solução desta teria uma complexidade O(n) considerando que o número de iterações estaria

em função do tamanho da lista de faces e com o procedimento que determina a intersecção sendo sempre um procedimento constante.

Revendo a figura anterior notamos que cada face possui uma adjacente que junto com ela forma um quadrilátero, exemplo: 0 e 1, 2 e 3, 4 e 5, ou seja, em uma lista de faces da classe Plane existe sempre uma face de ordem par e seu adjacente sucessor ímpar formando um quadrilátero. Parece não haver importância, mas esta informação é fundamental para o aprimoramento do procedimento de definição da face. Considerando o padrão de ordenação das faces, número de segmentos e dimensão do objeto tipo *Plane* podemos substituir o algoritmos de busca por uma única operação para encontrar o índice certo da face independentemente desta está no início, meio ou fim da lista. O método *Plane.parserPoin()* implementa esta ideia como visto a seguir:

Código fonte 11 – Determinando o índice da face.

```
/** Cálculo do índice da lista de triângulo onde se encontra
um ponto segundo suas coordenadas
@param x abcissa do ponto
@param y ordenada do ponto
@return int índice da grade cujo ponto está contido
*/
Public int parserPoint(int x,int y){
   //varredura horizontal dos triângulos da grade
   int a =(int)((2* segments)*(int)((x +(width /2))
   /(width/segments)));
   //varredura vertical dos triângulos da grade
   int b =(int)(2*(int)((y +(height /2))
   /(height/segments)));
   /*retorna soma dos valores das varreduras determinando o
   índice do triângulo cuja área contem o ponto indicado*/
   return a + b;
}
```

Fonte: Autor (2012).

Como se pode ver, o método utiliza as coordenadas dos ponto e através das dimensões(altura e largura) e número de segmentos, definidas no construtor, determinando o índice exato. Basicamente a ideia é:

a) subdividir a altura e largura da superfície em trechos. O trecho é o quociente de uma das dimensões pelo número de segmentos. Exemplo: Uma superfície com largura w, altura h e n segmentos terá n trechos horizontais com tamanho $\frac{w}{n}$ e *n* trechos verticais de tamanho $\frac{h}{n}$.

b) encontrar o trecho onde cada coordenada está contida em termos de abcissa e ordenada.

c) indicar quantos índices foram contados até chegar nos trechos determinados respeitando o sentido de crescimento dos índices tanto no horizontal e quanto vertical.

d) efetuar a soma dos valores de cada varredura.

O algoritmo descrito no código fonte 11 implementa a sequência de passos descrito de uma forma simplista, incrementando a performance do cálculo pois reduz a complexidade para O(1).

O problema do mapeamento das coordenadas geográficas para o terreno virtual não está totalmente resolvido. Coordenadas geográficas estão definidas em termos de latitudes e longitude no plano terrestre e variam de -90° a 90° e -180° a 180° respectivamente enquanto que um ponto no plano virtual tem suas coordenadas variando, mas não em graus. Somando-se ao problema da unidade, temos que as duas superfícies não compartilham no seu sistema de referência a origem no mesmo ponto. Utilizando somente a região nordeste como superfície total do mapa e conservando as coordenadas geográficas reais, a região nordeste teria valores negativos para a origem (o novo centro), mas o tereno virtual sempre tem sua origem com valor igual a zero.

O método *parserPoint()* exige que um determinado ponto seja descrito de acordo com as referências do terreno virtual proposto. Para resolver esta situação temos que "formatar" as coordenadas reais para o sistema de referência do terreno virtual. Se os sistemas de referência das superfícies do terreno virtual e região nordeste tivessem ambos a origem zeradas, usaríamos uma regra de três simples. Levando em conta que a latitude no globo terrestre está compreendida, em graus decimais, entre -90 e 90 e longitude entre -180 e 180, temos para latitude:

$$\frac{x}{Latitude} = \frac{\Delta X}{180}$$

Equação 1

e para longitude:

$$\frac{y}{Longitude} = \frac{\Delta Y}{360}$$

Equação 2

Onde ΔX e ΔY são respectivamente a largura e altura do terreno virtual, *x* e *y* o equivalente para latitudes e longitudes no terreno virtual.

Como os sistemas de referencias não possuem a origem em um ponto em comum, então a Equação 1 e Equação 2 não solucionam a conversão. É necessário conhecer dois pontos reais e suas respectivas correlações no terreno virtual com o intuito de "calibrar" a equação e encontrar os valores de um terceiro ponto. Logo, as equações apropriadas passariam a ser:

$$\frac{x - x_1}{x_2 - x_1} = \frac{Latitude - latitude_1}{latitude_2 - latitude_1}$$

Equação 3

e

$$\frac{y - y_1}{y_2 - y_1} = \frac{Longitude - longitude_1}{longitude_2 - longitude_1}$$

Equação 4

onde $latitude_1$ e $latitude_2$ são valores reais de latitudes com correspondência em x_1 ex_2 , respectivamente, e $longitude_1$ e $longitude_2$, longitudes correlacionadas à y_1 e y_2. A tarefa agora é pre-determinar dois valores de referência para latitude e longitude. Por meio de um conjunto de técnicas, em parte de tentativa e erro partimos da ideia que as latitudes -11,451944 e -8,533333, correspondem respectivamente e aproximadamente aos valores -0,091154700538793 e 0,071014083110324 e as longitudes -39,380278 e -37,343611 em 0,150148057706539 e 0,284666163525594. Os métodos *Plane calibraLatitude() e Plane.calibra Longitude()* como descrito na Tabela 1 implementam a equação 3 e 4 por meio dos pontos pré-determinados.

Agora que temos como representar de forma proporcional um ponto real no terreno virtual já podemos nos preocupar com o modo com que os elementos contorno de tensão e linhas de força serão representados neste mesmo terreno.

4.2 Representações da voltagem das subestações

Antes de falar sobre a maneira com que os dados de subestações serão representados, vejamos as informações sobre elas.

Tabela 2 – Subestações.

SIGLA	MUNICÍPIO	UF	TENSÃO	LATITUDE	LONGITUDE
ABX	PAULO AFONSO	BA	230	-9.3958	-38.1942
BJS	BOM JESUS DA LAPA	BA	230	-13.3089	-43.3461
BRA	BARREIRAS	BA	230	-12.1322	-44.9456
ENP	EUNAPOLIS	BA	230	-16.3775	-39.5803
RCD	JABOATÃO	PE	500	-8.1136	-35.0392
ITB	ITABAIANA	SE	230	-10.6603	-37.4583
ZBU	DELMIRO GOUVEIA	AL	138	-9.3558	-38.2028
MSI	RIO LARGO	AL	500	-9.3872	-35.8467
BVT	CAMPINA GRANDE	PB	69	-7.2194	-35.9003
SMD	SANTANA DOS MATOS	RN	138	-5.9547	-36.6575
QXA	QUIXADÁ	CE	500	-4.9525	-38.9278
TSD	TERESINA	PI	500	-5.2644	-42.7267
ELM	ELISEU MARTINS	PI	230	-8.09858	-43.6706
PRI	PIRIPIRI	PI	230	-4.2839	-41.7544

Fonte: CHESF (2010).

Cada subestação tem uma sigla; está situada em um município da região nordeste; possui uma determinada potência e coordenadas geográficas. Para representá-las, criamos, assim como na representação de terreno, duas classes espelhos contendo atributos que armazenam suas informações. Sabemos que cada objeto da classe Estacao tem a tarefa de representar internamente uma subestação de modo que fiquem disponíveis sempre que forem requisitadas. Para garantir a persistência dos dados armazenaremos os objetos utilizando o Db4o como descrito no capítulo anterior.

Código fonte 12 – Armazenando um objeto *Estacao*.

```
/*instancia um ObjectContainer para efetuar as operações com o
db4o*/
ObjectContainer db = server.openClient();
//criando um objeto para a estação ABX em Paulo Afonso-BA
Estacao estacaoABX = new Estacao("ABX","BA","PAULO AFONSO",
                                 -9.3958,-38.1942,230,1000,1000);
//armazenando o objeto estacaoABX na base de dados
db.store(estacaoABX);
```

Fonte: Autor (2012).

Os campos descritos na tabela 1 são informações concretas definidas por números ou palavras, diferentemente de um contorno de tensão, que, assim como um gráfico de pizza ou em barras, busca esta representatividade de forma conceitual, mais abstrata, como um *chart*[7]. O objetivo de um *chart* é a representação de informações a partir de formas, padrões de cores ou tamanho. Definir um contorno significa representar algo segundo seus limites ou fronteiras, destacando seu interior do que está externo a ele.

Agora que falamos de *chart* e de suas propriedades visuais importantes (forma,tamanho e cores), definiremos as características do contorno de tensão a ser utilizado no nosso sistema. Primeiro, se os contornos são de tensão, o principal dado a ser explorado será a tensão. Em segundo lugar, ao contrário da maioria dos *charts* que trabalham com imagens bidimensionais, exploraremos a terceira dimensão. O contorno de tensão terá altura como forma de representação da tensão.

4.2.1 Determinando área de contorno

Não podemos esquecer que a superfície do terreno virtual é formada por milhares de triângulos. Por isso, um contorno, para ser significativo em relação à superfície total do terreno, sugere a seleção de algumas centenas deles e tudo isso para cada subestação. Um algoritmo de seleção para as faces do contorno deve levar em conta seu grande número e buscar ser eficiente.

A computação gráfica sempre procurou utilizar desde seu início os mais eficientes tipos de algoritmos para criar imagens de retas, círculos, elipses, etc.. A

[7] Chart é um estrangeirismo usado como sinônimo da palavra gráfico. Um *chart* é uma representação dos dados, na forma de figuras geométricas, diagramas, desenhos ou imagens que permite uma interpretação rápida e objetiva sobre estes mesmos dados.

ideia consiste em conceber a tela de projeção como uma matriz e cada célula como um *pixel* associado. Desenhar um segmento de reta consiste em definir um padrão de cor para todos os *pixels* por onde ela passa.

Podemos citar os algoritmos de Bresenham e Analisador Diferencial Digital (DDA) como os mais conhecidos e utilizados. Partindo da representação de segmentos é possível desenhar outras estruturas como um quadrilátero e não só o seu contorno, mas também seu preenchimento interno, colorindo-o por meio de uma série de segmentos de retas internos a este mesmo quadrilátero.

Foi inspirado no algoritmo de Bresenham que definiremos as faces presentes em cada contorno. Se graficamente a tela de um dispositivo mapeia sua superfície como uma matriz de *pixels*, o terreno virtual, como objeto da classe *Plane*, pode ser mapeado como uma de triângulos através do método *Plane.parserPoint()* que trabalhar com esta analogia. Se uma matriz identifica uma célula, com uma combinação única de valores de linha e coluna, um triângulo da malha da primitiva *Plane* a faz por meio de índices.

É para determinar o contorno que o método *Plane.parserPoint()* mostra sua utilidade, ele é a base de um procedimento mais complexo, mostrado adiante. A classe *Plane,* no lado servidor, é manipulada diretamente por outra classe chamada *Virtualizer* que se preocupa diretamente em determinar os índices das faces de cada contorno.

Tabela 3 – Métodos da classe Virtualize.

Método	**Descrição**
Virtualizer	Construtor da classe. Recebe um plano e um ponto central para determinação de área.
line	Adaptação do algoritmo de Bresenham para definir faces de um objeto Plane por onde um suposto segmento de reta determinado por dois pontos passaria.
circleFill	Adaptação do algoritmo de varredura de linha para preenchimento de círculo, *Scan-LineCircleFill*, utilizando coordenadas cartesianas.

Fonte: Autor (2012).

Como queremos determinar uma superfície e o algoritmo de Bresenham determina pontos, então basta implementar este mesmo algoritmo trocando o comando que determina a pigmentação do *pixel* pelo método *Plane.parserPoint()*

visto a seguir:

Código fonte 13 – Método *Virtualizer line()*.

```
     /**
     * Adaptação do algoritmo de Bresenham para retornar os índices
     * do plano onde um dado segmento de reta o intercepta.
     * plane Plano em questão
     * x1 abcissa do primeiro ponto do segmento de reta
     * y1 ordenada do primeiro ponto do segmento de reta
     * x2 abcissa do segundo ponto do segmento de reta
     * y2 ordenada do segundo ponto do segmento de reta
     * TreeSet contendo os índices dos triângulos
*/
Public TreeSet line(Plane plane,int x1,int y1,int x2,inty2){
   //cria um objeto para armazenar os índices encontrados
   TreeSetindexTreeSet = new TreeSet();

   int dx = (int)Math.abs(x1 - x2);//delta x
   int dy = (int)Math.abs(y1 - y2);//delta y
   int const1, const2, p, x, y, step;

   if(dx >= dy){
      const1 = 2*dy; const2 = 2*(dy- dx);
      p =2*dy- dx;
      x = Math.min(x1, x2);y =((x1 < x2)?(y1):(y2));
      step =(y1 > y2)?(-1):(1);
   }
   if(plane.parserPoint(x, y)>=0
      )//Armazena o valor  encontrado no objeto do tipo TreeSet
      indexTreeSet.add(plane.parserPoint(x, y));
   }
   while(x <Math.max(x1, x2)){
      if(p <0){+= const1;}else{
         y += step;p += const2;
      }
      if(plane.parserPoint(++x, y)>=0){//Armazena
         indexTreeSet.add(plane.parserPoint(++x, y));
      }
   }
   }else{
      const1 =2* dx; const2 =2*(dx -dy);
      p =2* dx - dy;
      y = Math.min(y1, y2);x =((y1 < y2)?(x1):(x2));
      step = (x1 > x2)?(-1):(1);
     if(plane.parserPoint(x, y)>=0){//Armazena
         indexTreeSet.add(plane.parserPoint(x, y));
     }
   while(y <Math.max(y1, y2)){
     if(p <0) p += const1;}
     else{
       x += step;p += const2;
     }
     if(plane.parserPoint(x,++y)>=0){//Armazena
        indexTreeSet.add(plane.parserPoint(x,++y));
     }
   }
  }
 return indexTreeSet;
}
```

Fonte: Autor (2012).

O método retorna um *TreeSet* de índices de uma fileira a serem usados no contorno, isso porque durante as iterações muitos índices podem sair repetidos e um *TreeSet* garante o armazenamento do índice uma única vez evitando repetições. Definir uma fileira de faces, por si só, não faz sentido para determinar uma área. Citado anteriormente, um polígono com preenchimento interno implica na determinação de uma série de segmentos. Por isso, escolhemos que o contorno a ser preenchido será sempre um círculo. Agora basta utilizar o procedimento de preenchimento de círculo através do algoritmo *Scan-Line Circle Fill*:

Código fonte 14 – Método *Virtualizer.circleFill()*.

```
/**
* Adaptação do algoritmo de varredura de linha para
* preenchimento de círculo,Scan-LineCircleFill,utilizando
* coordenadas cartesianas.
* plane Plano em questão
* xc abcissa do ponto central do círculo
* yc ordenada do ponto central do círculo
* r valor do raio
* ArrayList contendo os índices dos triângulos
*/
Public ArrayList circleFill(Plane plane,int xc,int yc,int r){
   //cria um objeto para armazenar os índices encontrados
   TreeSet indexTreeSet = new TreeSet();
   int x1, x2;
   for(int y = yc - r; y <= yc + r;++y){
      x1=(int)Math.round(xc +Math.sqrt(Math.pow(r,2)
      -Math.pow(y -yc,2)));
      x2=(int)Math.round(xc -Math.sqrt(Math.pow(r,2)
      -Math.pow(y -yc,2)));
     //Armazena o valor encontrado no objeto do tipo TreeSet
     indexTreeSet.addAll(line(plane, x1, y, x2, y));
   }
   /*Cria um objeto do tipo ArrayList a partir objeto do tipo
   TreeSet*/
   ArrayListindexList = newArrayList(indexTreeSet);

   return indexList;
}
```

Fonte: Autor (2012).

O círculo é definido a partir de seu centro até um raio de alcance. O método utiliza ainda uma *TreeSet* com base na eliminação de índices repetidos para depois a converter em um *ArrayList* e por fim retorná-lo.

Toda vez que a visualização de contorno de tensão for requisitada na camada da interface, sua chamada terá como retorno objetos tipo *Estacao* e a lista

de índices das faces de contorno de cada um deles. Para vincular os dois tipos de objetos criamos um terceiro objeto, definido na classe *EstacaoPro*. Este objeto auxiliar recebe como parâmetro de construção um objeto Estacao e a lista de índices de faces pré-processada.

Código fonte 15 – Processando objetos *Estacao*.

```
//cria um objeto da classe Plane
Plane plane = new Plane(100,2000,2000);
//cria um objeto da classe Virtualizer
Virtualizer virtualizer = new Virtualizer();
//instancia query para requisitar os objetos
Query query = db.query();
/*indica que todos os objetos instancia da classe Estacao
devem ser recuperados */
query.constrain(Estacao.class);
/*executa a query e insere os objetos recuperados em um
ObjectSet*/
ObjectSet list = query.execute();
//laço para percorrer todos os itens do ObjectSet
while(list.hasNext()){
//cria um objeto Estacao a partir dos objetos do ObjectSet
Estacao e =(Estacao)list.next();
//cria um objeto EstacaoPro a partir de um objeto Estacao
EstacaoPro estacaoPro = new EstacaoPro(e);
/* mapeia e armazena o valor da longitude segundo o objeto
tipo Plane */
int posX = plane.calibrateLongitude(estacaoPro.getLongitude());
//insere no objeto tipo EstacaoPro
estacaoPro.setX(posX);
/* mapeia e armazena o valor da latitude segundo o objeto
tipo Plane */
int posY = plane.calibrateLatitude(estacaoPro.getLatitude());
//insere no objeto tipo EstacaoPro
estacaoPro.setY(posY);
//recupera o valor referente à tensão do objeto EstacaoPro
int t =(int)estacaoPro.getTensao();
/*utiliza a Classe Virtualizer para calcular a lista
contendo os os índices dos triângulos baseada na grade,
abcissa,ordenada e tensão*/
estacaoPro.setIndexList(virtualizer.circleFill(plane,posX
                        ,posY,(int)Math.pow(t,0.75)));
//armazena o objeto no banco
db.store(estacaoPro);
```

Fonte: Autor (2012).

4.2.2 Determinando a forma de contorno

No escopo deste trabalho, área de contorno e forma do contorno são coisas distintas. Para entender melhor, imaginemos uma ilha no oceano que foi criada a partir de um vulcão cônico que cresceu e ficou com uma parte acima do nível do mar. Uma foto de satélite tirada da ilha mostraria uma porção de terra de área circular. Uma pessoa que está no litoral da ilha olhando para o ponto mais alto vê o vulcão claramente com a forma de um cone porque consideramos a altura também. É a esta diferenciação que nos referimos aqui.

Sabemos da área, agora nos resta a forma. A forma do contorno assumirá uma superfície gaussiana definida por:

$$f(x,y) = Ae^{-\left(\frac{(x-x_0)^2}{\sigma_x^2}+\frac{(y-y_0)^2}{\sigma_y^2}\right)}$$

onde A é a amplitude, x_0 e y_0 definem o centro e σ_x e σ_y são os *spreads*, os raios em cada eixo da base. Como a área de contorno no plano é circular, os raios terão o mesmo valor.

Código fonte 16 – Implementando a função gaussiana.

```
/*x abscissa do ponto
  Y coordenada do ponto
  x0 abscissa do centro da superfície
  y0 ordenada do centro da superfície
  sx sigma x da superfície
  sy sigma y da superfície
*/
Public function gaussiana(x:Number,y:Number,x0:Number
                          ,y0:Number,sx:Number,sy:Number):Number{
   var a:Number = 200;
   var b:Number =-(Math.pow((x-x0),2)/3000
                 + Math.pow((y-y0),2)/3000);

   return a*Math.exp(b);
}
```

Fonte: Autor (2012).

O algoritmo descrito no Código fonte 16 está contido na camada de apresentação assim como tudo que diz respeito a propriedades visuais dos dados. Por fim, somente resta falar do procedimento que exibe o contorno de tensão.

Código fonte 17 – Geração do contorno para representação de cada subestação

```
// trata todos os índices de contorno do objeto
for each (var i:Objectinestacao.indexList){
 //recebe o índice
  varindex:int= i as int;
  // se o índice é válido
  if(i >-1){
     //cria a forma
     eleva(estacao,index);
  }
}
```

Fonte: Autor (2012).

Depois que cada objeto EstacaoPro é obtido, submetemo-o com o índice de face à função *eleva()*. Internamente a função *eleva* faz uso da função *gaussiana* que por sua vez utiliza a localização geográfica como ponto central do contorno e consequentemente da superfície gaussiana. O segundo parâmetro, *index*, é o valor que indicará a face a ser manipulada. Lembrando que cada face é definida por um triangulo e então, logicamente, por três pontos com abscissa (eixo X), afastamento (eixo Y) e cota (eixo Z). Os valores da abscissa e afastamento serão inseridos na função *gaussiana* e o retorno indicará o valor que a cota assumirá.

4.3 Representações das linhas de tensão

A visualização das linhas de tensão em três dimensões resume-se em exibir o traçado de cada uma delas. A seguir estão alguns exemplos de linhas de força situadas na região nordeste do país:

Tabela 4 – Linhas de força.

Fortaleza – Delmiro Gouveia			
Sigla	FTZ-DMG-U1-5	**Tensão**	230
Caminho			
-38.911553102028-5.31059640080247, -38.1261736126154-5.27024735620132, -37.9758915606866-5.21749374830906,-37.7827807487806 -5.19154161023444, -37.3925287161474 -5.13907720418478			
Banabuiú – Fortaleza			
Sigla	BNB-FTZ-U1	**Tensão**	230
Caminho			
-38.5403887823943 -3.8328703307921, -38.5471428958995 -3.85833862227956, -38.5569985943386 -3.89007473805062, -38.5555932856247 -3.89324846325367, -38.5542267296683 -3.89811130622092, -38.5468073555078 -3.924558776811, -38.5487764114533 -3.9317			
Russas II –Mossoró II			
Sigla	RSD-MSD-U1-9	**Tensão**	230
Caminho			
-37.3493976903003 -5.14869464190285, -37.3640334451004 -5.14537822471508, -37.6268994490797 -5.08924985823296, -37.7780115111763 -5.03369398247061, -37.9938462133712 -4.94897126296535			
Banabuiú – Mossoró II			
Sigla	BNB-MSD-U1-9	**Tensão**	230
Caminho			
-38.911553102028 -5.31059640080247, -38.1261736126154 -5.27024735620132, -37.9758915606866 -5.21749374830906, -37.7827807487806 -5.19154161023444, -37.3925287161474 -5.13907720418478			

Fonte: CHESF (2010).

De acordo com a Tabela 4, uma linha de força tem as seguintes informações: um trecho definido por duas localidades, um caminho definido por coordenadas geográficas, uma sigla e uma tensão associada a ela. Um caminho é a sequência de coordenadas geográficas que definem a trajetória da linha de força, partindo de uma localidade até chegar a outra. Esta sequência começa com uma longitude e em seguida uma latitude repetindo o processo sempre nesta ordem. Cada par de longitude e latitude define um ponto deste caminho.

Para o problema da representação interna criaremos classes espelhos com atributos para armazenar as informações que caracterizam uma linha de força e

armazenar seus objetos na base de dados.

Código fonte 18 – Armazenando um objeto linha que representa linha de força.

```
/*instancia um ObjectContainer para efetuar as operações com o
db4o*/
ObjectContainer db = server.openClient();
//criando um objeto para a linha de força Banabuiú-Mossoró II
LinhaForca linhaForca = new LinhaForca("Banabuiú","Mossoró
                                       ,II","BNB-MSD-U1-9"
                                       ,230);
/*inserindo as coordenadas do caminho entre Banabuiú e Mossoró
II*/
linhaForca.setCoordenadas("-38.9115531020280 -5.31059640080247
                          ,-38.1261736126154 -5.27024735620132
                          ,-37.9758915606866 -5.21749374830906
                          ,-37.7827807487806 -5.19154161023444
                          ,-37.3925287161474 -5.13907720418478
                          " );
//armazenando o objeto linhaForca na base de dados
db.store(linhaForca);
```

Fonte: Autor (2012).

Uma vez requisitados os objetos representantes das linhas de força para exibição, será feita a evocação remota dos objetos e renderização do caminho utilizando o PV3D. Da seguinte forma:

Código fonte 19 – Desenhando as linhas de força.

```
//cria uma textura do tipo  LineMaterial
var lm:LineMaterial = new LineMaterial(0xff0000,1);
//cria objeto Lines3D
var lines3D:Lines3D = new Lines3D(lm,"Lines3d#"+linha.sigla);
var i:int = 0;
//percorre a lista de coordenadas do caminho da linha de força
while(i<linha.indexList.length-3){
   //cria um segmento 3D a cada dois pontos
   lines3D.addLine(new Line3D(lines3D,lm,3
                   ,new Vertex3D(linha.indexList[i] as Number
                   ,linha.indexList[i+1] as Number,-10)
                   ,new Vertex3D(linha.indexList[i+2] as Number
                   ,linha.indexList[i+3] as Number,-10)
                   ));
   i= i+2;
}
```

Fonte: Autor (2012).

A classe *Lines3D* é utilizada para representar o traçado virtual das linhas de força passando como parâmetros uma textura e um nome para o objeto. Um *Lines3D* é uma especialização de *DisplayObject3D* e por isso pode ser manipulado

como qualquer outra forma tridimensional no PV3D, a exemplo de uma esfera ou cone.

A particularidade de um *Lines3D* é que este se forma a partir de segmentos de retas definido por pares de pontos no espaço tridimensional da cena. Utilizamos o método *Lines3D.addLine()* da classe para armazenar cada segmento. Um segmentos é definido por dois ponto com três coordenadas, duas coordenas já são conhecidas, X e Y, vindas da coordenadas geográficas mapeadas no terreno virtualdo, o caminho da linha de força. A nova coordenada, no eixo Z, terá valor fixo igual a -10 para todos os pontos de modo a deixar cada segmento 3D com a mesma altura em relação ao terreno virtual.

5 AMBIENTE DE VISUALIZAÇÃO 3D DE CONTORNO DE TENSÃO

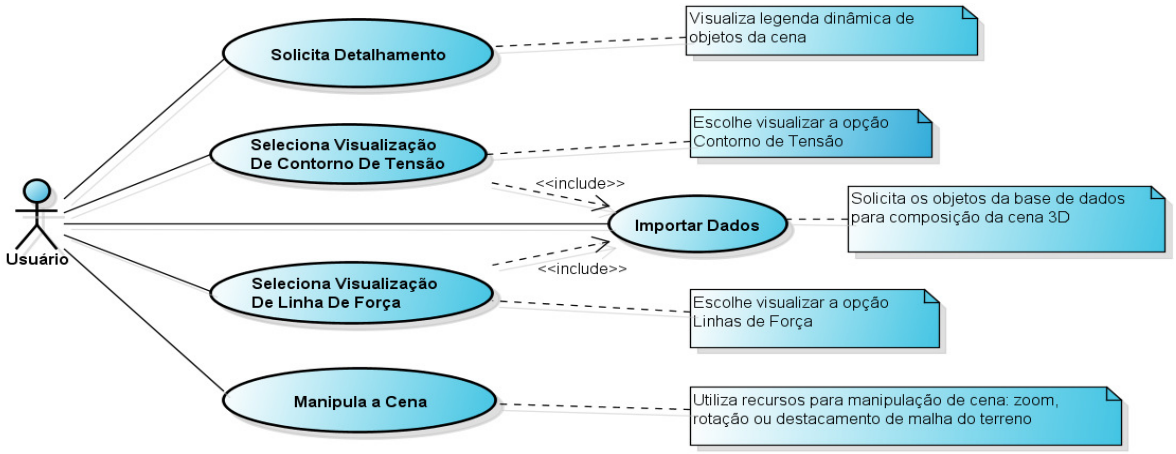
O capítulo anterior destinou-se a descrição das técnicas utilizadas para a proposta de desenvolvimento de algoritmos para visualização tridimensional de contornos de tensão com base na teoria e tecnologias descritas previamente (capítulos 2 e 3). Sabe-se também que tais visualizações serão apresentadas através de um sistema que tem como único objetivo oferecer-lhes suporte. Portanto, resta falar como este sistema viabiliza a exibição de contornos de tensão. Este capítulo destina-se a analisar e descrever o ambiente do sistema de visualização tanto em aspectos estruturais quanto em comportamentais.

5.1 Visão geral do ambiente

O sistema de visualização, como é de conhecimento, insere-se no contexto deste trabalho com finalidade exclusiva de oferecer suporte a execução dos algoritmos relacionadas as técnicas de visualização descritas no capítulo 4. O mapeamento de superfície descrito na seção 1 do quarto capítulo resolve o problema de referência espacial e abre espaço para o uso das outras informações adicionais geo-referenciadas na visualização. O sistema a ser descrito aqui está longe de ser uma rica ferramenta em relação à qualquer outra existente que manipule dados de um sistema de potência, por isso, o mais correto é descrevê-lo como um protótipo, algo previamente desenvolvido com clara motivação de torná-lo melhor. Seguiremos agora com a descrição deste sistema protótipo de maneira mais vertical a partir de agora.

O protótipo oferece duas opções de visualização em 3D, uma para contorno de tensão e outra para linhas de força. O sistema possui um elementos gráficos chamado *TabNavigator* cujo objetivo é servir como contêiner para outros elementos gráficos como botões, áreas de texto, listas entre outros, organizados semelhantemente a um ficheiro. Cada elemento inserido diretamente em um *TabNavigator* origina uma aba. As duas opções de visualização localizam-se cada um em uma aba. Cada aba possui elementos de suporte a cena a ser visualizada e recursos para sua manipulação.

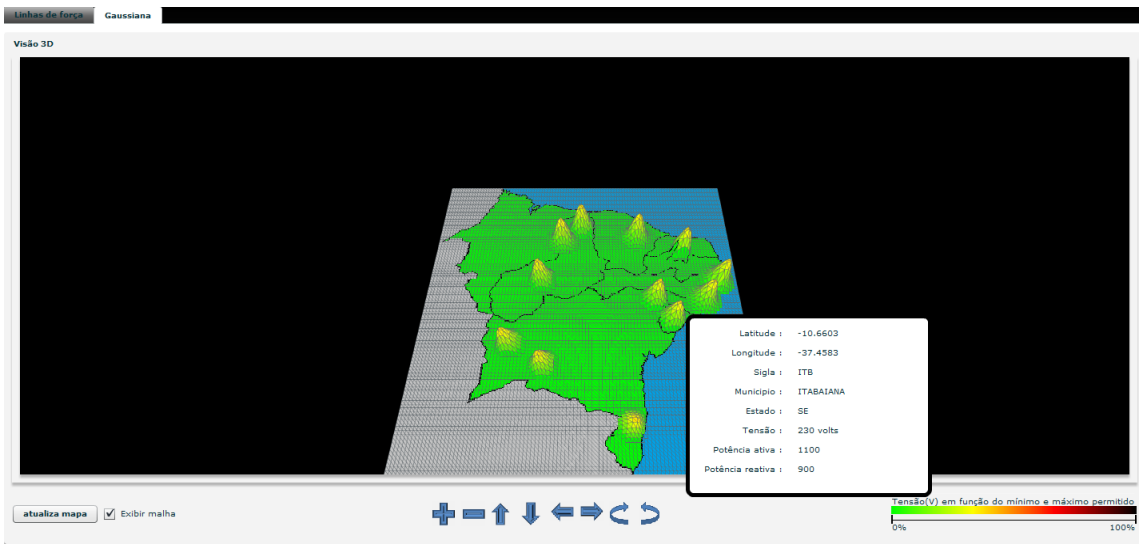
Figura 4 – Diagrama de Caso de Uso para o usuário do sistema.



Fonte: Autor (2012).

A opção de visualização de subestações como o próprio nome sugere, exibe cada uma delas sob a forma de contorno de tensão no terreno virtual que representa a superfície da região nordeste. Como foi dito, a tela de visualização oferece suporte para manipulação de cena, de uma forma mais específica, controle sobre o terreno ou câmera. Incluídos na tela de visualização estão um conjunto de botões que possibilitam rotacionar o terreno e opções de “zoom”. Outras interações possíveis são o acréscimo ou retirada de uma malha sobre a superfície do terreno e ativar a legenda de um contorno de tensão, realizada toda vez que o ponteiro do mouse encontrar-se sobre um deles.

Figura 5 – Tela e visualização de subestações.

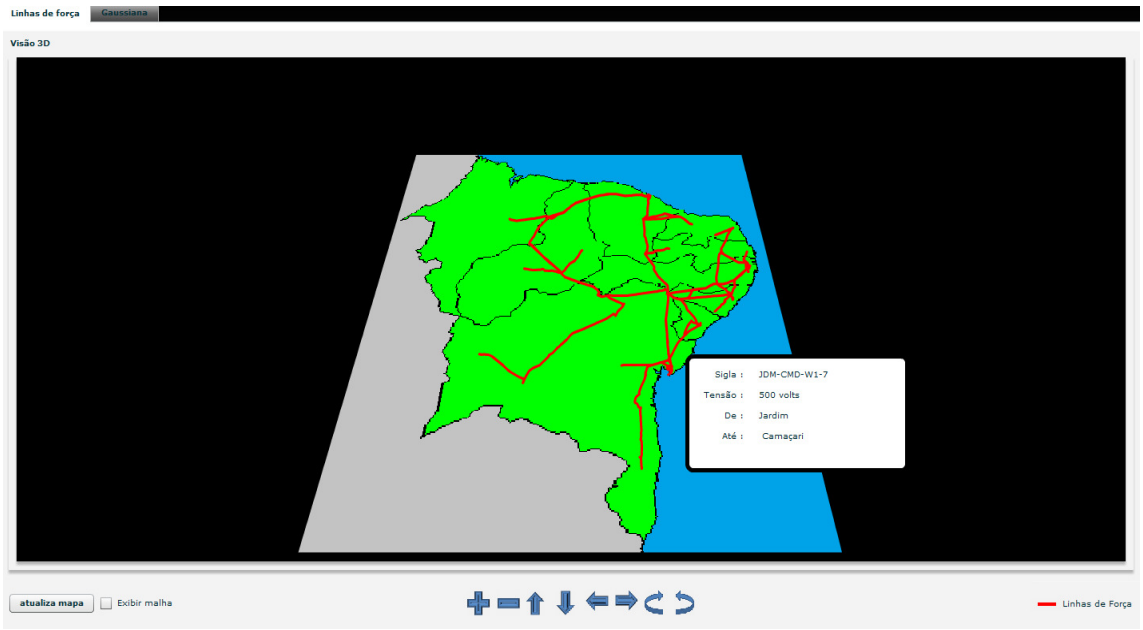


Fonte: Autor (2012).

É possível requisitar explicitamente a importação das subestações através de um botão também presente na tela de visualização. Embora a chamada seja automática ao escolher a visualização, seu uso previne contra uma eventual falha caso a primeira chamada não tenha sucesso ou para uma atualização caso informações de subestações tenham sofrido modificações.

Na outro opção de visualização, a das linhas de força, o processo é semelhante. Carrega-se os componentes da tela e a cena 3D com o terreno virtual e as representações das linhas de força. A ativação da legenda também é realizada toda vez que o ponteiro do mouse estiver sobre um dois trechos.

Figura 6 – Tela de visualização de linhas de força.



Fonte: Autor (2012).

Escolher uma das duas opções de visualização não são as únicas etapas do ciclo de execução do protótipo embora sejam as principais, além disso, existe a possibilidade da cena não ser exibida por completo caso as informações presentes na base de dados não seja importadas (em razão de uma falha). Caso o usuário queira, ao tornar a cena disponível, há a escolha de interagir ou encerrá-la. Da opção da visualização até o encerramento, o protótipo congrega muitos estados de execução podendo retornar e passar por alguns deles inúmeras vezes. Para entender melhor, cada estado será explicado de forma mais detalhada.

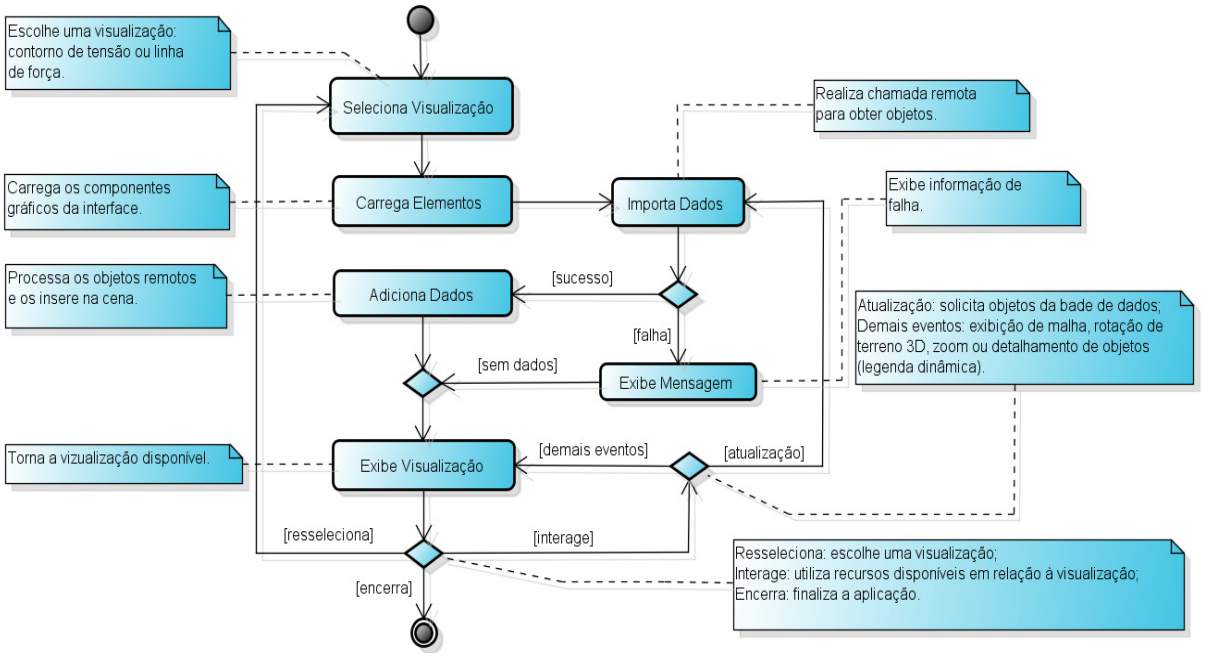
Partindo da seleção de uma das opções de visualização, todo o grupo de

componentes para ela passa a ser carregado, os de controle e o próprio componente de cena. O terreno virtual fica pronto para receber o dados, sejam eles de subestações ou de linhas de força, dependendo da opção.

Em seguida, uma chamada remota será feita para captação de objetos a serem processados e inclusos na cena. A partir daí, existem duas possibilidades: em caso de sucesso, o retorno de uma lista de objetos, senão uma mensagem de erro é exibida. Na primeira, ocorre a alteração da cena com a inserção dos dados. No segundo caso, nenhum processamento é realizado e a cena permanece a mesma. Contudo, o resultado em ambos leva a um mesmo estado, indicando que a cena montada torna-se disponível para visualização.

Em seguida, as escolhas podem ser encerramento, manipulação de cena ou nova seleção de visualização. Supondo agora que o ato de manipular seja o de atualizar dados, então retorna-se para o estado de importação de dados e se realiza outra vez uma chamada remota. Se a ação é de manipular a cena 3D, retrocede-se somente ao estado de exibição.

Figura 7 – Diagrama de Atividades.



Fonte: Autor (2012).

5.2 Arquitetura do ambiente

No processo de desenvolvimento do Sistema de Visualização 3D de Contorno de Tensão, como visto no capítulo 3, foi utilizado o padrão cliente/servidor comum em aplicações *web* e sua arquitetura consiste em seccionar o sistema em três camadas:

a) apresentação: responsável pela apresentação dos dados, visualização dos contornos de tensão, linhas de força e interação com o usuário. Desenvolvido utilizando *Flex* e *Papervision3D*;

b) aplicação: é a camada que contem a lógica operacional do sistema, implementado em código *Java*, realizando o controle de dados e a comunicação entre a camada de dados e a de apresentação;

c) dados: envolve tudo que torna a base de dados funcional para o sistema, garante o armazenamento e a recuperação de informações quando exigida, por isso, trata-se do *Db4o* e suas atividades.

A decomposição em três camadas tem como objetivo desacoplar elementos de acesso a dados, lógica de negócios e tarefas essencialmente de apresentação e interação com usuários. Qualquer alteração realizada em uma camada não afetaria a estrutura das outra. O padrão citado tem uma certa correspondência com o MVC (*Model-View-Controller*) e seu uso permitiu implementar e testa o sistema separadamente.

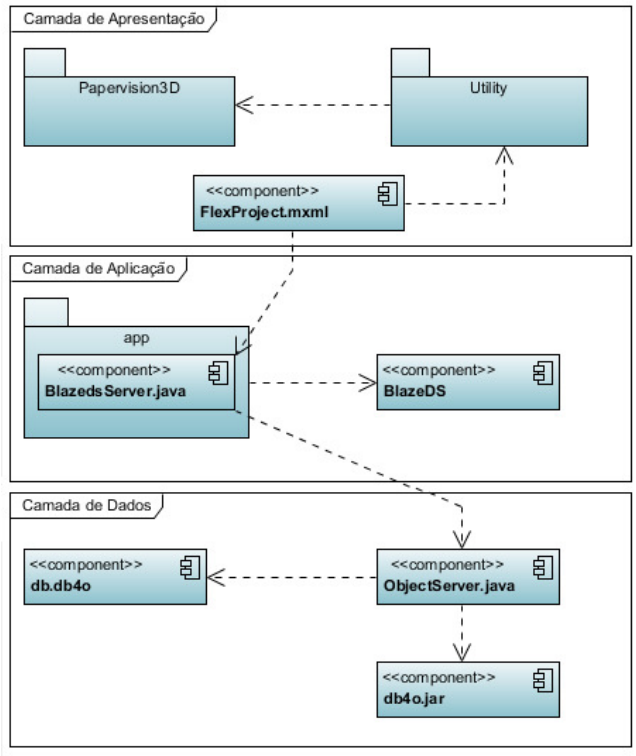
Do ponto de vista estrutural, o formato MVC visualiza um sistema a partir de três elementos, fazendo cada um referência direta a uma camada da arquitetura cliente/servidor. O elemento modelo representa o nível que se preocupa diretamente com o domínio das informações. O *Db4o* necessita de um conjunto de classes reunidas em um arquivo *db4o.jar* adicionado ao projeto e seu objetivo é gerenciar a base de dados (o arquivo *db.db4o*). Para completar, o arquivo *ObjectServer.java* define a fronteira do domínio de dados, seu código determina o único meio de comunicação com este nível.

A responsabilidade de comunicação com o modelo é de exclusividade do

controle, oferece suporte ao domínio da lógica da aplicação e está representado por um conjunto de classes no pacote app descritas na Figura 8. A classe *BlasedsServer* é a única classe de acesso a este nível e sua comunicação com o nível de visão é feita por chamada remota a seus métodos por intermédio do *BlazeDs*, um conjunto de classes em código Java e arquivos de configurações em marcação XML (*eXtensible Markup Language*).

A visão ocupa-se com a apresentação dos dados e todos os recursos disponíveis ao usuário perante o sistema, em outras palavras, os componentes que formam a interface gráfica. As Figuras 5 e 6, citadas na seção 5.1, ilustram este nível. Sua execução inicia-se com a exibição da interface *FlexProject.mxml* que por sua vez utiliza um conjunto de arquivos auxiliares em um pacote *Utility*. Agrupam-se no pacote, códigos de comunicação remota, definição da cena 3D, extensão de componentes gráficos, recursos diversos e a referencia ao *Papervisio3D*.

Figura 8 – Diagrama de Componente.



Fonte: Autor (2012).

Agora que são conhecidos os componentes existentes no sistema, partimos para a compreensão dos meio que os tornam operáveis. Vejamos como cada classe atua na estrutura do sistema.

A interface, definida no *FlexProject.html*, tem como alvo atender as

solicitações do usuário e se comunicar com a classe *BlazeDsService* no lado servidor quando ocorre a necessidade de importação de dados para composição da cena 3D. Os métodos ativados remotamente da classe *BlazeDsService* possíveis são *BlazeDsService.getProEstacao()* e *BlazeDsService.getLinhaForca()* para informações de subestações e linhas de força respectivamente. Temos também as classes discutidas no capítulo 4. São elas:

a) *Estacao*: representa internamente as informações das subestações;

b) *EstacaoPro*: especialização da classe *Estacao* com definição da superfície de contorno, o processamento dos objetos da classe *Estacao*;

c) *LinhaForca*: modelo interno das linhas de força junto ao sistema;

d) *Plane*: relacionada à representação do terreno virtual como referência ao mapa real da superfície geográfica da região nordeste;

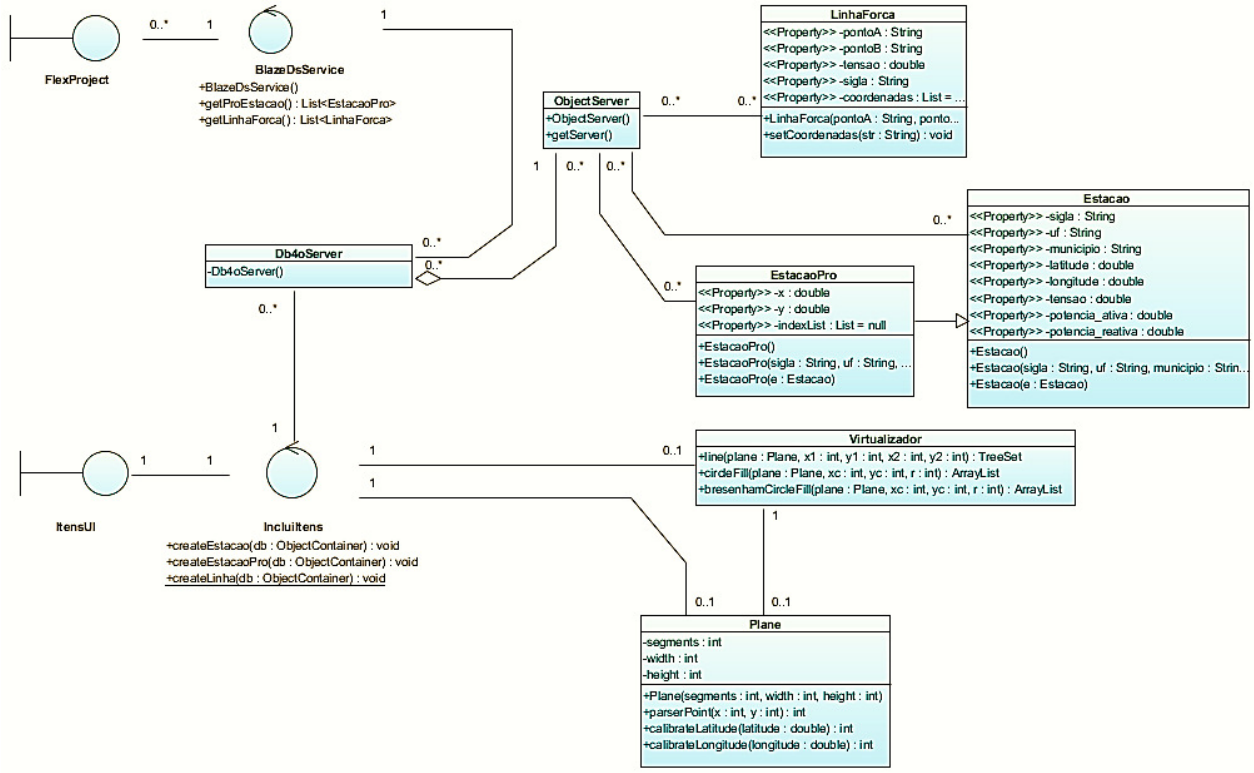
e) *Virtualizador*: Soluciona o problema da determinação da superfície de contorno segundo um objeto da classe *Plane*.

O acesso aos objetos de *EstacaoPro* e *LinhaForca* é feito somente por intermédio de um *ObjectServer* como visto na seção 4.2 e 4.3. Um *ObjectServer* é capaz de atender a inúmeras chamadas a base de dadas, então não há necessidade da criação de mais de um objeto deste tipo durante execução e para garantir isso, o sistema implementa a classe *Db4oServer*. Um *Db4oServer* retorna seu *ObjectServer* pelo método *Db4oServer.getServer()* e caso ainda seja nulo, o instancia primeiro e depois o cede.

Para que um dado esteja disponível para recuperação, deve ter primeiro sido criado. As outras classes restantes dizem respeito ao processo de criação dos objetos que serão adicionados a base de dados. Para inserção dos objetos tem-se duas classes, uma de interface chamada *ItensUI* e outra de controle, a *IncludeItens*. A primeira possui um método para cada tipo de objeto, classe *Estacao*, *EstacaoPro* e *LinhaForca*. A utilização desta classe é pouco sofisticada, inacessível a figura do usuário e isso tudo porque uma vez criado o objeto, a única operação a ser realizada

sobre ele seria sua recuperação, o usuário não cria dados somente os visualiza. *IncludeItens* acrescenta controle a *ItensUI*, manipula as classes *Plane* e *Virtualizador* além do acesso a *Db4oServer*.

Figura 9 – Diagrama de classes em nível de implementação.



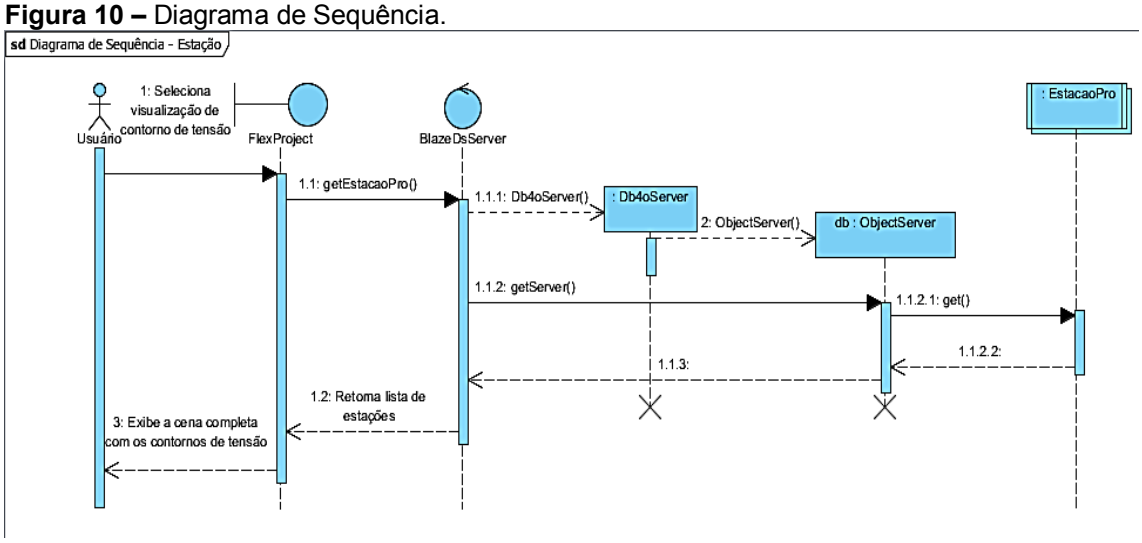
Fonte: Autor (2012).

IncludeItens realiza uma chamada ao método *IncludeItens.createEstacao()* toda vez que um novo objeto representando uma subestação é armazenado a base de dados. O método *IncludeItens.createEstacaoPro()* utiliza o método *Virtualizer.bresenhamCircleFill()* para assim obter o objeto *EstacaoPro* com a região de contorno já definida. O terceiro método, *IncludeItens.createLinha()*, trata da adição dos objetos do tipo *LinhaForca*. Uma vez criado os objetos, este fluxo de execução não se realizará novamente.

Com base nos casos de uso apresentados na Figura 4 e a descrição das classes apresentadas na Figura 9 suponhamos agora que o usuário selecionou a visualização de contorno de tensão. Dando um exemplo prático, veremos a seguir passo a passo como seria a sequência de interação entre os objetos que habilitam o

sistema a atender esta solicitação: Primeiro o usuário escolhe na interface(*FlexProject*) a visualização de contorno de tensão. A interface carrega os componentes visuais criando a cena ainda sem dados. Em seguida, é realizada uma chamada remota ao método *BlazeDsService.getEstacaoPro()* requisitando os objetos correspondentes das subestações. A recuperação dos objetos é realizado por intermédio do *ObjectServer*. Tendo a lista de objetos, a interface insere as informações completando a cena tornando-a disponível ao usuário.

Figura 10 – Diagrama de Sequência.



Fonte: Autor (2012).

5.3 Protótipo

A sessão anterior ocupou-se na descrição do sistema de suporte para a exibição das visualizações. Através dos diagramas de componente e de classe em nível de implementação, contidos nas Figuras 8 e 9 respectivamente, é possível entender os elementos presentes que viabilizam o funcionamento do sistema. Por outro lado, o diagrama de sequência apresentado na Figura 10, foca na mobilização dos elementos, o seu padrão de comportamento.

Quando o usuário opta por um tipo de visualização, um fluxo de comunicação é desencadeado começando na camada de apresentação até a camada de dados. Algumas solicitações secundárias, como a manipulação do terreno virtual, rotação ou translação ocupam somente a camada de apresentação. Independentemente do que o usuário requisita, a interface determina as possibilidades perante o sistema. Esta sessão é destinada ao detalhamento do

sistema sob o ponto de vista da utilização do usuário.

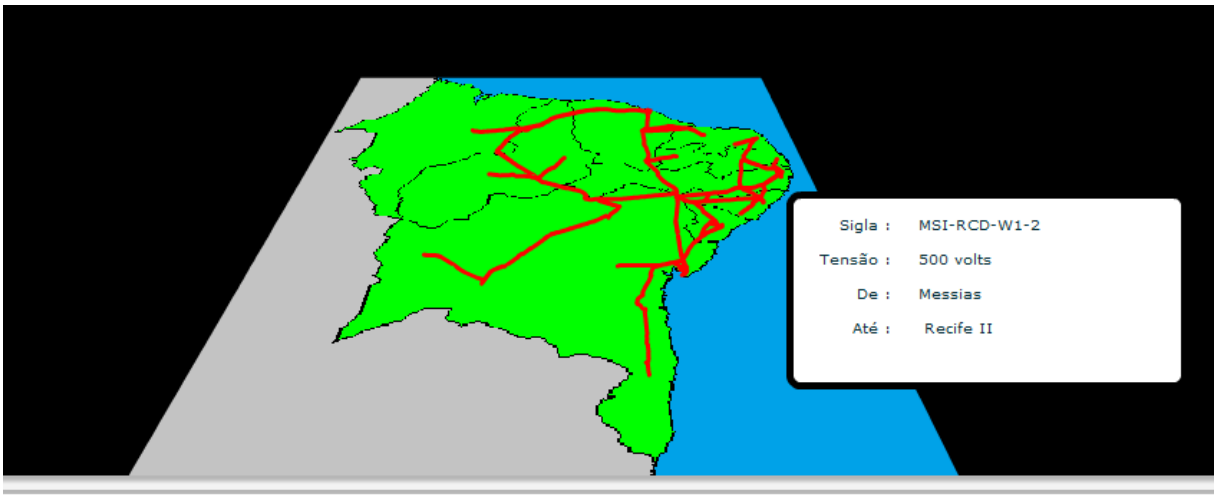
Na primeira sessão deste capítulo foram apresentadas as Figuras 5 e 6 que revelam a aparência da interface. Começando pelo topo da tela, no canto superior esquerdo encontram-se duas abas, uma para agrupar os componentes de exibição e manipulação de linha de força e o outro para agrupar os componentes de contorno de tensão.

Figura 11 – Opções de visualização.

Fonte: Autor (2012).

Cada aba possui o espaço de cena e componentes auxiliares inicializados logo em seguida à escolha do tipo de visualização. Grande parte da área destinada a estes componentes é ocupada pelo espaço de exibição da cena. Quando a cena está completa e os dados sobre linhas de tensão ou subestações já carregados, é possível verificar as informações dos mesmos através de legenda dinâmica ativada toda vez que o mouse encontrar-se sobre algum local do terreno pertencente a um destes elementos.

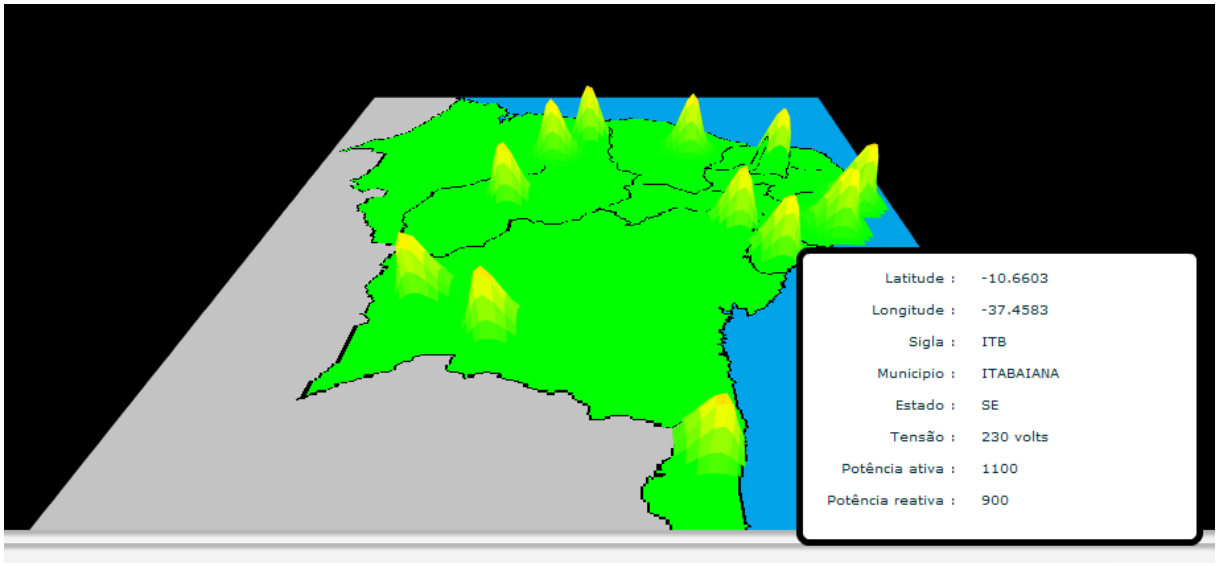
Figura 12 – Exibição de legenda dinâmica de linha de força.



Fonte: Autor (2012).

A Figura 12 demonstra um exemplo de interação com a cena diretamente acionado por um evento do mouse, quando este se localiza sobre uma linha de força. A legenda de uma linha de força exibe uma sigla, o valor da tensão que passa por ela (em volts), sua origem e seu destino. Ainda falando da mesma figura, temos como exemplo que a linha de força destacada pelo mouse tem como sigla MSI-RCD-W1-2, uma tensão equivalente à 500 volts e possui os estremos definidos como Messias e Recife II. Caso o usuário opte pela visualização de contorno de tensão, a legenda criada para este tipo de cena exibe outro agrupamento de informações que dizem respeito às subestações que são a localização geográfica (latitude e longitude), uma sigla, o município onde estas se encontram, a unidade federativa, a tensão de trabalho assim como a potência ativa e reativa.

Figura 13 – Exibição de legenda dinâmica de subestação.



Fonte: Autor (2012).

No exemplo acima da Figura 13, a subestação destacada esta situada no município de Itabaiana, estado de Sergipe, tem latitude em graus decimais de -10,6603 e -37.4583 para longitude, tensão de 230 volts, potência ativa de valor 1100 e 900 para reativa.

Por último, cada tipo de visualização dispõe de um conjunto de componentes de controle e de uma legenda estática explicando – no caso das subestações o gradiente de cores para os limites de tensão – ou traçado das linhas de força – para a outra visualização. Os componentes de controle estão presentes

na parte inferior da tela.

Figura 14 – Componentes de controle da visualização de linhas de força.

Fonte: Autor (2012).

Com base na Figura 14, da esquerda para direita, temos um botão com um texto sobreposto chamado "atualiza mapa". Este botão quando ativado por "click" requisita as informações das linhas de força, isso porque caso a primeira importação seja mal sucedida é possível realizá-la novamente. O próximo componente é um campo de checagem para exibição da malha do terreno virtual caso ela esteja selecionada, o terreno será exibido com o destacamento das arestas de cada face que forma o terreno virtual. Os símbolos de "mais" e "menos" correspondem à opção de zoom da câmera, mais ou menos tendo como foco o terreno virtual. Os outros dois, "seta para cima" e "seta para baixo" alteram o ângulo de visão da câmera em um deslocamento orbital em relação ao terrento, tal ângulo pode sofrer incremento ou decremento. Os símbolos seguintes, "seta para esquerda" e "seta para direita" controlam a rotação do terreno em relação ao eixo Z (altura) também por meio de incremento e decremento de valor. A "seta curva para direita" e "seta curva para esquerda" também realizam rotação, porém em relação ao eixo X. Por último, temos uma legenda explicando que as linhas de força possuem cor vermelha. Todos os elementos de manipulam a cena na barra de controle são ativados por "click".

5.4 Resultados

Ao final do processo de descrição do ambiente de visualização, mostrou-se como as técnicas de visualizações tridimensionais abordadas no capítulo 4 aliadas às tecnologias descritas no capítulo 3 incorporam-se ao sistema. O objetivo era de desenvolver um algoritmo para contorno de tensão, cabendo ao ambiente de visualização a função de servir como um meio para fins de demonstração. Embora os tipos de visualização não sejam novos, sua implementação através deste arranjo de recursos tecnológico são.

O *Papervision3D* mostrou-se um excelente *framework* na composição de

visualizações tridimensionais pois insere no "mundo" *web* a possibilidade de desenvolvimento em um elevado grau de abstração. É necessário ressaltar que o PV3D é limitado em performance ao contrário das aplicações 3D que são executadas nativamente com aceleração por *hardware*, o que já está sendo prometido por outros *frameworks*.

Não é recomendado a utilização do *Papervision3D* quando a lógica do sistema e o processamento de dados também são implementados em *ActionScript* devendo limitar-se ao máximo possível a camada de apresentação de dados.

O *Db4o* apresentou uma proposta interessante como base de dados para objetos, com gerenciamento simples, seu controle é feito diretamente através de programação orientada a objetos, o que deixa a passagem entre a lógica de aplicação e a camada de dados via código bastante sutil sem que haja necessidade explícita de definir arquivos de configuração ou mapeamento entre paradigma relacional e objetos. Por outro lado, esta facilidade de acesso em muitos casos torna-se indesejável para sistemas maiores e complexos que tenham requisitos ocultos como segurança e restrições de acesso sobre a base de dados, todas elas, são políticas já naturalmente presentes em muitos sistemas gerenciadores de base de dados relacionais e isto fica pendente no caso de implementação com o *Db4o*.

Continuando com o processo de análise dos resultados, seis características de um sistema computacional foram utililizadas como parâmetro de avaliação, são elas: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade.

A funcionalidade aqui se refere à capacidade de atendar às necessidades explicitas e implícitas a que se destina o produto. A confiabilidade diz respeito à propriedade de manter o desempenho a longo prazo e verificar se o sistema é imune a falhas. A facilidade de utilização do sistema caracteriza o grau de usabilidade. A eficiência evidencia o desempenho pelos aspectos dos recursos utilizados e tempo de resposta , sob condições estabelecidas. A manutenabilidade significa a avaliação da possibilidade de alteração do sistema. Caso haja existência de transferência para outro ambiente ele é portável.

A seguir, tabela de avaliação para o sistema segundo as características acima citadas:

Tabela 5 – Avaliação de resultados do sistema.

Característica	Subcaracterística	Pergunta	Resposta
Funcionalidade	Adequação	Propõe-se a fazer o que é apropriado?	Sim.
	Acurácia	Faz o que foi proposto de forma correta?	Sim.
	Interoperabilidade	Interage com os sistemas especificados?	Sim.
	Conformidade	Está de acordo com as normas, leis,etc?	Sim.
	Segurança de acesso	Evita acesso não autorizado aos dados?	Não.
Confiabilidade	Maturidade	Com que frequência apresenta falhas?	Algumas vezes em nível de software.
	Tolerância a falhas	Ocorrendo falhas, como ele reage?	Possui rotinas de tratamento somente em nível de software.
	Recuperabilidade	É capaz de recuperar dados em caso de falha?	Sim.
Usabilidade	Intelegibilidade	É fácil entender o conceito e a aplicação?	Sim.
	Apreensibilidade	É fácil aprender a usar?	Sim.
	Operacionalidade	É fácil de operar e controlar?	Sim.
Eficiência	Tempo	Qual é o tempo de resposta, a velocidade de execução?	De moderado a lento para controle em tempo real.
	Recursos	Quanto recurso usa? Durante quanto tempo?	Até 300.000 Kb aproximadamente de maneira constante.
Manutenibilidade	Analisabilidade	É fácil de encontrar uma falha, quando ocorre?	Não avaliado.
	Modificabilidade	É fácil modificar e adaptar?	Não avaliado.
	Estabilidade	Há grande risco quando se faz alterações?	Não avaliado.
	Testabilidade	É fácil testar quando faz alterações?	Em parte. Durante a etapa de desenvolvimento.
Portabilidade	Adaptabilidade	É fácil adaptar a outros ambientes?	Possibilidade não testada.
	Capacidade para ser instalado	É fácil instalar em outros ambientes?	Possibilidade não testada.
	Conformidade	Está de acordo com padrões de portabilidade?	Não analisado.
	Capacidade para substituir	É fácil usar para substituir outro?	Pendente de análise.

Fonte: Autor (2012).

Funcionalmente o sistema realiza o que foi previamente proposto

conforme descrito nas sessões ao longo deste capítulo. Os componentes especificados interagem da forma devida para o correto funcionamento. Não conflituam em nenhum aspecto legal como direitos autorais ou licenças de uso das ferramentas que foram necessárias, além das informações contidas na base de dados serem de domínio público.

Ao final da fase de teste, houve uma redução do número de falhas ficando restrito apenas ao não recebimento remoto de dados. Porém, o sistema se recupera desta falha através de tratamentos de exceções.

Ao visualizar a cena tridimensional, é possível entender de forma rápida o que está sendo representado. Os mecanismos de controle são em número reduzido e são autoexplicativos.

O sistema pode levar até 10 segundos gerando os componentes visuais da tela para cada tipo de visualização escolhida, 4 segundos fazendo chamadas remotas para importação de dados e 2 segundos para responder a uma ação de componentes da barra de controle. Existe um ônus no período de execução em relação à memória, em média com ocupação de 130.000 Kb de forma constante.

Há uma lacuna com relação a manutenibilidade e portabilidade, pois o sistema não foi devidamente testado por usuários aos quais este se destina. Além disso, nunca houve transposição para outro ambiente.

6 CONCLUSÃO

Este trabalho apresentou propostas de técnicas de visualização tridimensionais e suas respectivas implementações para possível utilização em ferramentas que lidam com Sistemas Elétricos de Potência, propondo algumas inovações. Outro ponto a ser destacado é a introdução de tecnologias alternativas na arquitetura de um sistema Cliente/Servidor. As principais contribuições deste trabalho foram:

a) novas técnicas de visualização;
   - contorno de tensão;
   - linhas de força;
b) criação de um sistema de exibição;
c) implementação das técnicas;
d) introdução de tecnologias alternativas;
   - *Papervision3d*;
   - *Db4o*;

Como foi dito na última seção do capítulo anterior não houve originalidade nas visualizações descritas até aqui, mas sim uma opção alternativa de desenvolvimento. Falar de formas de visualização, “o que se faz”, é diferente de discorrer sobre as técnicas de visualização. Toda a cadeia de procedimentos utilizados que possibilitam sua realização, diz respeito ao “como se faz” e isto confere a este trabalho uma configuração original.

Desenvolveu-se um sistema baseado na arquitetura Cliente/Servidor com três camadas para a demonstração provenientes do desenvolvimento das técnicas para fins práticos. Utilizou-se a prototipação no processo da criação do sistema de exibição. O procedimento exigiu o emprego de várias tecnologias assim como foi descrito (seção 3.1) e entre elas algumas alternativas(Db4o e PV3D).

Com relação às técnicas de contorno de tensão e linhas de força que foram concebidas teoricamente e devidamente implementadas, existiu sempre uma preocupação em relação ao processo de tratamento de seus dados. Evitou-se implementar a parte lógica da aplicação em código *ActionScript* e se inseriu também algoritmos utilizados para outros fins, como é o caso do Algoritmo de Bresenham,

para aumentar a eficiência da execução evitando assim perda significativa de performance e consequentemente prejuízo da capacidade do sistema de reagir em tempo real.

Voltando ao pensamento do “como se faz” destaca-se também o uso do *Papervision3D* que comprovadamente é um poderoso recurso de criação para ambientes tridimensionais. Seu manuseio é semelhante a muitos *frameworks* de mesmo propósito como *Irrlicht* e *Ogre3D* para aplicações *Desktop*. O *Db4o* solucionou o problema da persistência de objetos sem que para isso houvesse rompimento do paradigma da programação, o que em outros casos é garantido com recursos auxiliares como, por exemplo, o uso do *Hibernate* simplificando a estrutura do sistema.

Como proposta para trabalhos futuros especificamente direcionados à criação de ambientes tridimensionais para visualização de informações relativas ao sistema elétrico de potência, podemos citar de início, a proposta de aumento do grau de interação através do aperfeiçoamento dos componentes de controle presentes na cena, como a câmera e os objetos tridimensionais nela contidos. Um maior envolvimento no sentido de fazer o usuário adotar uma postura mais ativa em relação ao sistema apresenta-se como uma opção de trabalho ou até mesmo à capacidade de controlar o mundo real a partir do virtual. Não esquecendo que a utilização de novos dispositivos de exibição de imagens podem aprofundar o sentimento de imersão, que no caso do uso de monitores, proporciona-o em baixo nível.

Adicionalmente ressaltamos outras duas alternativas. Na primeira, a possibilidade do desenvolvimento de suporte para novos tipos de dados que desencadeariam novas formas de visualização e na segunda, a opção de centralizar estas mesmas visualizações em uma mesma cena, o usuário escolheria quais tipos exibir.

Por fim, existe a necessidade de realização de testes para avaliação das propostas que foram aqui discutidas e realização de testes em um ambiente de produção com operadores de sistemas elétricos de potência como forma de verificar a usabilidade do ambiente de visualização criado. A busca de soluções que facilitem o entendimento de informações do sistema elétrico de potência refletem diretamente na tomada de decisões por parte do operador e norteiam a pesquisa de novas formas de visualização.

REFERÊNCIAS

AUKSTAKALNIS, S.; BLATNER, D. **Silicon Mirage**: the art and science of virtual reality. Berkeley, CA: Peachpit Press, 1992.

BADAN, T. A. C. **Transformações Geométrica no Plano e no Espaço**. Disponível em: < http://www.eee.ufg.br/~assfalk/disciplinas/cg/plano_espaco.pdf>. Acesso em: 05 março 2012.

CARRETO, C. **Computação Gráfica e Interfaces: Modelação 3D**. . Disponível em: <http://www.ipg.pt/user/~ccarreto/_private/CGI/Docs/0304/Modelação3D.pdf>. Acesso em: 06 março 2012.

CASAS, L. A. **Construção do Conhecimento por Imersão em Ambientes de Realidade Virtual**. In: SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO, 7, 1996, Belo Horizonte: Instituto de Informática, Centro de Informática na Educação, PUCRS. p. 29-43.

COHEN, M., MANSSOUR, I. H. OpenGL: uma abordagem prática e objetiva . Rio de Janeiro: Novatec, 2006.

CROW, M. L.; N. SHETTY. **Electric Power Measurements and Variables Encyclopedia of Energy**, 2004, p. 245-254.

DB4O: banco de objetos de código aberto. [s. d.]. Disponível em:<http://www.db4o.com/portugues/db4o%20Product%20Information%20V5.0(Portuguese).pdf>. Acesso em: 04 mar. 2012.

FOWLER, M. **Padrões de arquitetura de aplicações corporativas**. Porto Alegre: Bookman, 2006.

GOMES JR, D. L. **Visualização de Sistemas de Potência com Dados Georreferenciados**. 2010, p. 17.

GUERRA, G. (s.d.). **DB4Objects na terra de gigantes do BD relacional com Java**. Disponível em DevMedia: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=4121>. Acesso em: 05 maio 2012.

HALDAR; Sibsankar; ARAVIND, Alex. **Distributed Systems**. [s. l.]: Pearson Education India, 2010.

JIDOSTAR. **Papervision3D class diagram**. 2007. Disponível em: <http://blog.jidolstar.com/197>. Acesso em: 07 fev. 2012.

MCCUNE, D.; SUBRAMANIAM. **Adobe Flex 3.0 For Dummies**. [s. l.]: John Wiley &

Sons, 2009.

PARTNERS, C. A. **Db4Objects**. 2012. Disponível em: <http://www.db4o.com/about/customers/>. Acesso em: 04 maio 2012.

PATERSON, J.; EDLICH, S.; HÖRNING, H.; HÖRNING, R. **The Definitive Guide to db4o**. New York, CA: Apress, 2006.

TANENBAUM, A. S., VAN STEEN, M. **Distributed Systems**: Principles and Paradigms. 2. ed. Upper Saddle River, NJ: Prentice Hall, 2006.

WIEGMANN, D. A., J., O. T., HOPPE, S. M., ESSENBERG, G. R., SUN, Y. Human Factors Aspects of Three-Dimensional Visualization of Power. **IEEE Power Engineering Society 18**, 2003, p. 76-82.

WINDER, J.; TONDEUR, P. **Papervision3D Essentials**. Birmingham, UK: Packt Publishing, 2009.