FUNDAÇÃO DE ENSINO EURÍPIDES SOARES DA ROCHA
CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

BRUNO SEIJI MIYAMOTO

DESENVOLVIMENTO DE AMBIENTE DE VISUALIZAÇÃO TRIDIMENSIONAL PARA DADOS DEMOGRÁFICOS

Marília / 2009

BRUNO SEIJI MIYAMOTO

DESENVOLVIMENTO DE AMBIENTE DE VISUALIZAÇÃO TRIDIMENSIONAL PARA DADOS DEMOGRÁFICOS

Monografia apresentada ao Centro Universitário Eurípides de Marília, mantido pela Fundação de Ensino Eurípides Soares da Rocha, como parte dos requisitos para obtenção do Título de Bacharel em Ciência da Computação.

Orientador: Prof. Ms. Leonardo C. Botega.

Marília / 2009

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

Bruno Seiji Miyamoto

DESENVOLVIMENTO DE AMBIENTE DE VISUALIZAÇÃO TRIDIMENSIONAL PARA DADOS DEMOGRÁFICOS

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9.0 (NOVE)

Orientador: Leonardo Castro Botega

1°. Examinador: Ildeberto de Gênova Bugatti

2°. Examinador: Fábio Dacêncio Pereira

Marília, 03 de dezembro de 2009.

"Em geral, a idéia que se tem do 'vazio' é falsa. Quando não se compreende algo, se diz que está 'vazio' de sentido, porém esse não é o verdadeiro 'vazio'. É um engano... Quando teu espírito se encontrar livre da menor nuvem, quando as nuvens da confusão se houverem diluído, aí estará o verdadeiro 'vazio'. "

Miyamoto Musashi.

AGRADECIMENTOS

Primeiramente agradecer a Deus por tudo que conquistei pelo caminho da vida.
Aos meus pais pelo apoio e pela confiança e estarem investindo seus sonhos em mim.
Ao meu orientador, Leonardo C. Botega, pela paciência e sabedoria, sem as quais não chegaria ao meu nível atual de pesquisa acadêmica.
Ao Adriano Bezerra pela modelagem dos objetos 3D.
A todos os meus Familiares, amigos e colegas de classe pelo apoio nesses longos cinco anos de curso.
Aos demais professores, por todo conhecimento passado.
Ao pessoal do Laboratório de Arquitetura de Sistema (LAS) que sempre me ajudou nos momentos de necessidade, e que se tornaram parte importante na minha história.
E finalmente as pessoas que mesmo indiretamente, me influenciaram e me deram forças para continuar seguindo em frente.

MIYAMOTO, Bruno Seiji. Desenvolvimento de Ambiente de Visualização Tridimensional para Dados Demográficos. 2009. 67 F. Graduação em Ciência da Computação – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2009.

RESUMO

Atualmente, dados de demografia e indicadores sociais são disponibilizados para visualização através de relatórios técnicos. Entretanto, a organização e visualização de informações combinadas, na forma impressa ou digital tornam-se algo inviável quando se deseja relacionar informações em uma única estrutura de visualização. Este estudo possui como objetivo o desenvolvimento de um ambiente computacional tridimensional que represente e organize a distribuição de informações combinadas, de forma intuitiva e interativa, facilitando assim a visualização de dados. Ao final do projeto consultas a dados regionais são mais facilmente interpretadas, considerando a apresentação visual intuitiva dos resultados buscados.

Palavras Chaves: Ambientes de Visualização, Computação Gráfica, DemoVis, Dados Demográficos.

MIYAMOTO, Bruno Seiji. Desenvolvimento de Ambiente de Visualização Tridimensional para Dados Demográficos. 2009. 67 F. Graduação em Ciência da Computação – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha, Marília, 2009

ABSTRACT

Currently, demographic data are available for viewing trough reports. However, the organization and display of combined information through printed or digital form becomes impracticable when you wish to relate information into a single structure. This work has as objective the development of a three-dimensional environment that represents and organizes the distribution of combined information in an intuitive and interactive manner, thus facilitating the visualization of data. At the end of the project, regional data queries are easier and well interpreted, considering the intuitive visual presentation of searched results.

Keywords : Visualization Environments, Graphic Computing, DemoVis, Demographic Data.

LISTA DE ILUSTRAÇOES

SUMÁRIO

INTRODUÇÃO

A área de visualização de dados estuda aplicações de técnicas de computação gráfica, que visa auxiliar o processo de análise e compreensão de um conjunto de dados através de representações gráficas que podem ser manipuladas para melhorar a visualização o a compreensão dos dados exibidos (Freitas, 2001)

A visualização de dados através de um ambiente computacional tornou-se abrangente na última década, sendo utilizada principalmente em pesquisa, comércio e educação, auxiliando seus usuários na obtenção de informação de forma rápida, fácil e eficiente (Chen, 1999)

Atualmente a visualização de dados regionais é disponibilizada através de gráficos, tabelas ou desenhos bidimensionais onde a combinação entre dois ou mais dados gera alguma informação relevante ao usuário. Entretanto, a organização e visualização de informações combinadas, na forma impressa ou digital tornam-se algo inviável quando se deseja relacionar informações em uma única estrutura de visualização.

O presente trabalho tem como objetivo o desenvolvimento de um ambiente de visualização tridimensional, que organize e represente dados relativos à demografia de regiões pré-estabelecidas, através de estruturas intuitivas e interativas, auxiliando em pesquisas e coleta de informações, trazendo conhecimento, a respeito de determinadas regiões e seus municípios

O presente trabalho esta organizada em quatro capítulos.

No Capitulo 1 são discutidos os conceitos gerais sobre Ambientes de Visualização, seu histórico, suas áreas de atuação e os tipos de Ambientes de visualização.

No Capitulo 2 são apresentadas as características da API Java 3D, tratando dos conceitos de Grafo de Cena, Geometrias e Manipulação de Objetos 3D.

No Capitulo 3 é discutida a forma como os dados regionais são apresentados aos usuários atualmente e também como os dados da região de Marília foram disponibilizados.

O Capitulo 4 apresenta a metodologia utilizada para o desenvolvimento do sistema de visualização demográfica DemoVis.

No Capitulo 5 serão discutidos os resultados obtidos com a implementação do sistema.

Finalmente no Capitulo 6 são descritas as conclusões do a partir dos resultados obtidos e sugeridos trabalhos futuros que podem dar continuidade ao presente projeto.

E por fim, são apresentadas as referencias bibliográficas que forneceram a base teórica do projeto

CAPITULO I – AMBIENTES DE VISUALIZAÇÃO

A área de visualização de dados estuda aplicações de técnicas de computação gráfica, que visa auxiliar o processo de análise e compreensão de um conjunto de dados através de representações gráficas que podem ser manipuladas para melhorar a visualização e a compreensão dos dados exibidos (Freitas, 2001).

Card *et al*(1999) definem visualização de informação como sendo a representação visual interativa de dados através do uso de um computador para ampliar a percepção de dados abstratos.

Ambientes de visualização de dados podem ser unidimensionais utilizados para visualização de linhas do tempo e documentos de textos, bidimensionais, onde suas representações gráficas estão baseadas em apenas dois eixos, X e Y, normalmente utilizados para visualização de dados geográficos, ou ainda podem ser tridimensionais tendo sua representações gráficas dispostas em três eixo, X, Y e Z, utilizado normalmente para a visualização de dados físicos (Card,1999)

Devido ao crescimento das tecnologias existentes varias formas de se obter e organizar dados, através de meios eletrônicos, se desenvolveram. Dessa forma surgiu a oportunidade de se criar uma forma eficiente de se visualizar dados, algo além de apenas textos e gráficos bidimensionais. É possível criar um ambiente de visualização tridimensional capaz transformar até mesmo dados abstratos em imagens compreensíveis, além de agrupar dados e informações de forma mais prática, e mostrá-las através de uma interface intuitiva e interativa, facilitando a compreensão dos dados e informações dispostas. (Chen, 1999)

A visualização de dados através de um ambiente computacional era utilizado apenas sobre dados científicos, mas devido ao seu crescimento agora vem sendo usado também em campos de pesquisa, comércio, na área de ensino e educação, entre outros.Tornando-se uma área de grande potencial. Auxiliando seus usuários na obtenção de informação de forma rápida, fácil e eficiente (Chen, 1999).

1.1 - Histórico

Os primeiros trabalhos utilizando de gráficos e dados data de meados de 1786, Willian Playfair (1786), inventor da maioria das formas gráficas conhecidas hoje: o gráfico de barras,

o gráfico de linhas baseado em dados econômicos e o gráfico circular eram utilizados para visualizar dados. (Card,1999)

Os métodos de plotagem de dados foram desenvolvidos em 1967, Bertin (1967), um cartógrafo Frances, publicou sua teoria dos gráficos no "*the semiology of graphics*". Esta teoria identificava os elementos básicos de diagrama e descrevia o framework para seu design.

Tukey (1977) começou um movimento juntamente com seu trabalho em análise de dados exploratórios. A ênfase de seu trabalho estava no uso de imagens para dar um rápido entendimento estatístico dos dados.

Tufte(1983) publicou a teoria dos gráficos de dados que enfatiza a maximização da densidade de informações úteis. Tanto as teorias de Bertin quanto as de Tufte se tornaram bastante conhecidas e influentes em varias comunidades que levaram a criação da visualização de informação como disciplina.

Houve um interesse das comunidades de computação gráfica e de inteligência artificial em design automático de apresentação visual de dados. Os esforços foram catalisados pela tese APT de Mackinlay (1986), a qual formalizou a teoria de design de Bertin (1967), adicionou dados psicofísicos, e utilizou-o para gerar apresentações.

Cleveland e McGill(1988) escreveram um livro bastante influente, "*Dynamic graphics for statistics*" (gráficos dinâmicos para estatística), explicando novas visualizações de dados nessa área

A primeira conferência de visualização, IEEE, ocorreu em 1990. Esta comunidade foi liderada por cientistas de recursos terrestres, físicos, e cientistas da computação em super computadores. Satélites estavam enviando de volta grandes quantidades de dados, então a visualização de dados foi um método bastante útil para acelerar a análise e melhorar a identificação de fenômenos interessantes, isso foi também promissor como parte de um esforço para substituir experimentos caros de simulações computacionais (ex., Tuneis de Vento).

Roth e Mattis(1990) estudiosos da área gráfica que deram uma grande contribuição ao desenvolvimento da Visualização de Informações construíram um sistema capaz de fazer visualizações mais complexas.

Feiner e Beshers(1990) apresentaram um método, "*world within world*",para exibição em seis dimensões de dados finaceiros em realidade virtual. O método de coordenadas paralelas de Inselberg (Inselberg e Dimsdale, 1990) e a técnica de ciclismo entre variáveis em

taxas diferentes de Mihalisin (Mihalisin *et al*,1991) foram uma importante contribuição nesse período.

Casner (1991) adicionou a representação de tarefas. A preocupação era a automatização da combinação entre tipos de dados, "*Automating the match between data types*" intenção de comunicação, e representação gráfica de dados.

Finalmente, a comunidade de interface de usuário viu avanços em hardwares gráficos abrindo a possibilidade de uma nova geração de interfaces de usuário. Essas interfaces focavam na interação do usuário com grandes montantes de informação, assim como banco de dados multivariados ou coleções de documentos.

Card, Robertson e Mackinlay(1991) apresentaram formas de utilizar animação e distorção para interagir com grupos grandes de dados em um sistema chamado visualização de informação. O escopo estava no significado para ampliação de reconhecimento. Interatividade e animações eram recursos importantes desses sistemas.

Shneiderman (1992) desenvolveu uma técnica chamada "*dynamic queries*" (consultas dinâmicas) para seleção interativa de subgrupos de dados itens e mapas de arvore.

O grupo de Erick *et al*. (1992) trabalhou com técnicas de grafos estatísticos para grandes escalas de agrupamento de dados associado a um importante problema na rede de telecomunicações e em grandes programas de computador.

A ênfase dos estatísticos era na análise de dados multidimensionais, multivariavel e em novos tipos de dados.

Essas incursões iniciais vieram seguidas por refinamentos e novas visualizações, as diferentes comunidades influenciaram mutuamente umas as outras

Teylingen *et al* (1997) publicaram seu trabalho "*Virtual Data Visualizer*", que apresenta o visualizador de dados virtuais (VDV) que seria um ambiente altamente interativo e imersivo para visualização e análise de dados.

Jern (1998) publicou "*3D Data Visualization on the Web*" que explica as vantagens de se utilizar componentes configuráveis de visualização de dados de baixo custo, que podem ser incorporados e distribuídos em documentos eletrônicos e relatórios.

Kwan-Liu Ma (2000) publicou "*Visualizing Visualization*" um estudo sobre como formas e métodos de visualização de dados e sua interatividade com usuário são capazes de auxiliar em trabalhos de pesquisas.

Mykola Kolodnytsky e Andriy Kovalchuk (2001) publicaram "*Interactive Software Tool for Data Visualisation*" onde é discutido a respeito de interfaces interativas e adaptativas

de visualização de dados científicos chamado "*Graph Server*" que foi concebido e desenvolvido pelos autores.

Freitas *et al* (2001), publicaram "Introdução à Visualização de Informações", o trabalho apresenta uma introdução à visualização de informações, abordando aspectos considerados fundamentais e técnicas que ilustram esses aspectos, assim como comenta importantes características interdisciplinares dessa área.

Benjamin Winchester (2003) fez uso de um Sistema de Informação Geográfica (GIS) para a apresentação de dados demográficos, são apresentados vários mapas de Minnesota (EUA) contendo informações como faixa etária, crescimento populacional, taxa de mortalidade e taxa de desemprego, essas estão dispostas pelo mapa de forma que é possível identificar a situação demográfica de cada região apresentada.

Nathalie Rey Da Silva (2006) publicou "Visualização 3D de dados oceanográficos simulados" neste trabalho, foi realizado um estudo das técnicas e algoritmos de visualização de dados oceânicos. Também foram abordadas diversas características presentes em sistemas de visualização, que serviram como base para especificação das funcionalidades para desenvolvimento de uma ferramenta de visualização de dados oceanográficos simulados.

Godinho, P.I.A. *et al* (2007), publicaram "*PRISMA - A Multidimensional Information Visualization Tool Using Coodinated Views*", o principal objetivo do trabalho foi apresentar a ferramenta de visualização de informação PRISMA, que explora o uso de vários pontos de vista coordenados, provendo a usabilidade, portabilidade e extensibilidade.

Greimstead, I.J. *et al* (2007), publicaram "*3D Anatomical Model Visualization within a Grid–Enabled Environment*", o trabalho apresenta um modelo anatômico de visualização 3D utilizando um ambiente em grade, para treinamento médico.

Dan Zhao e Ning Zhou (2008) publicaram "*Analisys of the Present Aplications of Information Visualizations in E-Commerce Websites*" o trabalho discute a relação entre a visualização de informação e os websites de comercio eletrônico, demonstra vários pontos chaves usados no design visual de um website do gênero, aponta a importância da visualização de informação para tornar informações compreensíveis , e ao final do projeto foi concebido um supermercado virtual tridimensional.

Granda, J.C. *et al* (2008) publicaram "*Design Issues in Remote Visualization of Information in Interactive Multimedia E Learning Systems*", o principal objetivo do trabalho é ir alem do ambiente local, considerando as funcionalidades requeridas em ambientes de

visualizações remotos, em sistemas de multimídia e aprendizagem, e prover aspectos detalhados para seu design e implementação.

1.2 - Áreas de Atuação de Ambientes de Visualização.

Atualmente vários grupos utilizam de sistemas de ambientes de visualização para auxiliar em suas atividades (Azevedo, 2003).

Na medicina é possível observar sistemas tridimensionais que simulam a anatomia humana auxiliando em estudos e treinamentos (Azevedo, 2003).

Na geografia é possível através de um sistema de visualização realizar, por exemplo, estudos a respeito do solo de determinadas áreas ou regiões sem a necessidade de se estar fisicamente no local a ser estudado, a partir do sistema é possível obter todos os dados necessários, e ainda fazer projeções de como estará o local no futuro, sob determinadas condições (Azevedo, 2003).

Na área de jogos virtuais também é possível observar grande uso de ambientes de visualização, por exemplo, no desenvolvimento de cenários e interação com o jogador (Azevedo, 2003).

Na arquitetura é possível criar perspectivas, desenvolver projetos e maquetes virtuais (Azevedo, 2003)

Na psicologia pode ser criado um ambiente para tratamento de fobias e dores, e auxiliar em reabilitações (Azevedo, 2003).

Na área educacional um ambiente de visualização pode auxiliar no aprendizado, desenvolvimento motor, e reabilitação (Azevedo, 2003).

Na segurança pública pode se criar um ambiente para definição de estratégias , treinamento e reconhecimento (Azevedo, 2003).

Em Indústrias pode se utilizar um sistema para realizar treinamentos a baixos custos, controle de qualidade, e desenvolvimento de projetos (Azevedo, 2003).

Na astronomia é possível fazer o tratamento de imagens e modelagem de superfícies de corpos estelares (Azevedo, 2003).

1.3 - Tipos de sistemas de visualização.

Os sistemas de visualização podem ser divididos em três tipos, sistemas de visualização de informação, sistemas de visualização científica e sistemas de visualização de dados. Cada sistema possui distintas características as quais serão apresentadas a seguir. (Friendly, 2008).

1.3.1 - Visualização de informação

Atualmente o termo visualização de informação é geralmente aplicado a representação visual de coleções em larga escala de informações não numéricas, assim como arquivos e linhas de código em sistemas de softwares (Erick, 1994), livrarias e bancos de dados bibliográficos, redes de relacionamentos na internet. (Friendly, 2008). A Figura 1 apresenta uma apresentação gráfica de uma rede de relacionamentos.

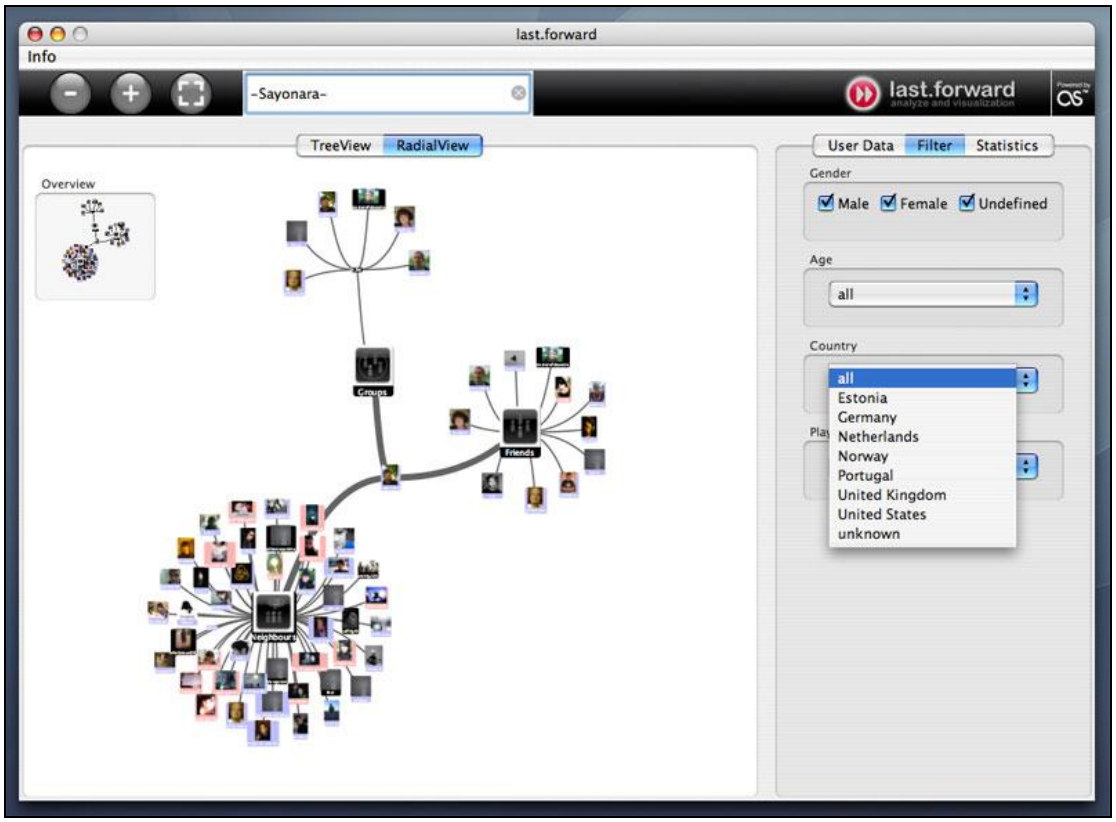

Figura 1 - Representação gráfica de uma rede de relacionamentos (KNOLL, 2007).

1.3.2 - Visualização científica

Outro campo presente é chamado de visualização cientifica. Essa área se preocupa primeiramente com a visualização de fenômenos 3D, para facilitar o entendimento de conjunto de dados de alta complexidade (arquitetônicos, meteorológicos, médicos, biológicos, etc.), onde a ênfase é a renderização realística de objetos, volumes, superfícies, fontes de iluminações, em diante. Talvez com um componente dinâmico (tempo). (Friendly, 2008). A Figura 2 apresenta alguns exemplos de visualização cientifica.

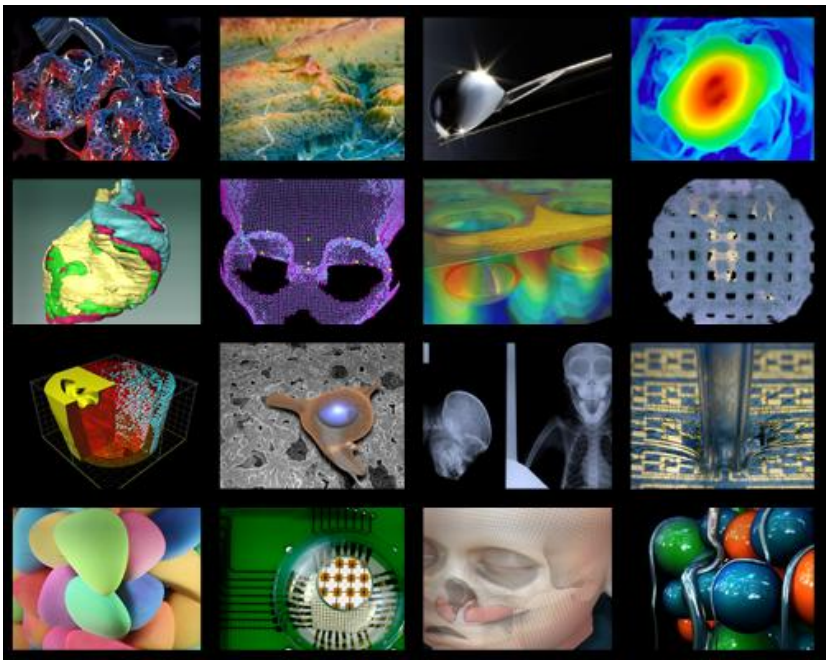

Figura 2 - Exemplos de visualização cientifica (*Beckman Institute for Advanced Science & Technology*, 2008).

1.3.3 - Visualização de dados

Este trabalho se foca em um domínio chamado de visualização de dados, a ciência da representação visual de dados, definida como informações abstraídas de alguma forma esquemática, incluindo atributos ou variáveis para unidades de informações. Este tópico pode assumir dois principais focos: Cartografia temática e Gráficos estatísticos. (Friendly, 2008).

Gráficos cartográficos e estatísticos, ambos se preocupam com a representação visual de dados quantitativos e categóricos, porém com diferentes objetivos. (Friendly, 2008).

1.3.3.1 - Cartografia Temática

Visualização cartográfica se preocupa primeiramente com a representação focada em domínios espaciais. Estes vão desde o simples mapeamento das localidades (massa de terra, rios, terrenos), a distribuição espacial das características geográficas (espécies, a doença, ecossistemas). (Friendly, 2008). A Figura 3 apresenta um Sistema de Informação Geográfica (GIS/SIG).

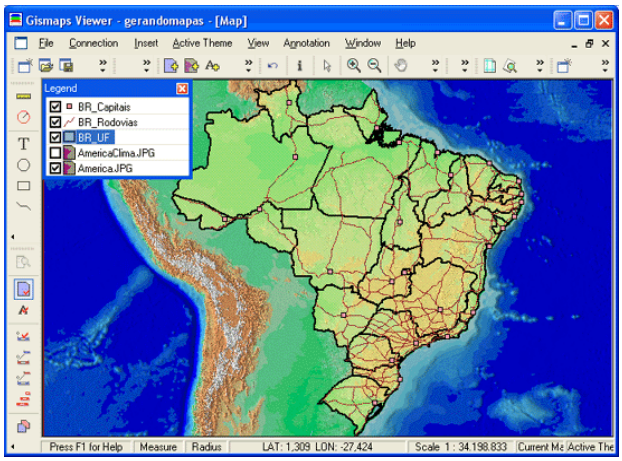

Figura 3 - Exemplo de um sistema de cartografia temática (Gismaps Sistemas Ltda, 2008)

1.3.3.2 - Gráficos Estatísticos

Gráficos estatísticos se aplicam para qualquer domínio no qual métodos gráficos de visualização são aplicados para analises estatísticas usados para descrever padrões, tendências e indicações. Este é o principal foco do trabalho desenvolvido. (Friendly, 2008). A Figura 4 apresenta um exemplo de gráfico estatístico.

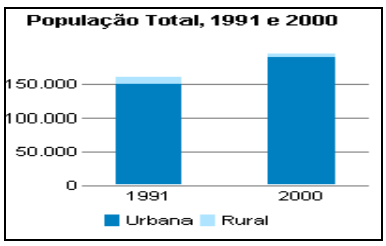

Figura 4 - Exemplo de um gráfico estatístico (CEURB, 2000).

CAPITULO II – JAVA 3D

Neste capitulo são apresentadas as principais técnicas, classes e métodos disponíveis na API (*Application Programming Interface*) Java 3D para a construção de um ambiente de visualização, bem como uma breve descrição das estruturas que compõem uma cena. Posteriormente, são apresentados os passos para construção do sistema, desde a criação do ambiente até a formação das estruturas de visualização de dados demográficos.

2.1 - API Java 3D

Segundo Manssour (2003) a API Java 3D trata-se de um pacote constituído de classes e métodos hierárquicos utilizados na construção de interfaces destinada ao desenvolvimento de sistemas gráficos tridimensionais. Tal conjunto de classes possui construtores de alto nível que viabilizam a criação e manipulação de geometrias e objetos importados, definidos sobre um universo virtual. Possibilitando ainda a criação de ambientes sintéticos com uma grande flexibilidade e portabilidade, onde as cenas são representadas por grafos, seus atributos agrupados e suas informações efetivadas. Desta maneira a implementação de um programa Java 3D se baseia na modelagem ou importação de modelos e na sua atribuição a grafos de cena, os quais são combinados em estruturas hierárquicas. Os grafos de cena são responsáveis pela especificação do conteúdo do universo virtual e pela forma como é visualizado.

A API Java 3D foi desenvolvida pela *Sun Microsystems*, em conjunto com a *Apple*, *Intel* e *Silicon Graphics*, e teve seu código disponível a partir de 1998. Com o intuito de fornecer classes e métodos gráficos em uma plataforma independente, o Java 3D surgiu como uma alternativa de desenvolvimento semelhante ao VRML (*Virtual Reality Modeling Language*), porém otimizada e baseando-se em tecnologias como o OpenGL e o DirectX (MANSSOUR,2003).

2.1.1 - Universos Virtuais

Para a criação e manipulação de geometrias, os programadores fazem uso de construtores de alto nível, pois os detalhes para a geração das imagens são gerenciados

automaticamente. Representações 3D, juntamente com luzes, som e outros elementos integrados, compõe um universo virtual. Neste universo podem existir um ou mais grafos de cena, os quais cuidam da organização dos objetos em uma estrutura do tipo árvore hierárquica (MANSSOUR, 2003).

2.1.2 - Grafos de Cena

As instanciações de classes Java 3D que definem luz, som, orientação espacial, aparência, geometrias e localização, representam um Grafo de Cena, tais instancias são conhecidas na estrutura por nodos (ou vértices). As mesmas apresentam relacionamentos por referencia, o qual simplesmente associa um objeto com o Grafo de Cena; e hierárquico (pai-filho), onde um "nodo do tipo grupo" pode ter filhos os quais contem suas definições e transformações, e apenas um pai. Um "nodo do tipo folha" não pode ter filhos. Em um Grafo de Cena habitual (FIGURA 5), os nodos do tipo grupo são identificados graficamente por círculos, e as folhas por triângulos. Assim, o circulo é considerado a raiz e os demais são acessados hierarquicamente (MANSSOUR, 2003).

Por questão de padronização, cada grafo de cena deve conter apenas um Universo Virtual, objeto do tipo *VirtualUniverse*. Tal objeto configura o universo a ser utilizado e possui ao menos um *Locale*, que demarca um ponto de referencia no universo virtual e funciona como ponto base de todos os sub-grafos do tipo *BranchGroup*. A principal finalidade dos objetos do tipo grupo é associar nodos através de algum atributo comum por meio de um conjunto de características.

De acordo com Manssour (2003), os objetos instanciados como Grafos de Cena regem duas categorias básicas comportamentais as que descrevem o conteúdo do universo virtual, *content branch graphs* ou sub-grafo de conteúdo, as quais tratam de aparência, geometrias, localizações, sons e iluminações, e as que controlam os parâmetros de controle da visualização da cena, *view branch graphs* ou sub-grafo de visualização.

Seguindo esta metodologia observa-se graficamente que todos os nodos do tipo grupo são representados por círculos. Nodos folhas são identificados por triângulos e os demais objetos por retângulos.

Outros nodos de relativa importância e que serão melhores descritos no item seguinte são: *TransformGroup*, usado para especificar posições, orientações e escala de objetos geométricos no universo visual; *Behavior*, *shape* 3D refere-se a dois objetos: *Geometry*, que

trata geometrias , e *Appearence*, que possibilita a utilização de cores , texturas e transparências; e finalmente, *View Platform*, instancia que determina a visão final do usuário no universo sintético(MANSSOUR, 2003).

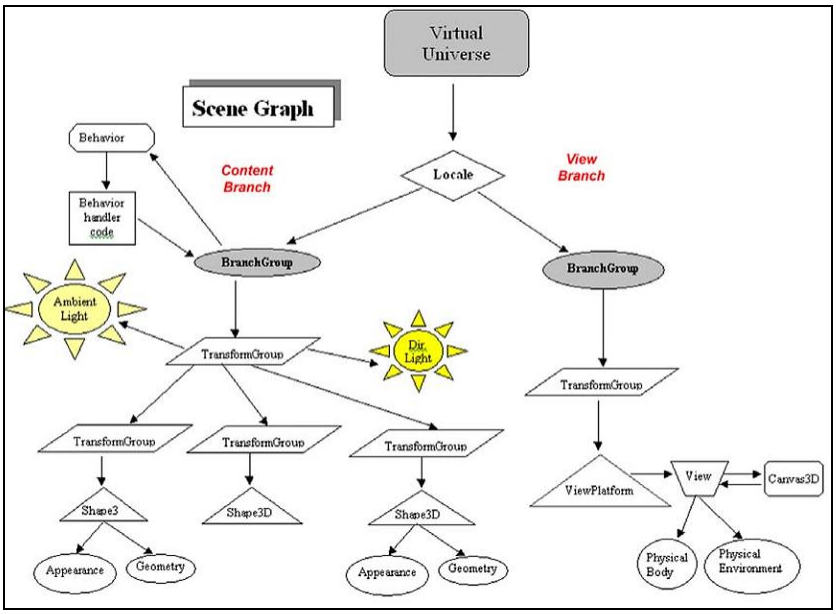

Figura 5 - Representação do Grafo de Cena (MANSSOUR, 2003).

De forma simplificada, os passos para a criação de um programa Java3D são: a criação de um objeto *GraphicConfiguration* e um *Canvas* 3D criado, cria-se um objeto *SimpleUniverse*, que referencia objeto *Canvas* 3D criado, em seguida vem a criação dos objetos *VirtualUniverse* e *Locale*, construindo o subgrafo e visualização. Finalmente insere-se o subgrafo no universo virtual.

2.1.3 - Principais Classes do Java 3D

A API Java 3D possui um grande numero de classes implícitas e pré-programadas para especificar, posicionar e manipular objetos gráficos modelados ou importados. Nesta seção são apresentadas algumas classes fundamentais para o desenvolvimento do trabalho aqui apresentado.

Uma classe de expressiva importância é a *SimpleUniverse*, responsável pela configuração de um ambiente propicio para executar uma aplicação Java 3D, fornecendo todos os recursos necessários para a maioria das transformações. Quando uma instância de *SimpleUniverse* é criada, são criados todos os objetos necessários para o sub-grafo de visualização, tais como *locale* e *viewingPlatform*. Outras classes necessárias são a

GraphicsConfiguration, a qual faz parte do pacote *awt*, responsável pela descrição das características do dispositivo gráfico (impressora ou monitor) e a classe *Canvas 3D*, fornece o *canvas*, ou seja, uma área de desenho ou trabalho, onde é realizado a visualização tridimensional (MANSSOUR, 2003).

A classe *BranchGroup* serve como ponto base de um Grafo de Cena, ou seja, a raiz das transformações. Objetos desta classe são os únicos que podem ser inseridos e associados em um objeto tipo *Locale*, compilados, inseridos em um universo virtual em tempo de execução.

Transformações geométricas de escala, rotação e translação, são instanciadas através da classe de *Transform3D*, formando uma matriz 4x4 de números reais (*float*). Já os objeto da classe *TransformGroup* especificam uma transformação, através de um objeto *Transform3D*, que será herdada a todos os seus filhos. Ao serem aplicadas as transformações, deve-se considerar que os efeitos num grafo de cena são cumulativos (MANSSOUR, 2003). A Figura 6 apresenta a hierarquia das principais classes da API Java 3D.

Já a classe *BoundingSphere* delimita as fronteiras de um objeto sob a forma de uma esfera descrita a partir de um ponto central em um raio. Tal esfera é associada com o limite do objeto sendo usada para aplicar transformações que envolvem as dimensões da geometria.

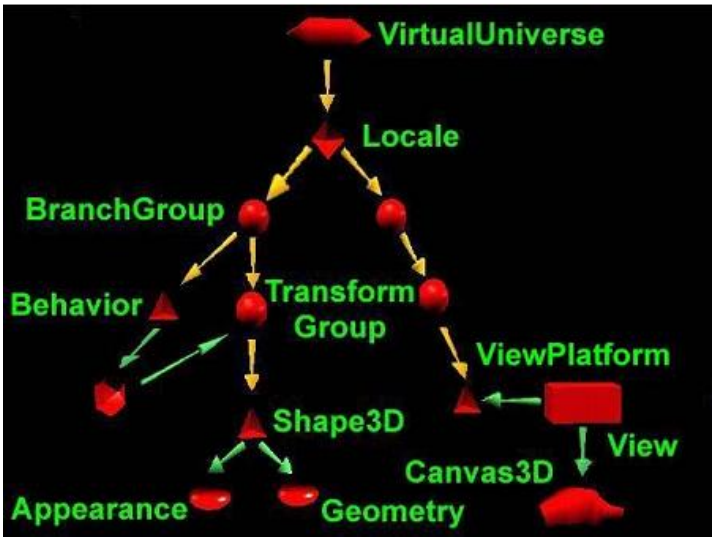

Figura 6 - Representação da hierarquia das principais classes da API Java 3D (BOTEGA, 2005).

2.1.4 - Geometrias

Por padrão, para representar entidades físicas, abstratas ou fenômenos em Computação Gráfica, são utilizados modelos. Desta maneira a modelagem passa a ser uma etapa muito

importante na qual se descreve o modelo de forma que possa ser desenhado. Uma representação de um objeto deve ser feita de forma simples que facilite a usabilidade e analise. Atualmente, existem varias técnicas para a representação de modelos 3D. Nesta seção, são apresentadas formas de representar um modelo em Java 3D, desde a sua geometria, que delimita sua forma física até sua aparência, especificando as propriedades do material que compões a forma geométrica, como cor, transparência e textura (MANSSOUR, 2003).

2.1.5 - Representação de Objetos

Para definir estruturas básicas simples em Java 3D utiliza-se primitivas gráficas como *Box*, *Sphere*, *Cylinder* e *Cone*, disponíveis no pacote *com.sun.j3d.util.geometry*. Sua utilização baseia-se em instâncias de classes com os mesmos nomes das primitivas.

Outra forma de representar objetos graficamente é utilizando-se listas de vértices, arestas ou faces planas, que podem ser triângulos (preferencialmente) ou quadrados. Desde objetos simples até os mais complexos são modelados desta forma. O nodo *Shape3D* é usado para definir um objeto em Java3D. Instancias desta classe referenciam um nodo *Geometry* e um nodo *Appearance*. *Geometry* é uma superclasse abstrata que tem como subclasses, por exemplo, *GeometryArray* e *Text3D*. A classe *GeometryArray* também é uma classe abstrata, e suas subclasses são usadas para especificar pontos, linhas e polígonos preenchidos, tal como um triangulo. Algumas de suas subclasses são: *QuadArray*, *TriangleArray* e *GeometryStripArray*, que, por sua vez, tem *LineStripArray*, *TriangleStripArray* e *TriangleFanArray* como subclasses. Cada uma destas classes possui uma lista de vértices que podem ser conectados de diferentes maneiras. Alem disso, objetos *GeometryAray* também podem armazenar coordenadas do vetor normal, de cores e texturas (MANSSOUR, 2003).

Pode-se também importar dados geométricos com o Java 3D, assim como conjunto de pontos , arestas ou faces, criados por outras aplicações, ou seja, trazer para dentro de um ambiente virtual, modelos desenvolvidos com ferramentas especificas. Neste caso, um arquivo de formato padrão é importado através da classe *Loader* e a cena armazenada é representada em código Java 3D. O pacote *com.sun.j3d.loaders* fornece os subsídios para a implementação. Entre os arquivos que podem ser importados estão 3D Studio (.3ds), *Wavefront* (.obj), VRML (.wrl) e *AutoCad* (.dfx). A Figura 7 apresenta um trecho de código que utiliza um *Loader*. A Figura 8 mostra exemplos de primitivas, lista de vértices e modelos importados.

```
ObjectFile f = new ObjectFile(ObjectFile.RESIZE,
(float)(60.0 * Math.PI / 180.0));
Scene s = null;

try
{
s = f.load( new
java.net.URL(getCodeBase().toString() + "./teapot.obj"));
}
catch (FileNotFoundException e) { … }
catch (ParsingErrorException e) { … }
catch (IncorrectFormatException e) { … }
catch (java.net.MalformedURLException ex) { … }

objRaiz.addChild(s.getSceneGroup());
```

Figura 7 - Representação de um trecho de código que demonstra um *loader* (SUN, 2005).

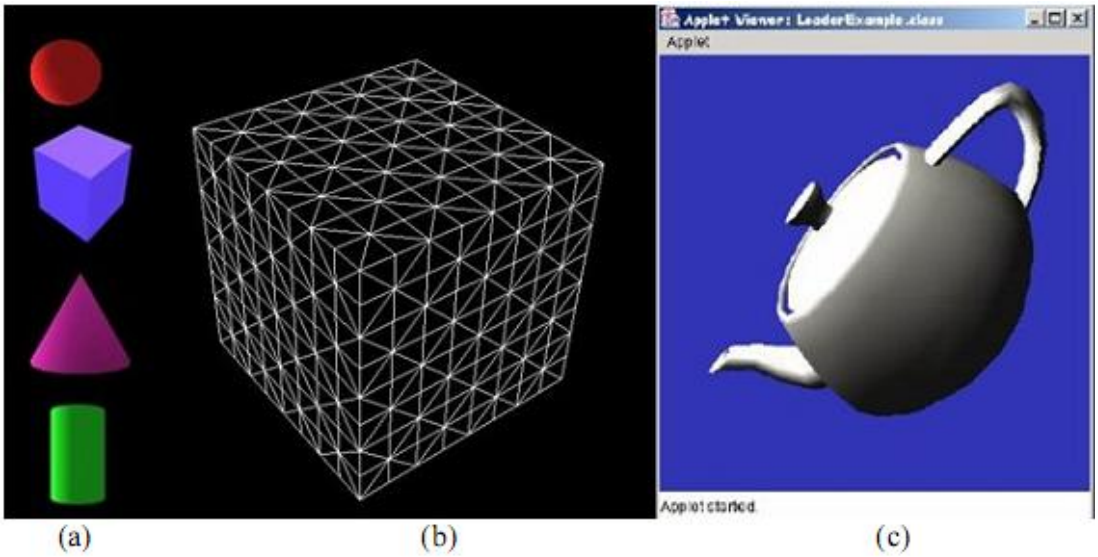

Figura 8 - (a) Representação do uso de primitivas (SUN, 2005), (b) Índices de vértices (PAVARINI et al, 2005) e (c) Modelos importados (MANSSOUR, 2003).

2.1.6 - Aparência

Quando são criadas primitivas gráficas, as mesmas não têm cores especificas. Tal atributo é determinado pela classe *Appearance* e caso a classe não for instanciada, a forma geométrica apresentará a cor branca e outras propriedades como transparência, textura e material não poderão ser manipuladas.

Alguns de seus métodos são listados na Figura 9, mostrando que esta classe faz referencias a vários objetos.

```
void setColoringAttributes (ColoringAttributes coloringAttributes)
void setLineAttributes (LineAttributes lineAttributes)
void setTexture (Texture texture)
void setMaterial (Material material)
Material getMaterial ()
```

Figura 9 - Representação de trecho de código que demonstra a declaração da classe *Appearance* (SUN, 2005).

Já o nodo *Material* é regularmente utilizado para definir a aparência de um objeto levando em consideração as diferentes fontes de iluminação. As vertentes a serem tratadas por esta classe são: cor ambiente refletida da superfície do material, cujo valor padrão é (0.2, 0.2, 0.2); cor difusa, que consiste na cor do material quando iluminado e possui como valor padrão (1.0, 1.0, 1.0); cor especular do material, que tem branco como padrão; cor emissiva, isto é a cor da luz que o material emite (preto por *default*); e *shininess*, que indica a concentração de brilho do material, que varia entre 1 e 128, sendo o padrão 64. A Figura 10 mostra um trecho de código que utiliza métodos da classe *Material*.

```
Appearance app = new Appearance();
Material material = new Material(new Color3f(0.8f,0.8f,0.1f),
new Color3f(0.0f,0.0f,0.0f),
new Color3f(0.8f,0.8f,0.1f),
new Color3f(1.0f,1.0f,1.0f), 100.0f);
app.setMaterial(material);
Cone cone = new Cone(0.4f, 0.8f);
cone.setAppearance(app);
```

Figura 10 - Representação de trecho de código que demonstra o uso de métodos da classe Material (SUN, 2005).

2.1.7 - Iluminação

Para trabalhar com modelos iluminados, deve-se, primeiramente definir qual a fonte de luz que será utilizada no ambiente, ou seja, objeto que emite energia.

Os tipos existentes de fonte de luz são: pontos de luz, luz direcional e *Spot*. Uma fonte de luz por ponto é aquela cujos raios emitem energia uniformemente em todas as direções a partir de um único ponto. Uma fonte de luz direcional é aquela cujos raios provem de uma mesma direção, e *Spot* é uma luz que emite raios de um ponto com uma intensidade variável (SELMAN, 2002).

Posteriormente é necessário definir o modo como a luz interage com os modelos virtuais, em termos de superfície e natureza da luz com o principal objetivo de se obter o efeito tridimensional em espaços bidimensionais, aproximando-se ao Máximo da realidade.

Existem três tipos principais de modelos de reflexão são: ambiente, que é uma luz que vem de todas as direções, resultante da luz refletida no ambiente: difusa, luz que vem de uma direção atinge a superfície e é refletida em todas as direções, fazendo com que o objeto possua o mesmo brilho independente de onde esta sendo visualizado, especular, que vem de uma direção e tende a ser refletida em uma única direção (MANSSOUR, 2003).

Em Java 3D existe a classe abstrata *Light*, que define um conjunto de parâmetro em comum para todos os tipos de fonte de luz. Tais parâmetros são usados para definir cor da luz, se está "ligada" ou não, e qual sua região de influência. As suas subclasses são *AmbientLight*, *DirectionalLight* que por sua vez tem, *SpotLight* como subclasse (SELMAN,2002).

A classe *AmbientLight* trata componentes de reflexão ambiente. Seus construtores permitem ativar ou não sua cor. Já a classe *PointLight* é utilizada para especificar uma fonte de luz em um ponto fixo que espalha raios de luz igualmente em todas as direções. A classe *DirectionalLight* determina uma fonte de luz regular e com destino certo com origem no infinito. Estas fontes de luz contribuem para a reflexão difusa e especular. Nos construtores é possível definir a cor, a posição e o coeficiente de atenuação da fonte de luz (SELMAN, 2003).

Nos construtores da classe *SpotLight* os argumentos são direção, o ângulo de expansão e concentração da luz, sendo que a intensidade da luz é alterada em função do ângulo de expansão. Os trechos de código da Figura 11 demonstram a utilização destas classes para criar diferentes efeitos de iluminação. Em cada método é utilizada uma fonte de luz diferente (direcional, pontual e *spot*) e as reflexões ambiente, difusa e especular são combinadas (MANSSOUR, 2003).

2.1.8 - Textura

Tipos diferentes de material possuem identificação distinta. Desta maneira, características próprias como microestruturas que produzem rugosidade na superfície dos objetos podem ser simuladas com a utilização de imagens digitalizadas especificas.

```
// Exemplo de fonte de luz direcional
 Color3f corLuz = new Color3f(0.9f, 0.9f, 0.9f);
 Vector3f direcaoLuz = new Vector3f(-1.0f, -1.0f, -1.0f);
 Color3f corAmb = new Color3f(0.2f, 0.2f, 0.2f);
 AmbientLight luzAmb = new AmbientLight(corAmb);
 luzAmb.setInfluencingBounds(bounds);
 DirectionalLight luzDir= new DirectionalLight(corLuz,direcaoLuz);
 luzDir.setInfluencingBounds(bounds);
 objRaiz.addChild(luzAmb);
 objRaiz.addChild(luzDir);

 Material material = new Material(new Color3f(0.8f,0.8f,0.1f),
 new Color3f(0.0f,0.0f,0.0f), new Color3f(0.8f,0.8f,0.1f),
 new Color3f(1.0f,1.0f,1.0f), 100.0f);
```

Figura 11 - Representação de trecho de código que ilustra o uso de iluminação (SUN, 2005).

Segundo conceitos de computação gráfica estas, imagens da superfície de um objeto são chamadas de textura. Uma técnica de formação de texturas consiste simplesmente no mapeamento de imagem (mapa de textura/padrão de textura) para a superfície de um objeto (MANSSOUR, 2003). Para aplicar texturas em modelos desenvolvidos através da API Java 3D é necessário criar uma aparência, armazenar a imagem de textura e fazer a associação entre estes objetos. Também é preciso definir o posicionamento da textura na geometria, bem como os seus atributos. Resumindo, os passos para especificação de uma textura em modelagens nativas são: preparar a imagem de textura, carregar a textura, associar a textura com a aparência e determinar as coordenadas de textura da geometria.

É importante salientar que as imagens devem se previamente tratadas com a utilização de um programa externo a API Java 3D e deve estar em um formato compatível com o Java 3D (JPG, GIF, ou PNG). Alem disso, o seu tamanho deve ser múltiplo de dois em cada dimensão. Esta imagem pode estar armazenada em um arquivo local ou em uma URL. Para carregar a textura utiliza-se uma instância da classe *TextureLoader*, que deve ser associada com um objeto *Appearance*. No final, deve-se especificar as coordenadas de textura da geometria. Nesta etapa, as posições da textura em uma geometria são determinadas por coordenadas, definidas por vértices e pontos de textura. Conseqüentemente, uma imagem pode ser esticada, rotacionada ou duplicada.

Já para modelos importados, a aplicação de texturas funciona através de um processo ligeiramente diferente, utilizando-se das classes *TextureLoader* e *Toolkit*, responsáveis por atribuir uma textura a um objeto do tipo *Appearance*, a qual deve ser instanciada no momento em que o modelo importado esta sendo carregado, por meio do método *getImage()*, ou seja, dentro da classe *Loader*, utilizando-se de suas instâncias internas (objetos de aparência).

2.1.9 - Interação

Uma interação ocorre quando o usuário recebe uma resposta, física ou meramente visual, às suas ações sobre o sistema, como pressionar uma tecla ou mover o *mouse*, cujo objetivo é mudar o grafo de cena. Tais reações são possíveis através da subclasse abstrata *Behavior* que, bem como no tratamento de animações, viabiliza alterações no grafo de cena como, por exemplo, a remoção de modelos ou alguns de seus atributos (MANSSOUR, 2003).

A classe *Behavior* é responsável por interpretar as ações do usuário e traduzi-las em alterações para o sistema, ou seja, faz a conexão entre o estimulo e a reação. Existem dentro desta classe, subclasses muito utilizadas para interação, por exemplo, a *MouseBehavior* e a *KeynavigatorBehavior*, que controlam eventos de mouse e teclado, respectivamente (MANSSOUR, 2003).

Outra forma de interagir com os objetos em Java 3D é através dos métodos da classe *OrbitBehavior*. Desta maneira, a *View* do ambiente é "deslocada", ou seja, os modelos permanecem estáticos enquanto é movida apenas a órbita. Fazem parte desta classe ações como rotação, translação e *zoom*. A Figura 12 apresenta um trecho de código que ilustra a utilização de métodos de interação (MANSSOUR, 2003).

```
BranchGroup scene = criaGrafoDeCena();
universe = new SimpleUniverse(canvas);

ViewingPlatform viewingPlatform = universe.getViewingPlatform();
viewingPlatform.setNominalViewingTransform();

// Adiciona "mouse behaviors" à "viewingPlatform"
OrbitBehavior orbit = new OrbitBehavior(canvas,
OrbitBehavior.REVERSE_ALL);
BoundingSphere bounds = new BoundingSphere
(new Point3d(0.0,0.0,0.0), 100.0);
orbit.setSchedulingBounds(bounds);
viewingPlatform.setViewPlatformBehavior(orbit);
universe.addBranchGraph(scene);
```

Figura 12 - Representação de trecho de código que ilustra métodos de interação (SUN, 2005).

CAPITULO III – VISUALIZAÇÃO DE DADOS REGIONAIS.

Atualmente a visualização de dados regionais são disponibilizados através de gráficos, tabelas ou desenhos bidimensionais onde a combinação entre dois ou mais dados gera alguma informação relevante ao usuário (FIGURA 13). Entretanto, a organização e visualização de informações combinadas, na forma impressa ou digital tornam-se algo inviável quando se deseja relacionar informações em uma única estrutura de visualização, pois as informações ficam normalmente espalhadas em varias estruturas diferentes.

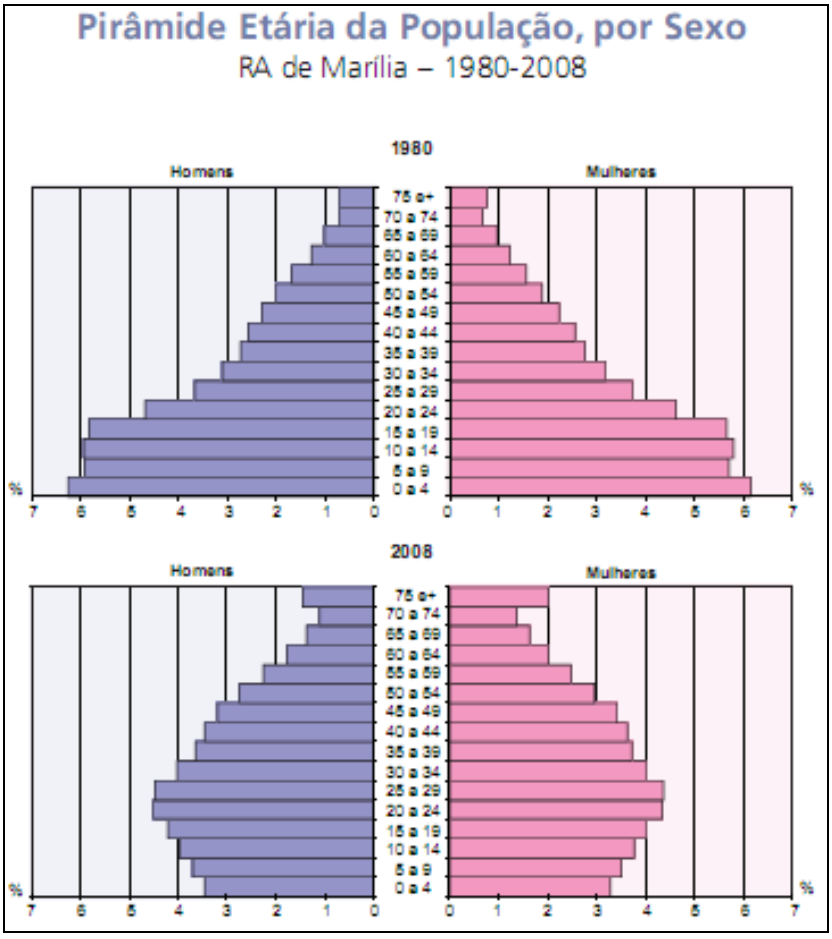

Figura 13 - Gráfico apresentando a faixa etária populacional divida por sexo nos anos de1980 e 2008 (Projeção). (SAEDE, 2007).

Em particular, foram tratados os dados relacionados à região de Marília, por ser sede da instituição de ensino na qual o projeto é desenvolvido.

3.1 - Dados Demográficos da região de Marília

Os dados relacionados à região da cidade de Marília foram disponibilizados para pesquisa pela SAEDE (Fundação Sistema Estadual de Analise de Dados), pela Secretaria de Economia e Planejamento (2007) e pelo Centro Virtual de Estudos Ambientais Urbanos (CEURB, 2000) da Universidade Estadual Paulista "Julio de Mesquita Filho" (UNESP). A região abriga as cidades de Álvaro de Carvalho, Alvinlândia, Echaporã, Fernão, Gália, Garça, Julio Mesquita, Lupércio, Marília, Ocauçu, Oriente, Oscar Bressane, Pompéia, Vera Cruz e seus respectivos distritos (FIGURA 14). (SAEDE, 2007).

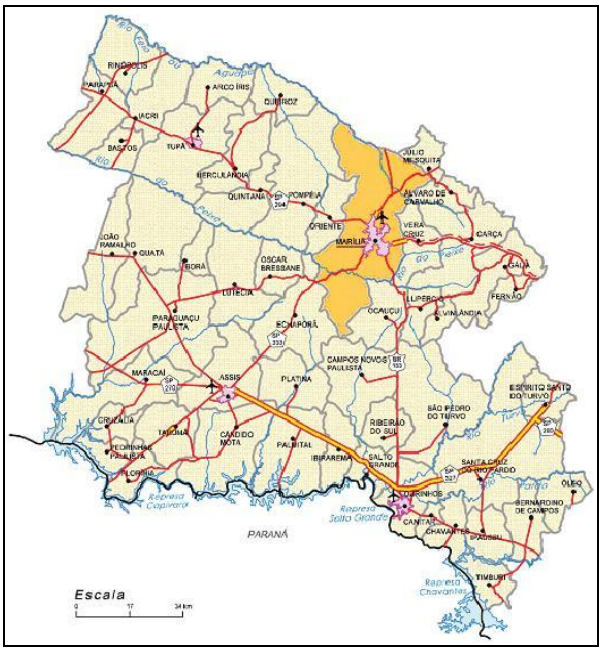

Figura 14 - Mapa com a descrição da Região de Marília (SAEDE, 2007).

Os dados utilizados estão relacionados à demografia regional como, população total das cidades, taxa de população urbana e rural, taxa de analfabetismo e faixa etária. Tais dados foram coletados entre a década de 1980 e o ano de 2007 com projeções ao ano de 2008. (SAEDE, 2007) (CEURB, 2000).

Desta maneira, o objetivo deste trabalho é o desenvolvimento de um ambiente de visualização tridimensional, que organize e represente dados relativos à demografia, através de estruturas intuitivas e interativas, auxiliando em pesquisas e coleta de informações trazendo conhecimento, a respeito de determinadas regiões e seus municípios.

CAPITULO IV – METODOLOGIA

Como descrito anteriormente, o presente trabalho teve como objetivo o desenvolvimento de um ambiente de visualização tridimensional, que organize e represente dados relativos à demografia de regiões pré-estabelecidas, através de estruturas intuitivas e interativas, auxiliando em pesquisas e coleta de informações trazendo conhecimento, a respeito de determinadas regiões e seus municípios. O sistema implementado foi nomeado de DemoVis (*Demographic Visualization*), pois seu propósito está em apresentar visualmente dados demográficos diversos em uma única estrutura, auxiliando desta forma utilização e compreensão de vários gráficos diferentes para cada tipo de dados apresentado.

4.1 - Implementação do Sistema de Visualização Tridimensional para dados Demográficos

Para a implementação deste trabalho foi utilizada a linguagem Java sob a API Java3D. A linguagem foi escolhida devido a sua gratuidade de utilização, replicação de distribuição de seus pacotes, classes e métodos, juntamente com o objetivo de desenvolver uma ferramenta de baixo custo para visualização de dados demográficos. Outro ponto importante na escolha da linguagem foi a flexibilidade e usabilidade fornecida pela mesma ao desenvolvedor.

O Banco de Dados utilizado é o *SQLite*, pela sua gratuidade e fácil manuseio por estar altamente integrado ao sistema desenvolvido, lembrando que o Banco de Dados não é o escopo do projeto, porém, essencial no seu desenvolvimento.

Anterior a fase de implementação foi realizado um profundo estudo sobre da API apresentada, levantando as principais classes e métodos que seriam uteis ao desenvolvimento do projeto. Paralelamente a estas atividades, foram obtidos todos os objetos modelados, ou, seja, a representação de uma árvore com seu tronco e galhos e de uma fruta, em formato "AutoDesk WaveFront" (AutoDesk, 2009) construídas através da ferramenta "3D Studio Max" (AutoDesk, 2009) e exportadas com a extensão OBJ.

Para o desenvolvimento do tabalho foi traçado um diagrama de visão geral (FIGURA 15) para facilitar o entendimento das funcionalidades do sistema e seu processo e desenvolvimento.

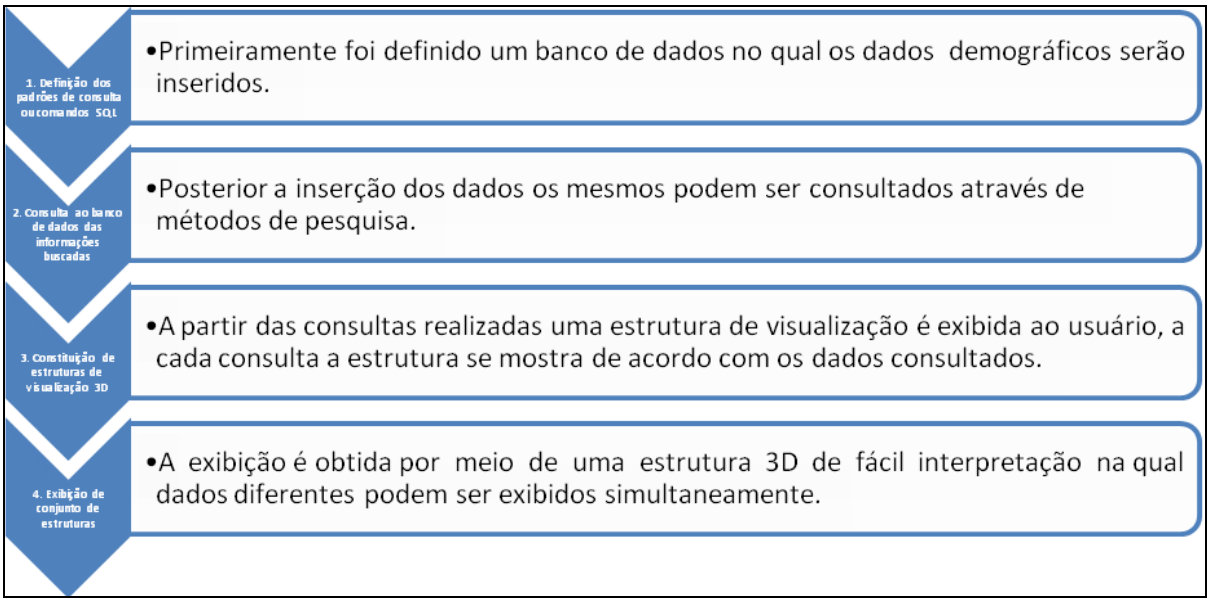

Figura 15 - Diagrama de Visão Geral do Projeto.

O projeto foi dividido em módulos, os quais foram representados por classes independentes criando uma hierarquia a partir de uma classe principal com o objetivo de facilitar a manipulação e o entendimento do sistema. As classes do sistema são apresentadas no diagrama da Figura 16, e descritas nos itens que seguem.

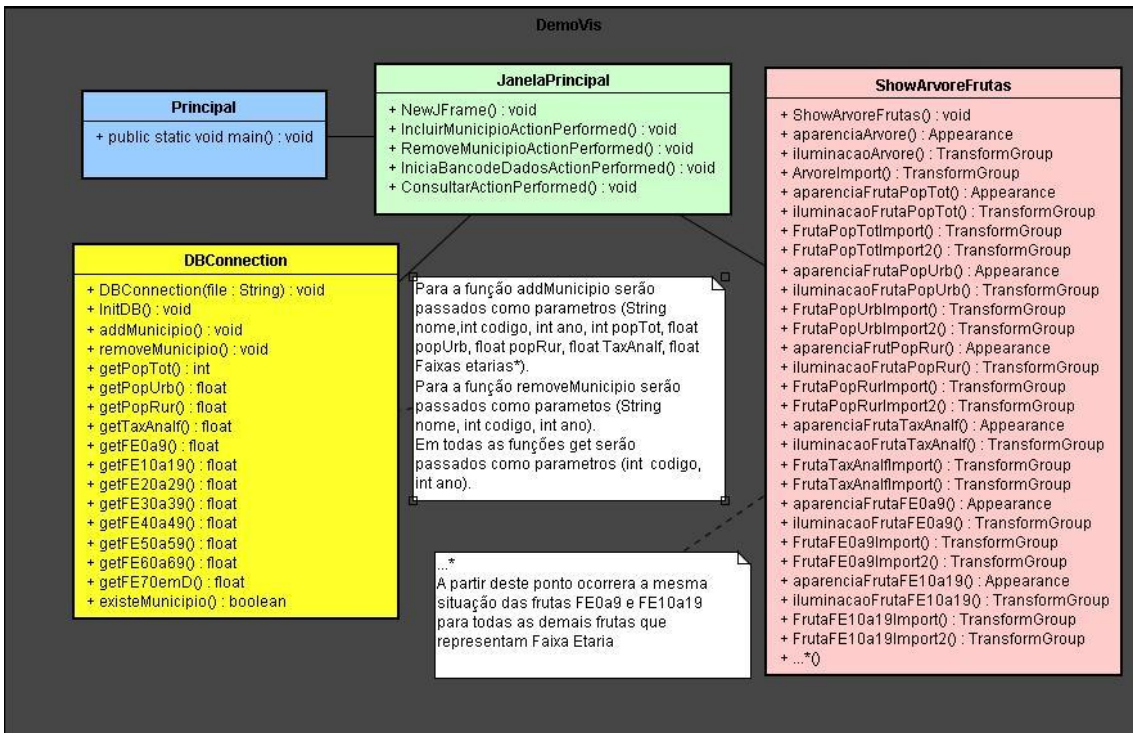

Figura 16 - Representação do diagrama de classes do DemoVis.

4.1.1 - Descrição da Classe “Principal”

A classe “Principal” simplesmente realiza uma chamada instanciando a classe “JanelaPrincipal”e seta as propriedades, de tamanho, localização na tela e visibilidade, da janela principal do programa. Assim como descrito na Figura 17.

```
import javax.swing.JFrame;
/**
 *
 * @author SEIJI
 */
public class Principal {
    public static void main(String[] args) {
     JanelaPrincipal janela = new JanelaPrincipal();
     janela.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     janela.setSize(700,800);
     janela.setLocation(0,0);
     janela.setVisible(true);
    }
}
```

Figura 17 - Trecho de código que representa a classe “Principal”.

4.1.2 - Descrição da Classe “Janela Principal”

A classe “Janela Principal” representa a janela principal do programa contendo o layout e design do projeto além de fazer chamadas às classes “*DBConnection*” e “*ShowArvoreFrutas*”. Através da Classe “Janela Principal” serão realizadas as atividades de inclusão, remoção, reinicio e consulta ao banco de dados.

4.1.2.1 - Inclusão de Dados

Para a realização da inclusão de dados no banco de dados, é necessário que os campos obrigatórios: “Nome do Município”, “Código do Município”, e “Ano de Coleta dos Dados”, seja preenchido. Posteriormente os dados demográficos referentes ao município deveram ser preenchidos em seus respectivos campos, porém não são obrigatórios, ou seja, pode-se criar um município cujos dados demográficos sejam todos de valor zero (Valor Padrão). Em seguida ao clicar no botão “INCLUIR MUNICÍPIO” uma mensagem de confirmação do ato é

exibida, caso o município já esteja presente no Banco de Dados, uma mensagem de erro indicando que existem dados conflitantes é exibida. A Figura 18 demonstra os campos que devem ser preenchidos.

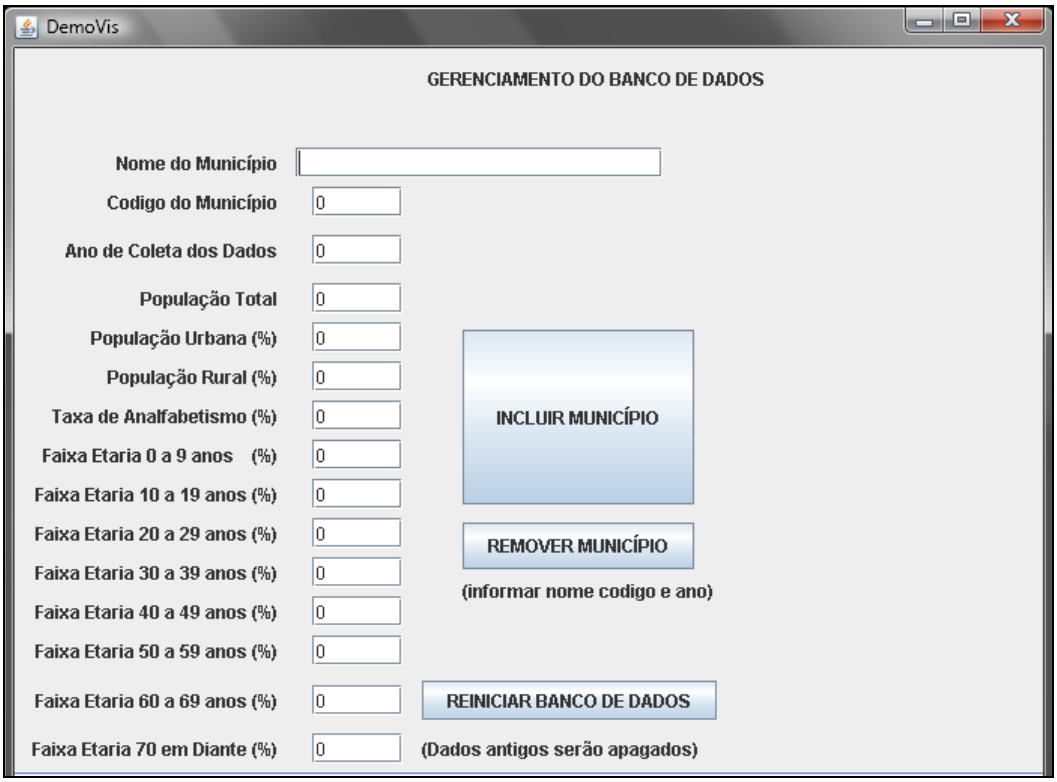

Figura 18 - Imagem apresentando os campos de preenchimento necessários para a inclusão e remoção de dados no banco de dados.

Os dados fornecidos servirão de parâmetro para o método "*addMunicipio*" da classe "*DBConnection*". A Figura 19 apresenta o trecho de código utilizado para a chamada ao método "*addMunicipio*".

4.1.2.2 - Remoção de Dados

Para a remoção dos dados referentes a algum município é necessário o preenchimento dos campos "Nome do Município", "Código do Município", e "Ano de Coleta dos Dados". Em seguida ao clicar no botão "REMOVER MUNICÍPIO" uma mensagem de confirmação é exibida, caso o município não esteja no Banco de Dados uma mensagem relatando a inexistência de Dados é exibida.

A remoção dos dados ocorre por meio da chamada do método “removeMunicipio” da classe “*DBConnetcion*”. A Figura 20 apresenta o trecho de código utilizado para a chamada do método “removeMunicipio”.

```
   private void IncluirMunicipioActionPerformed(java.awt.event.ActionEvent evt) {
       // TODO add your handling code here:
try {
     DBConnection connection = new DBConnection("file");
     if(connection.existeMunicipio(jTextField_inc_NomeMun.getText(),
                      Integer.parseInt(jTextField_inc_CodMun.getText()),
                      Integer.parseInt(jTextField_inc_Ano.getText()))== false)
       {
         connection.addMunicipio(jTextField_inc_NomeMun.getText(), Integer.parseInt(jTextField_inc_CodMun.getText()),
         Integer.parseInt(jTextField_inc_Ano.getText()), Integer.parseInt(jTextField_inc_PopTotal.getText()),
         Float.parseFloat(jTextField_inc_PopUrb.getText()), Float.parseFloat(jTextField_inc_PopRur.getText()),
         Float.parseFloat(jTextField_inc_TaxAnalf.getText()), Float.parseFloat(jTextField_inc_FE0a9.getText()),
         Float.parseFloat(jTextField_inc_FE10a19.getText()), Float.parseFloat(jTextField_inc_FE20a29.getText()),
         Float.parseFloat(jTextField_inc_FE30a39.getText()), Float.parseFloat(jTextField_inc_FE40a49.getText()),
         Float.parseFloat(jTextField_inc_FE50a59.getText()), Float.parseFloat(jTextField_inc_FE60a69.getText()),
         Float.parseFloat(jTextField_inc_FE70Diante.getText()));

         JOptionPane.showMessageDialog(rootPane, "Dados Incluidos","Menssagem do Sistema",JOptionPane.INFORMATION_MESSAGE);
       }else
        JOptionPane.showMessageDialog(rootPane, "Dados Conflitantes","Menssagem do Sistema",JOptionPane.WARNING_MESSAGE);
      } catch (java.sql.SQLException e) {}
      catch (ClassNotFoundException e1) {}


}
```

Figura 19 - representação do trecho de código que realiza a chamada ao método “addMunicipio”.

```
   private void RemoveMunicipioActionPerformed(java.awt.event.ActionEvent evt) {
       // TODO add your handling code here:
       try
       {
       DBConnection connection = new DBConnection("file");
        if(connection.existeMunicipio(jTextField_inc_NomeMun.getText(),
                      Integer.parseInt(jTextField_inc_CodMun.getText()),
                      Integer.parseInt(jTextField_inc_Ano.getText()))== true)
        {
       connection.removeMunicipio(jTextField_inc_NomeMun.getText(),
                                  Integer.parseInt(jTextField_inc_CodMun.getText()),
                                  Integer.parseInt(jTextField_inc_Ano.getText()));
       JOptionPane.showMessageDialog(rootPane, "Dados Excluidos","Menssagem do Sistema",JOptionPane.INFORMATION_MESSAGE);
       }else
       JOptionPane.showMessageDialog(rootPane, "Dados Inexistente no Banco de Dados","Menssagem do Sistema",JOptionPane.WARNING_MESSAGE);
       }
       catch (java.sql.SQLException e){}
       catch ( ClassNotFoundException e1){}

}
```

Figura 20 - representação do trecho de código que realiza a chamada ao método “removeMunicipio”.

4.1.2.3 - Reinicio do Banco de Dados

Esta funcionalidade do sistema realiza a criação das tabelas do banco de dados, caso haja algum dado remanescente de alguma execução anterior os mesmos serão apagados e perdidos, e uma nova tabela vazia será criada. A funcionalidade pode ser ativada através do botão "REINICIAR BANCO DE DADOS", em seguida é exibida uma mensagem de confirmação.

O reinicio do Banco de Dados ocorre através da chamada do método "*initDB*" da classe "*DBConnetion*". A Figura 21 apresenta o trecho de código que realiza a chamada ao método "*initDB*".

```
    private void IniciaBancodeDadosActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        try {
            DBConnection conection = new DBConnection("file");
            conection.initDB();
        } catch (java.sql.SQLException e) {
        } catch (ClassNotFoundException e1) {}

JOptionPane.showMessageDialog(rootPane,"Banco de Dados Resetado","Menssagem do Sistema",JOptionPane.INFORMATION_MESSAGE);

}
```

Figura 21 - Representação do trecho de código que realiza a chamada ao método "*initDB*".

4.1.2.4 - Consulta ao Banco de Dados

Para realizar a atividade de consulta será necessário o preenchimento dos campos "Nome do Município", "Código do Município" e "Ano de Coleta dos Dados" na área de "CONSULTA AO BANCO DE DADOS". Posteriormente devem-se escolher os dados demográficos a serem consultados através da seleção de suas respectivas Caixas de Seleção (*CheckBox*). A Figura 22 representa a área de "CONSULTA AO BANCO DE DADOS".

Para realizar a consulta os dados obtidos através dos campos "Nome do Município", "Código do Município" e "Ano de Coleta de Dados", serão passados como parâmetros para o método "existeMunicipio" da classe "*DBConnection*" que verifica a existência dos dados do município solicitado no banco de dados. A Figura 23 apresenta o trecho de código no qual é feita a chamada ao método descrito.

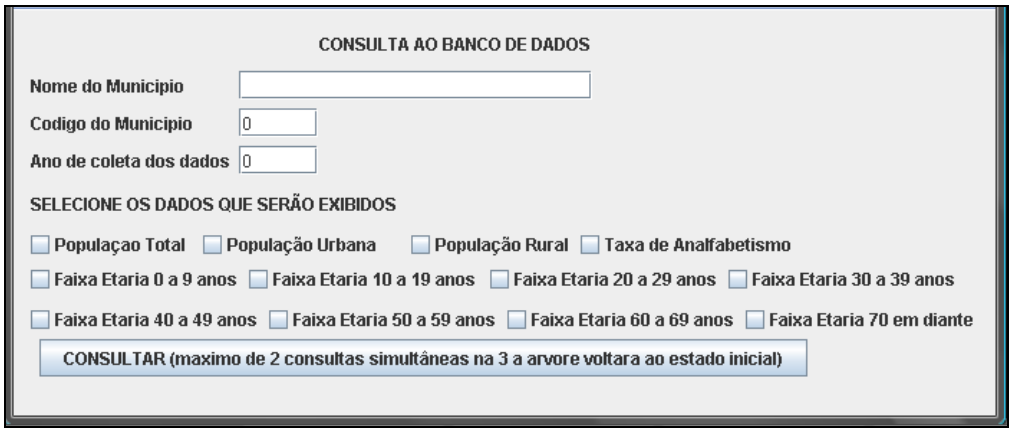

Figura 22 - Imagem apresentando os campos necessários para realização das consultas assim como as caixas de seleção dos dados demográficos a serem buscados.

```
 //verifica antes a existencia dos dados do municipio no banco
if (connection.existeMunicipio(jTextField_Busc_NomeMun.getText(),
                               Integer.parseInt(jTextField_Busc_CodMun.getText()),
                               Integer.parseInt(jTextField_Busc_Ano.getText()))== true)
 {
```

Figura 23 - Representação do trecho de código que faz chamada ao método "existeMunicipio".

Após a verificação da existência dos dados código, faz se a verificação de qual *Check Box* esta selecionado para que seja sabido os dados que o usuário deseja visualizar, caso o *CheckBox* esteja selecionado uma variável, tipo *boolean*, global recebera o valor *true*. A Figura 24 apresenta um trecho de código no qual a verificação de *CheckBox* é realizada.

```
//verifica quais check box estao selecionados
if(jCheckBox_Busc_PopTot.isSelected() == true )
{
    CheckPopTot = true;
}else
   CheckPopTot = false;

if(jCheckBox_Busc_PopUrb.isSelected() == true )
{
    CheckPopUrb = true;
}else
    CheckPopUrb = false;
```

Figura 24 - Representação de trecho de código no qual se verifica a seleção de quatro *Check Box*.

Feita a verificação dos *CheckBox* em seguida é feita a instanciação da classe "*ShowArvoreFrutas*", que se estende de um tipo *Applet* e se encapsula em um *JFrame*, Tal classe gera a estrutura de visualização de dados,.A figura 25 apresenta o trecho de código responsável pela instanciação.

```
//caso ache o municipio monta a estrutura com as frutas
//selecionadas nos check boxes
Arvore.add(new ShowArvoreFrutas());
Arvore.setVisible(true);
Arvore.setSize(578,800);
Arvore.setLocation(701,0);
```

Figura 25 - Representação de trecho de código no qual a estrutura de visualização é gerada.

4.1.3 - Descrição da Classe "*DBConnection*"

Na classe *DBConnectio*n estão contidos todos os métodos relacionados ao Banco de Dados.

Algumas de suas funcionalidades principais são abordadas nos sub-capítulos seguintes.

4.1.3.1 - Iniciar o Banco de Dados

A iniciação do Banco de Dados ocorre por meio da função "*initDB*", a qual primeiramente verifica a existência da tabela "*MuniTable*", a tabela descrita abriga todos os dados demográficos juntamente com o nome do município, código, e ano de coleta de dados onde os dois últimos citados formam uma chave primaria composta, caso exista a tabela é apagada e uma nova tabela de mesmo conteúdo e nome é criada porem sem nenhum dado preenchido, caso a tabela não exista simplesmente cria se uma nova tabela "*MuniTable*". A Figura 26. Apresenta o método responsável pela iniciação do Banco de Dados.

4.1.3.2 - Adicionando Conteúdo ao Banco de Dados

A adição de conteúdo é realizada através do método "*addMunicipio*", no método abordado os dados provindos da classe "JanelaPrincipal" serão utilizados como parâmetro

para o método abordado, e utilizando de códigos da linguagem SQL é realizada a inserção de dados. A Figura 27 apresenta o método de adição de dados.

```
  /*
    Cria uma nova tabela de DadosMunicipios (MuniTable), apagando quaisquer dados
   * ou tabela criada anteriormente
   */
public void initDB()
{
try {
     this.stm.executeUpdate("DROP TABLE IF EXISTS MuniTable");
     this.stm.executeUpdate("CREATE TABLE MuniTable (nome varchar(50) NOT NULL, muniCodigo integer NOT NULL," +
             "ano integer NOT NULL,pop_total integer , pop_urb float , pop_rur float , tax_analf float  ," +
             "fe_0_9 float ,fe_10_19 float , fe_20_29 float , fe_30_39 float , fe_40_49 float , " +
             "fe_50_59 float , fe_60_69 float , fe_70_diante float ," +
             "PRIMARY KEY (muniCodigo,ano))");
     this.stm.close();

    }
catch (SQLException e)
   {
     System.out.println(e.getMessage());
      e.printStackTrace();
    }
}
```

Figura 26 - Representação de trecho de código que realiza a inicialização do Banco de Dados.

```
  /*
   Adiciona uma nova tupla na tabela de Municipios.
   */
public void addMunicipio(String nome, int codigo, int ano, int pop_total, float pop_urb, float pop_rur,
                         float tax_analf, float Fe_0_9, float Fe_10_19, float Fe_20_29, float Fe_30_39,
                         float Fe_40_49, float Fe_50_59, float Fe_60_69, float Fe_70_diante)
{
    try {
      this.stm = this.conn.createStatement();
      this.stm.executeUpdate("INSERT INTO MuniTable VALUES (\""
               + nome + "\"," + codigo + ","+ ano +","+ pop_total + "," + pop_urb + ","
               + pop_rur+ "," + tax_analf+ ","+ Fe_0_9+ "," + Fe_10_19+ ","
               + Fe_20_29+"," + Fe_30_39+ ","+ Fe_40_49+"," +Fe_50_59+ ","
               + Fe_60_69+","+ Fe_70_diante+")");
      this.stm.close();
     }
     catch (SQLException e)
     {
      e.printStackTrace();
     }
}
```

Figura 27 - Representação o trecho de código do método de inserção de dados

4.1.3.3 - Removendo Conteúdo do Banco de Dados

A remoção de conteúdo é realizada através do método “removeMunicipio”. No método abordado os dados: nome, código e ano são fornecidos pelos campos “Nome do Municipio”, “Codigo do Municipio” e “Ano de Coleta dos Dados”, respectivamente, da classe “JanelaPrincipal”, e são utilizados como parâmetros, a tupla contendo os mesmos valores descritos será eliminada da tabela. A Figura 28 apresenta o método de remoção de dados.

```
  /*
    Remove a linha da tabela cuja coluna "nome" "muniCodigo" e "ano" seja igual a string passada
    como parâmetro.
  */
public void removeMunicipio(String nome,int codigo,int ano)
{
  try {
       this.stm = this.conn.createStatement();
       this.stm.executeUpdate("DELETE FROM MuniTable WHERE nome ='" + nome + "' and muniCodigo = "
                              + codigo+" and ano = "+ano);
       this.stm.close();
      }
  catch (SQLException e)
  {
    e.printStackTrace();
  }
}
```

Figura 28 - Representação o trecho de código do método de remoção de dado

4.1.3.4 - Recuperando Dados Demográficos

Para se recuperar os dados regionais que foram selecionados na classe “JanelaPrincipal”, utiliza se de vários métodos “*getters**” ,um para cada dado demográfico, cada método retorna seu respectivo valor, e possui como parâmetros de entrada o código do município e o ano de coleta dos dado, tais valores são utilizados na criação da estrutura de visualização. A Figura 29 apresenta um dos métodos *get* da classe “*DBConnection*”.

4.1.3.5 - Verificação de existência

O método “existeMunicipio” da classe “*DBConnection*”, recebe como parâmetro o nome do município a ser verificado, assim como o seu código e ano de coleta de dados, o

**Getters*: Métodos que retornam o valor de algum atributo de um objeto.

retorno é de tipo *boolean*, retornando *true* caso o município exista no Banco de Dados e *false* caso não encontre. A Figura 30 apresenta o trecho de código de um dos métodos descrito.

```
// metodo que retorna a populaçao total de determinado municipio
public int getPopTot(int cod, int ano)
{
    ResultSet rs;
    int popTot = 0;
    try {
          rs = this.stm.executeQuery("SELECT pop_total FROM MuniTable where muniCodigo = "+ cod +" AND ano = "+ano);
          if (rs.next())
          {
          popTot=rs.getInt("pop_total");
          }
         this.stm.close();
         }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
  return popTot;
}
```

Figura 29 - Representação do trecho de código que retorna o valor do dado demográfico População Total.

```
//faz a verifica a existencia do municipio no banco de dados
public boolean existeMunicipio(String nome, int codigo,int ano)
{
    ResultSet rs;
    try {
         rs = this.stm.executeQuery("SELECT * FROM MuniTable where nome = '"+ nome +"' AND muniCodigo = "+codigo+" AND ano = "+ano);
          if (rs.next())
           {
            existe = true;
          }else
              existe = false;
          this.stm.close();
        }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
    return existe;
}
```

Figura 30 - Representação do trecho de código com o método "existeMunicipio".

4.1.4 - Descrição da Classe "*ShowArvoreFrutas*"

A Classe "*ShowArvoreFrutas*" é responsável por montar toda a estrutura básica de visualização tridimensional de dados.

A estrutura principal utilizada para agrupar os dados é representada por uma árvore, para representar os dados demográficos foi modelada uma fruta a qual é replicada para representar os vários tipos de dados demográficos, sendo que cada fruta apresenta uma cor

predeterminada para representar cada tipo de dado e posição predeterminada na arvore, para que nos casos de consulta composta as comparações estatísticas sejam melhor visualizadas sendo que frutas de mesma cor ficam posicionadas em um mesmo galho. O valor de cada dado poderá ser visualizado para comparações estatísticas através da mudança no tamanho de sua fruta, considerando que a mudança de tamanho representa a porcentagem do valor buscado

Tal estrutura de arvore foi escolhida pela facilidade de visualização e agrupamento dos dados demográficos, os quais são representados pelas frutas.

Por exemplo: ao realizar uma busca por população urbana de determinado município, será exibida na arvore uma fruta na cor verde em seu respectivo galho, e dependendo do valor do dado, seu tamanho será alterado com relação a uma fruta de tamanho padrão, se o dado tiver como valor 0.5 ou 50% o seu tamanho terá de metade do tamanho da fruta que representa a população total que por definição possui sempre o tamanho padrão.

As Figuras 31 e 32 apresentam a estrutura gráfica da árvore e a fruta utilizada para representar os dados, respectivamente.

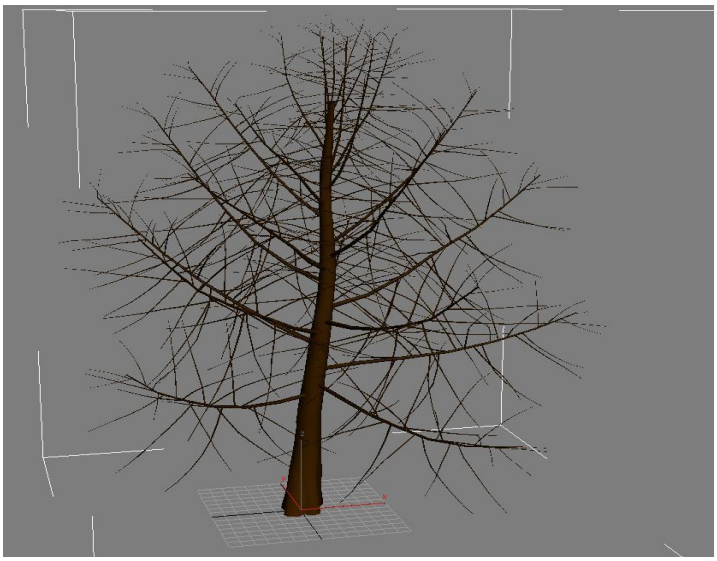

Figura 31 - Representação gráfica da arvore com seu tronco e seus galhos.

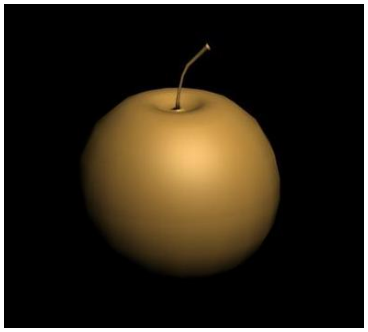

Figura 32 - Representação gráfica da fruta, a qual sofre variações de acordo com a consulta.

No capítulo a seguir serão descritos os passos seguidos para a obtenção da estrutura completa, onde as frutas estão devidamente posicionadas e coloridas, juntamente com a árvore.

4.1.4.1 – Árvore

A árvore é a estrutura principal da visualização dos dados, ela é responsável pelo agrupamento de todos os dados referentes à demografia.

Antes de realizar a instanciação da estrutura de arvore primeiramente foram definidos alguns atributos que a estrutura deve conter com relação à aparência e iluminação. Posteriormente a estrutura de arvore é importada no formato .obj, em seguida é setada a aparência e a iluminação para a mesma. Os atributos abordados serão descritos a seguir.

A - Aparência da Árvore

Para setar os atributos referentes à aparência é necessário instanciar a classe *Appearance* da que será responsável pelo encapsulamento de suas propriedades referentes ao material, atributos poligonais, de linha, de ponto e transparência. A Figura. 33 apresenta o trecho de código que instancia a classe *Appearance* e suas propriedades referentes a estrutura de árvore.

A cor escolhida para a estrutura de árvore é a cor marrom, e foi atribuída na instanciação da classe “Material” onde o atributo de cor ambiente foi modificado para gerar a cor desejada.

B- Iluminação da Árvore

Para setar os atributos referentes a iluminação foi criado um método cujo retorno é um “*TransformGoup*”, no método gerou-se um “*TransformGroup*” o qual devera encapsular as propriedades de iluminação e servir como retorno do método, o tipo de luz utilizado é o “*AmbientLight*”. A Figura 34 apresenta o trecho do código responsável pelo método de iluminação da árvore.

```
//aparencia da arvore
 private Appearance aparenciaArvore()
 {
     Appearance appArvore = new Appearance();

     Material materialArvore = new Material(
             new Color3f(0.60f,0.30f,0.0f),
             new Color3f(0.0f,0.0f,0.0f),
             new Color3f(0.9f,0.9f,0.9f),
             new Color3f(1.0f,1.0f,1.0f),64.0f);
     appArvore.setMaterial(materialArvore);

     PolygonAttributes PolyAttArvore = new PolygonAttributes();
     PolyAttArvore.setPolygonMode(PolygonAttributes.POLYGON_FILL);
     PolyAttArvore.setCullFace(PolygonAttributes.CULL_NONE);

     LineAttributes lineAttrArvore = new LineAttributes();
     lineAttrArvore.setCapability(lineAttrArvore.ALLOW_PATTERN_WRITE);
     lineAttrArvore.setLinePattern(LineAttributes.PATTERN_SOLID/*PATTERN_DASH*/);
     appArvore.setLineAttributes(lineAttrArvore);

     PointAttributes pointAttrArvore = new PointAttributes();
     pointAttrArvore.setCapability(PointAttributes.ALLOW_SIZE_WRITE);
     pointAttrArvore.setPointSize(50.0f);
     pointAttrArvore.setPointAntialiasingEnable(true);
     appArvore.setPointAttributes(pointAttrArvore);

     TransparencyAttributes transparencyArvore = new TransparencyAttributes(TransparencyAttributes.NICEST, -0.0f );
     appArvore.setTransparencyAttributes(transparencyArvore);

     return appArvore;
 }
```

Figura 33 - Representação de trecho de código que instancia a classe *Appearance* e suas propriedades referentes à estrutura de árvore.

```
//iluminaçao arvore
 private TransformGroup iluminacaoArvore()
 {
     TransformGroup tgLuzArvore = new TransformGroup();
     BoundingSphere boundArvore = new BoundingSphere( new Point3d(0,0,0),0.5d);
     Color3f corLuzArvore = new Color3f(0.60f,0.30f,0.0f);
     AmbientLight luzAmbientArvore = new AmbientLight(true,corLuzArvore);
     luzAmbientArvore.setInfluencingBounds(boundArvore);
     tgLuzArvore.addChild(luzAmbientArvore);

     return tgLuzArvore;
 }
```

Figura 34 - Representação do trecho de código responsável pelo método de iluminação da árvore.

C- Importando a Árvore

Para realizar a importação da estrutura de árvore para o sistema foi implementado um método que tem como retorno um "*TransformGroup*" o qual é responsável por encapsular a estrutura. Para tal a Figura 35 apresenta o trecho de código responsável pela importação da estrutura para o sistema.

Após a importação da estrutura é necessário posicioná-la na cena virtual e setar seu tamanho (FIGURA 36), realizar a atribuição de sua aparência (FIGURA 37) e iluminação (FIGURA 38) e setar permissão para a capacidade de se realizar transformações geométricas na estrutura (FIGURA 39).

```
try {
  cena = arquivo.load("c:/Objetos3D/TroncoGalhos.obj");
} catch (FileNotFoundException e) {
  System.err.println(e);
  System.exit(1);
} catch (ParsingErrorException e) {
  System.err.println(e);
  System.exit(1);
} catch (IncorrectFormatException e) {
  System.err.println(e);
  System.exit(1);
}
```

Figura 35 - Representação do trecho de código responsável pela importação da estrutura de árvore.

```
Transform3D tArvore = new Transform3D();
//seta a posição inicial da arvore
tArvore.setTranslation(new Vector3f(0.0f,0.0f,0.0f));
//seta o tamanho da arvore
tArvore.setScale(1d);
```

Figura 36 - Representação do trecho de código responsável pelo posicionamento e tamanho da estrutura de árvore na cena.

```
//seta a aparencia da arvore
((Shape3D)(cena.getSceneGroup().getChild(0))).setAppearance(this.aparenciaArvore());
objTrans.addChild(cena.getSceneGroup());
```

Figura 37 - Representação do trecho de código responsável por atribuir a aparência a estrutura de árvore.

```
// seta a iluminaçao da arvore
objTrans.addChild(this.iluminacaoArvore());
```

Figura 38 - Representação do trecho de código responsável por atribuir a iluminação a árvore.

```
//Permissao para realizar transformações geometricas na arvore
objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
objTrans.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
```

Figura 39 - Representação do trecho de código responsável pela permissão da capacidade de se realizar transformações geométricas na árvore.

4.1.4.2 - Frutas

As frutas que serão apresentadas, juntamente a arvore, representam os dados demográficos e poderão ser diferenciadas através de suas cores e posições diferentes, a estatística poderá ser visualizada através das proporções de seus tamanhos com relação com o tamanho padrão da fruta importada.

Assim como a árvore é necessário que se defina os atributos de aparência e iluminação para que sejam atribuídos a fruta no método de importação da mesma.

A - Aparência das Frutas

Para cada fruta foi definido uma cor diferentes, pois dessa forma é possível diferenciar cada uma delas, o atributo de cor de cada fruta pode ser alterado através da instanciação da classe “Material”, onde encontramos quatro atributos de reflexão de cor e um atributo para o brilho, o atributo referente à cor ambiente foi modificado de forma diferente para cada fruta desta forma cada uma delas possuirá uma cor diferente.

B - Iluminação das Frutas

A iluminação das frutas foi realizada através de fontes de luz do tipo “*AmbientLight*” para que as frutas sejam iluminadas de forma homogênea e por todas faces facilitando a visualização de suas cores.

C - Importando as Frutas

As frutas são importadas da mesma forma que a arvore, porém diferenciam-se na posição e tamanho.

As posições foram definidas de forma que as frutas aparentem estar fixas aos galhos da arvore.

Seu tamanho será setado de acordo com o retorno do método *get* da classe "*DBConnection*" referente ao dado que a fruta representa. A Figura 40 apresenta o trecho de código no qual e setado o tamanho da fruta através da chamada de seu respectivo método *get* da classe "*DBConnection*".

```
//faz a busca no banco para setar o scale da fruta
try{
DBConnection connection = new DBConnection("file");
popUrb = connection.getPopUrb(JanelaPrincipal.codMunicipio,JanelaPrincipal.anoMunicipio);

}catch (java.sql.SQLException e){}
catch ( ClassNotFoundException e1){}

// seta o tamanho da fruta
t3dFrutaPopUrb.setScale(padraoTamanho*popUrb);
```

Figura 40 - Representação do trecho de código responsável por setar o tamanho da fruta.

Para que seja visualizada a estrutura completa onde a arvore apresenta as diversas frutas consultadas foi criado o método "*createSceneGraph*" o qual retorna um tipo "*BranchGroup*", o método é responsável por criar a cena tridimensional, para tal primeiramente declara-se um "*BranchGroup*" que devera encapsular um "*TransformGroup*", esse por sua vez será responsável por encapsular a estrutura completa, abrigar a iluminação geral da cena e os comportamentos da estrutura, responsáveis pela interatividade com o usuário.

D - Gerando a Cena

Para se criar a estrutura completa é anexada primeiramente a arvore através de seu método de importação, posteriormente para cada fruta é verificado se ela foi selecionada para

ser exibida na consulta em caso afirmativo ela será anexada da mesma forma que a arvore, no caso de ser uma consulta composta onde duas frutas deverão ser exibidas, outra fruta de mesma cor é anexada porém com posições diferentes, para que seja possível visualizar as duas frutas sem que uma fique por cima da outra, porem no mesmo galho. A Figura 41 apresenta o trecho de código responsável por instanciar o "*BranchGroup*" de retorno, o "*TransformGroup*", anexar a arvore e as frutas, responsáveis por representar o dado de população total do município e população urbana, a cena.

Para tornar a cena mais realista foi necessário alterar a cor de fundo padrão "preta" para "azul céu" e criar uma iluminação própria para a cena, a iluminação escolhida foi a luz direcional pelo efeito obtido. A Figura 42 apresenta o trecho de Código responsável por alterar a cor de fundo e atribuir a iluminação a cena. A Figura 43 apresenta um exemplo de cena completa, gerada a partir de dados demográficos de teste, com a estrutura de arvore e as frutas.

```
// cria a cena
private BranchGroup createSceneGraph()
{
    BranchGroup root = new BranchGroup();

    TransformGroup tGroup = new TransformGroup();

        //add dos objetos na cena scene
        // adiciona a arvore a cena
        tGroup.addChild(this.ArvoreImport());

        // adiciona as frutas que foram selecionadas na JanelaPrincipal
        if(JanelaPrincipal.CheckPopTot){
        tGroup.addChild(this.FrutaPopTotImport());
        }
        if(JanelaPrincipal.CheckPopTot && JanelaPrincipal.clickCount % 2 != 0){
        tGroup.addChild(this.FrutaPopTotImport2());
        }

        if(JanelaPrincipal.CheckPopUrb){
        tGroup.addChild(this.FrutaPopUrbImport());
        }
        if(JanelaPrincipal.CheckPopUrb && JanelaPrincipal.clickCount % 2 != 0){
          tGroup.addChild(this.FrutaPopUrbImport2());
        }
```

Figura 41 - Representação do trecho de código responsável por criar parte da cena de visualização.

```
//seta a cor de fundo
Color3f bgColor = new Color3f(0.05f, 0.74f, 1f);
Background bgNode = new Background(bgColor);
bgNode.setApplicationBounds(bounds);
root.addChild(bgNode);

//iluminação geral
Color3f corLuzGeral = new Color3f(0.5f,0.5f,0.5f);
Vector3f direcaoLuzGeral = new Vector3f(-1.0f,-1.0f,-1.0f);
DirectionalLight luzDirecionalGeral = new DirectionalLight(true,corLuzGeral,direcaoLuzGeral);
luzDirecionalGeral.setInfluencingBounds(bounds);
tGroup.addChild(luzDirecionalGeral);
```

Figura 42 - Representação de trecho de código responsável por alterar a cor de fundo e gerar a iluminação da cena.

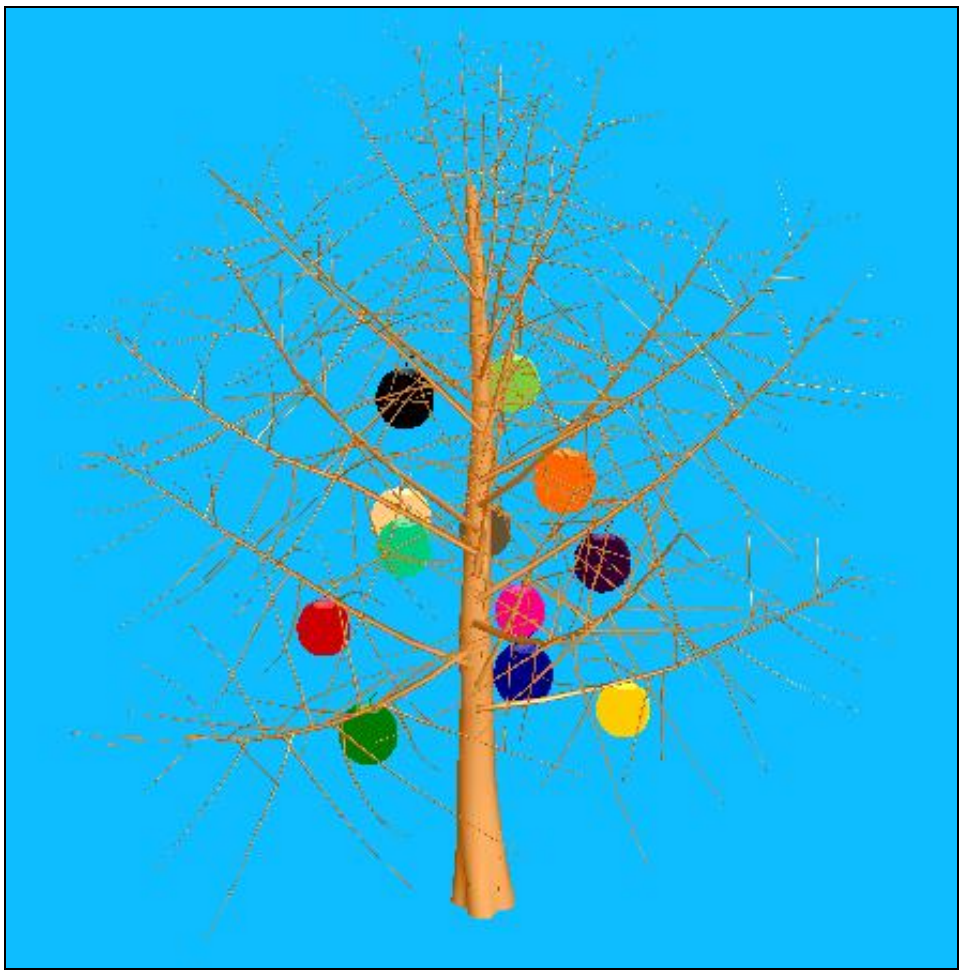

Figura 43 - Exemplo da organização completa da cena.

Para oferecer interação com o usuário foram gerados alguns comportamentos básicos que permite ao usuário rotacionar e aplicar zoom à estrutura, a interação da estrutura com o usuário contribui com a melhor visualização dos dados facilitando desta forma a compreensão

da informação gerada. A Figura 44 apresenta o trecho de código responsável por atribuir os comportamentos de interação da estrutura com o usuário.

```
//atribui os Comportamentos a cena
MouseRotate behavior = new MouseRotate(tGroup);
tGroup.addChild(behavior);
behavior.setSchedulingBounds(bounds);

MouseWheelZoom behavior3 = new MouseWheelZoom(tGroup);
tGroup.addChild(behavior3);
behavior3.setSchedulingBounds(bounds);
```

Figura 44 - Representação de trecho de código responsável por atribuir os comportamentos a cena.

CAPITULO V - RESULTADOS

O presente projeto proporciona à organização dos dados demográficos relacionados entre si, e visualizados sob estruturas tridimensionais representativas únicas para cada situação de busca no banco de dados. Com isso espera-se que consultas ao banco de dados sejam melhor interpretadas de forma rápida e intuitiva, considerando a apresentação visual intuitiva dos resultados buscados.

Para realização de Testes foram adicionados os dados demográficos referentes ao município de Marília nos anos de 1991 a 2000, de acordo com as informações obtidas através de relatórios descritos pela SAEDE e CEURB. Alguns dos dados foram adaptados para serem utilizados no DemoVis.

5.1 - Incluindo os Dados Demográficos

Os dados demográficos referentes à cidade de Marília no ano de 1991 e 2000 foram inseridos conforme descritos nas Figuras 45 e 46 respectivamente.

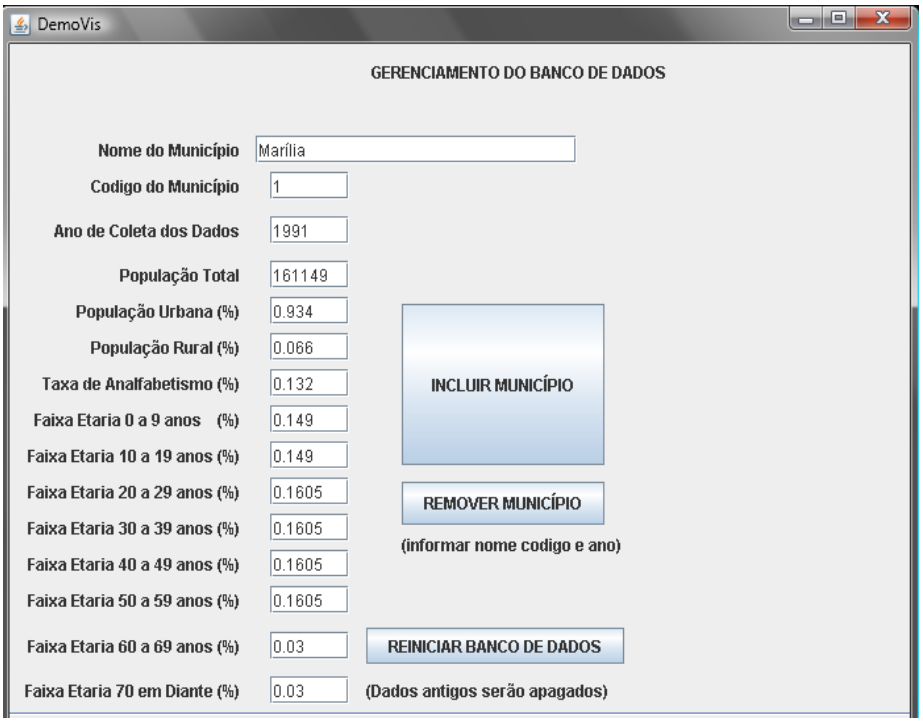

Figura 45 - Inclusão de dados demográficos referentes ao ano de 1991 da cidade de Marília.

Dessa forma as informações relacionadas a demografia serão armazenadas no Banco de Dados.

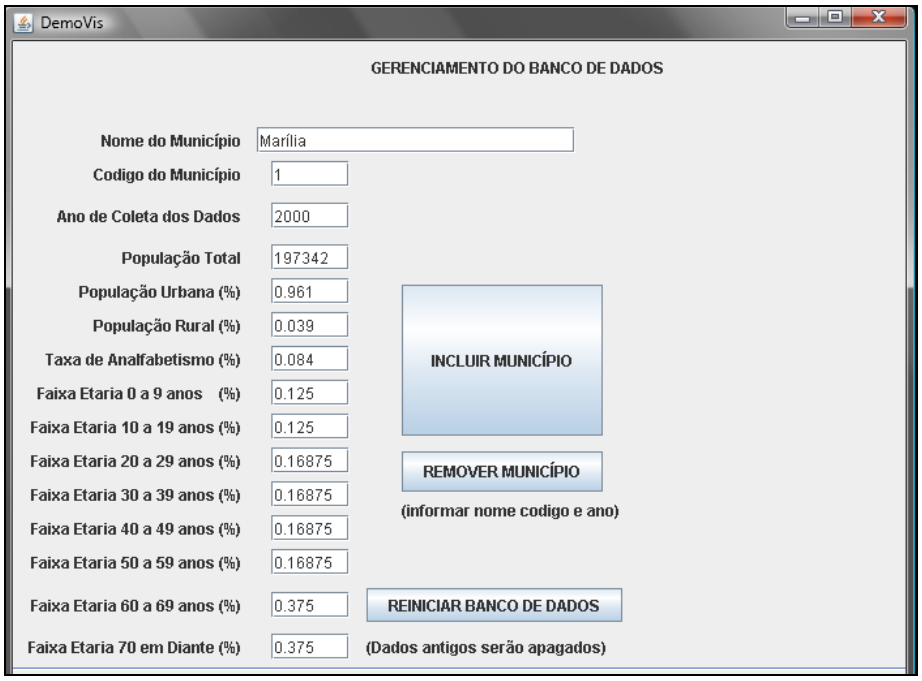

Figura 46 - Inclusão de dados demográficos referentes ao ano de 2000 da cidade de Marília.

A mesma situação da Figura 45 ocorre na Figura 46, porem os dados inseridos são referentes ao município de Marília no ano de 2000.

5.2 - Visualizando os Dados

Para a visualização, devido ao valor de alguns dos dados serem muito pequenos algumas da frutas não eram visíveis, para tal foi tomada a estratégia de se tratar a escala de tamanho, multiplicando seu valor para que a fruta se tornasse visível, mantendo um padrão linear de aumento de tamanho entre os valores menores que 20% e maiores que 1% para que tal redimensionamento não atrapalhe no momento de realizar comparações com os dados, os valores aplicados no redimensionamento da escala estão presentes na janela de legenda para que o usuário tenha conhecimento de quando e quanto os dados são redimensionados. As Figuras 47, 48 e 49 apresentam respectivamente, a consulta realizada na janela principal do DemoVis, a janela de legenda onde estão contidos os detalhes da estrutura e a própria estrutura obtida na consulta aos dados relacionados ao município de Marília no ano de 1991

onde todos os dados relacionados a demografia foram selecionados, recordando que tal estrutura pode ser rotacionada e ampliada para melhor visualização, interpretação e compreensão dos dados.

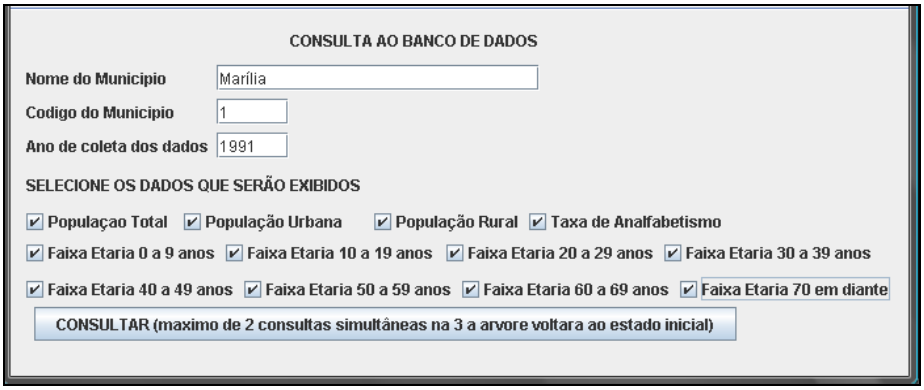

Figura 47 - Imagem apresentando a consulta realizada através da janela principal do DemoVis por dados referentes ao município de Marília no ano de 1991.

Para realizar a consulta os dados devem ser informados na área de consulta ao banco de dados conforme a Figura 47.

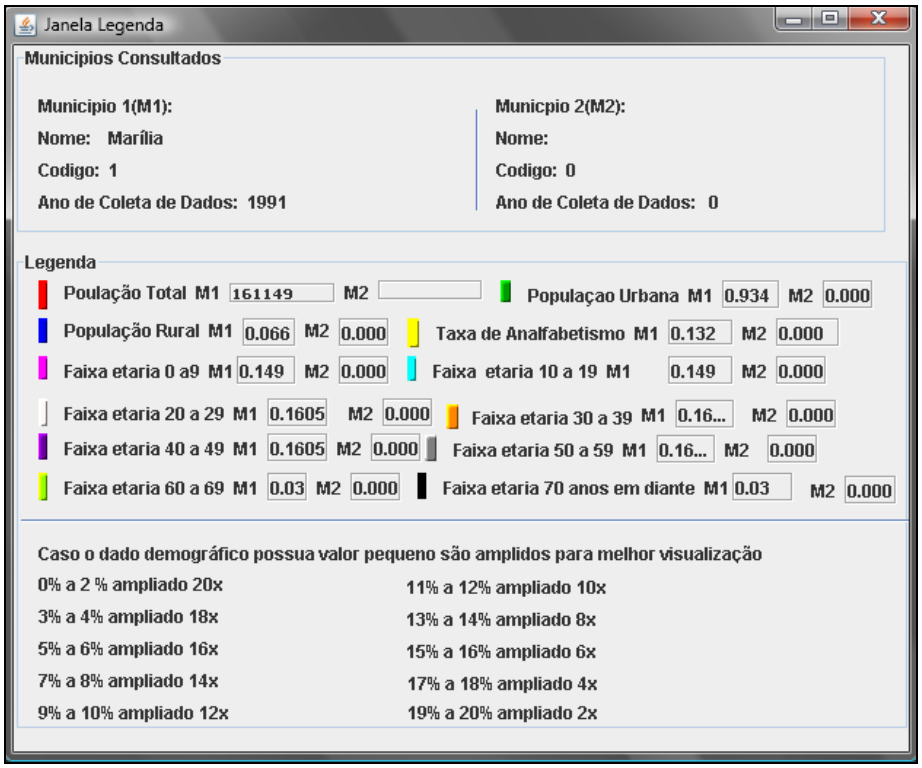

Figura 48 - Imagem apresentando a legenda com os detalhes a respeito do município consultado.

Após a consulta uma janela com legendas relacionadas a estrutura será exibida contendo os detalhes relacionados aos dados demográficos onde são apresentados as cores das frutas para identificação o valor de cada dado e a descrição do nome código e ano de coleta dos dados, desta forma o usuário é capaz de assimilar a informação a estrutura gerada.

Figura 49 - Imagem apresenta a estrutura resultante da consulta realizada, referente ao município de Marília no ano de 1991.

A estrutura de árvore gerada pela consulta apresenta dados reais relacionados ao município de Marília e que podem ser identificados na janela de legenda, algumas das frutas foram redimensionadas para melhor visualização, tal redimensionamento esta devidamente descrita na janela de legenda.

Assim como foi explicada no capitulo anterior cada dado demográfico esta sendo representado por uma fruta de cor diferente e posição predeterminada visando a organização

dos dados na árvore, e dependendo do valor do dado o tamanho da fruta é alterada para que sejam realizadas comparações estatísticas.

Através do DemoVis é possível ainda realizar consultas compostas onde dois municípios são consultados e seus dados são apresentados em uma mesma estrutura possibilitando o usuário fazer comparações entre municípios diferentes ou municípios iguais porem com a data de coleta de dados diferente, para fins estatísticos. As Figuras50, 51 e 52 apresentam respectivamente, uma segunda consulta ao município de Marília, porem com o ano de coleta de dados igual a 2000, a janela de legenda com os detalhes relacionados à estrutura onde os dados de Marília nos anos de 1991 e 2000 são apresentados, a estrutura resultante da consulta composta onde para cada dado será representado por duas frutas neste caso uma para cada ano de coleta de dados, poderia ocorrer de ser dois municípios diferentes em outra ocasião.

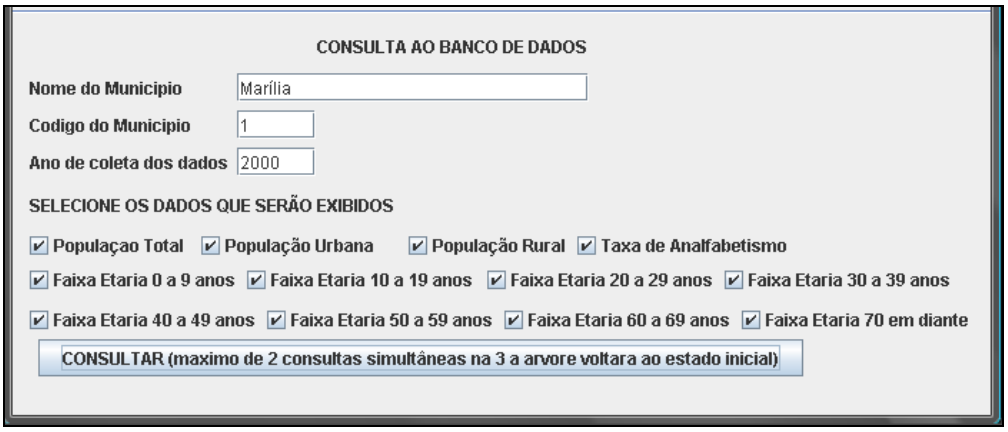

Figura 50 - Imagem apresenta a segunda consulta realizada.

Uma segunda consulta ocorre da mesma forma como na Figura 47, porem com ano de coleta de dados diferentes.

Outro caso de consulta composta poderia ser exemplificada, onde poderia ser realizada uma primeira consulta a cidade de Marília com o ano de coleta de dados 2000 e uma segunda consulta a cidade de Bauru como o mesmo ano de coleta de dados (caso o município tenha sido devidamente adicionado ao banco de dados)

Os dados exibidos na janela de legendas em uma consulta composta agora apresentam dados relacionados a dois municípios ou a um mesmo município porem com ano de coleta de dados diferentes.

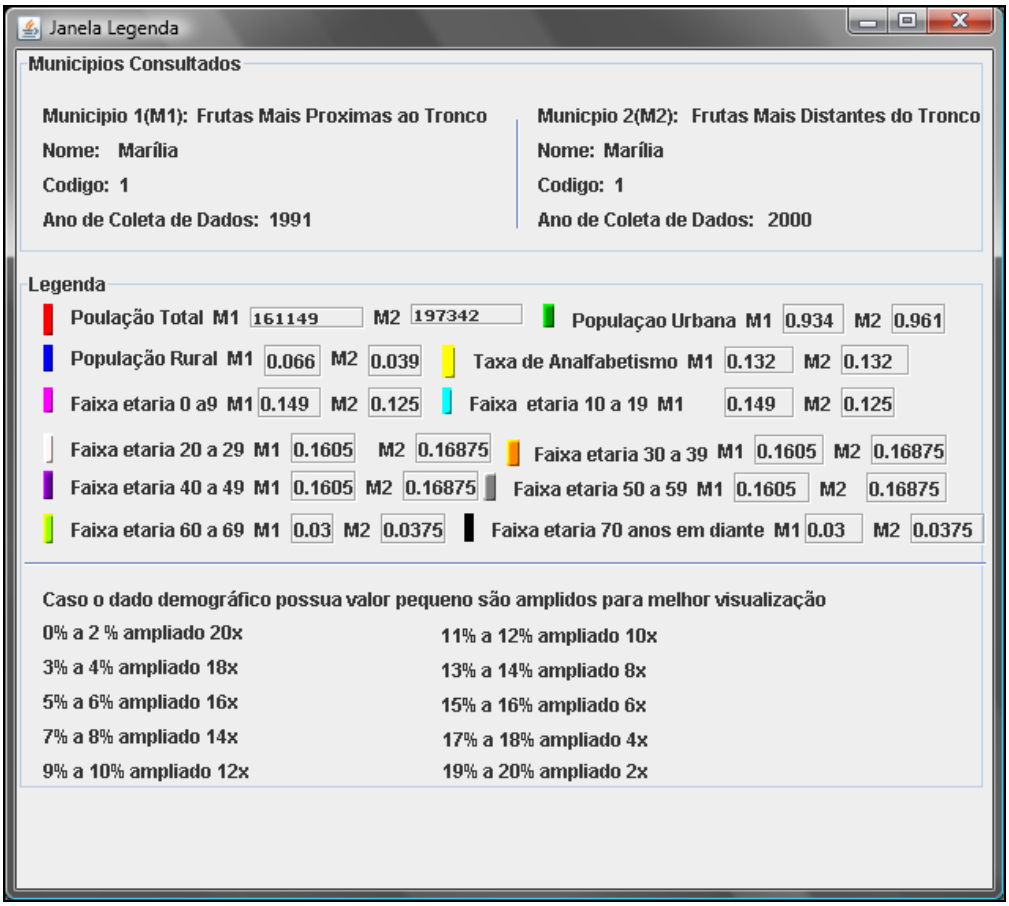

Figura 51 - Imagem apresentando a janela legenda referente à consulta composta realizada.

Na Figura 52 podemos observar a existência de duas frutas de cada cor sendo que as frutas mais próximas ao tronco representam dados referentes ao primeiro município consultado, neste caso o município de Marília no ano de 1991, e as frutas mais distantes do tronco representam os dados referentes ao segundo município consultado, neste caso o município de Marília no ano de 2000.

As frutas que possuem cores iguais estão todas posicionadas no mesmo galho, pois desta forma a visualização para se realizar comparações estatísticas é facilitada.

A posição das frutas foram predeterminadas de forma que sua visualização não ficasse comprometida auxiliando na compreensão e obtenção de informação, lembrando que a estrutura pode ser rotacionada, para os casos em que algo dificulte a visualização da fruta assim como o próprio tronco da árvore ou até mesmo uma outra fruta que fique na frente de um dado que o usuário deseja visualizar. Desta forma ao rotacionar a estrutura o dado poderá ser melhor visualizado, ou ampliada para nos casos em que o usuário necessite visualizar melhor determinado dado em particular.

Figura 52 - Imagem apresenta a estrutura gerada a partir da consulta composta relacionada ao município de Marília nos anos de 1991 e 2000.

CAPITULO VI - CONCLUSÕES

Até o presente momento, por meio de revisão da literatura relacionada, conclui-se a ausência de ferramentas relacionadas que explorem e qualifiquem a visualização de dados regionais. Observou-se também que o desenvolvimento de um sistema que facilite a observação de atributos demográficos pode contribuir para o desenvolvimento econômico regional.

Desta maneira, o objetivo deste trabalho foi o desenvolvimento de um ambiente de visualização tridimensional, para organizar e representar dados relativos à demografia, através de estruturas intuitivas e interativas, auxiliando em pesquisas e coleta de informações trazendo conhecimento e contribuindo em tomadas de decisões, a respeito de determinadas regiões e seus municípios.

Através do sistema gerado pôde-se observar que o mesmo agregou grande contribuição na área de pesquisas demográficas, por ser capaz de complementar o uso de vários gráficos diferentes relacionados à demografia, devido a capacidade de exibir os vários dados distintos relacionados à demografia em uma única estrutura interativa (FIGURA 53), facilitando desta forma a visualização e compreensão dos dados por parte do usuário.

Em trabalhos futuros existem as possibilidades de tornar o sistema desenvolvido em um sistema imersivo ou, que vise à utilização de dispositivos de realidade virtual com técnicas de realidade aumentada, adicionalmente sugere-se portar o ambiente para ambiente web.

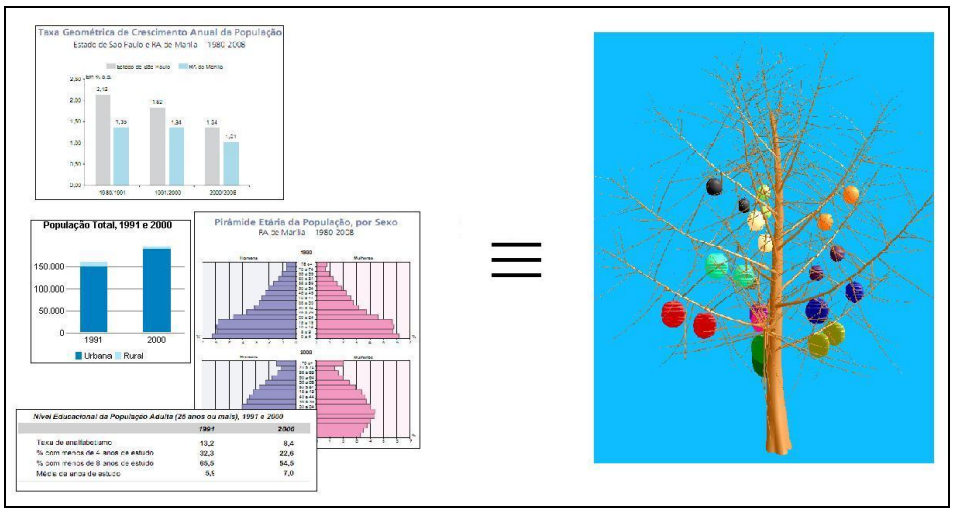

Figura 53 - Imagem apresenta a forma como o DemoVis pode substituir vários gráficos e tabelas por apenas uma estrutura.

REFERENCIAS BIBLIOGRÁFICAS

AUTODESK. WAVE FRONT. "*Wave Front File Format Specification*". Disponível em <http://www.fileformat.info/format/wavefrontobj/spec/index.htm>. Acesso em Julho 2009.

AUTODESK. "*3D Studio Max specification*". Disponível em <http://usa.autodesk.com/adsk/servlet/index?siteID=123112&id=9861583>(2009).

AZEVEDO, E; CONCI, A. "Computação gráfica: teoria e prática". Rio de Janeiro: Campus, ISBN 85-352-1253-3. (2003).

BECKMAN INSTITUTE FOR ADVANCED SCIENCE & TECHNOLOGY. *Technology Group*, UIUC. Disponível em <http://www.itg.uiuc.edu/vmil/> Acesso em novembro 2009.

BERTIN, J."*Semiology of Graphics: Diagrams, Networks, Maps (W. J. Berg, Trans.)*". Madison, WI: University of Wisconsin Press. (1967).

BOTEGA, L. C. "Implementação de Estereoscopia de Baixo Custo Para Aplicações em ferramentas de Realidade Virtual para Treinamento Médico". Monografia apresentada na conclusão de curso. Fundação de Ensino Eurípides Soares da Rocha, Centro Universitário Eurípides de Marília.(2005).

CEURB. "Perfil Municipal – Marília". Centro Virtual de Estudos Ambientais Urbanos, Universidade Estadual Paulista "Julio de Mesquita Filho" (UNESP). (2000).

CHEN, C. "*Information Visualization and Virtual Environments*". Springer, London. (1999).

CARD,S.K., e MACKINLAY, J.D; SHNEIDERMAN, B. "*Readings in Information Visualization: Using Vision to Think*". Morgan Kaufmann Series in Interactive Technologies. Morgan Kaufmann Publishers Inc. (1999).

CARD, S. K., ROBERTSON, G.G., e MACKINLAY, J. D. "*The Information Visualizer: An Information Workspace*". Proceedings of CHI`91. ACM Conference on Human Factos in Computing Systems, New Orleans. 181-188. (1991).

CASNER, S. "*Task-Analytic Approach to the Automated Design of Graphic Presentations*". ACM Transatons on Graphics, 10(2,April), 111-151. (1991).

CLEVELAND, W. S., e McGILL, M. E. "*Dynamic Graphics for Statistics*". Pacific Grove, CA: Wadsworth and Brooks/Cole. (1988).

ERICK, S. G., STEFFEN, J. L., e SUMNER, E. E. "*Seesoft-A Tool for Visualizing Line Oriented Software Statistics*". IEEE Transactions on Software Engineering, 18(11-Nov.), 957-968. (1992).

FEINER, S. K., e BESHERS, C. "*Worlds Within Worlds: Metaphors for Exploring n-Dimensional Virtual Worlds*". Proceedings of UIST`90, ACM Symposium on User Interface Software and Technology. 76-83. (1990).

FREITAS, C. M. D. S; CHUBACHI, O. M; LUZZARDI, P. R. G; CAVA, R. A. "Introdução à Visualização de Informações". Revista de Informática Teórica e Aplicada, 8(2):143–158. (2001).

FRIENDLY, M. "*Milestones in the History of Thematic Cartography, Statistical Graphics, and Data Visualization*". National Sciences and Engineering Research Council of Canada, Grant OGP0138748. (2008).

GISMAPS SISTEMAS, "*Gismaps Viewer*" 2007. Disponivel em <http://www.gismaps.com.br/viewer/tutorial/gerandomapas10.htm> Acesso em novembro 2009.

GODINHO, P. I. A., MEINGUINS, B. S., MEIGUINS, A. S. G., CARMO, R. M. C., GARCIA, M. B., ALMEIDA, L. H., and LOURENÇO, R. "*PRISMA - A Multidimensional*

Information Visualization Tool Using Coodinated Views".11th International conference Information Visualization (IV`07), 0-7695-2900-3/07. (2007).

GRANDA, J. C., URIA C., GARCIA, SUAREZ, F. J., e GONZALEZ, F. "*Design Issues in Remote Visualization of Information in Interactive Multimedia E Learning Systems*". University of Oviedo, Department of Informatics Campus de Viesques, Gijon, SPAIN, IEEE 978-0-7695-3271-4/08. (2008).

GRIMSTEAD, I. J., WALKER, D. W., AVIS, J. N., KLEINERMANN, F., e McCLURE, J. "*3D Anatomical Model Visualization within a Grid-Enabled Environment*". Co published by the IEEE, CS and the AIP, 1521-9615/07. (2007).

INSELBERG, A., e DIMSDLE, B. "*Parallel Coordinates: A Tool for Visualizing Multi-Dimensional Geometry*". Proceedings of IEEE Visualization`90 onference, Los Alamitos, CA, 361-375. (1990).

JERN, M. "*3D Data Visualization on the Web*". Proceedings of IEEE Advanced Visual Systems, O-8186-8911-0/98. (1998).

KNOLL, T., ENDLICH, C., SCHERER, H. "*Conference Paper "last.forward*"" Disponivel em <http://lastforward.sourceforge.net/index.html>. acesso em novembro 2009.

KOLODNYTSKY, M., e KOVALCHUK, A. "*Interactive Software Tool for Data Visualisation*". Proceedings of IEEE international Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications 1-4 July 2001, Foros. Ukraine. (2001).

MA, K. L. "*Visualizing Visualizations*". Proceedings of IEEE, University of California at Davis, 0272-1716/00 IEEE. (2000).

MACKINLAY, J. D. "*Automating the Design of Graphical Presentations of Relational Information*". ACM Transactions on Graphic, 5(2),110-141. (1986).

MANSSOUR, I. H. “Introduçao à Java3D” 2003. Disponivel em <http://www.inf.pucrs.br/~manssour/Java3D/index.html> Acesso em Julho de 2009.

MIHALSIN, T., TIMLIN, J., e SCHWEGLER, J. “*Visualizing Multivariate Functions, Data and Distributions*”. IEEE Computer Graphics and Applications, 11(13), 28-35. (1991).

PAVARINI, L. “Proposta de Implementação de Deformação em Ferramentas Virtuais de Treinamentos Médico”. Anais do II Simpósio de Instrumentação e Imagens Medicas, Vol.II. 2005.

PLAYFAIR, W. “*Commercial and Political Atlas: Representing*”, by Copper-Plate Charts, The Progress of the Commerce, Revenues, Expenditure, and Debts of England, during the Whole of the Eighteenth Century. London: Corry. Republished in Wainer, H. and Spence, I.(eds.), The Commercial and Political Atlas and Statistical Breviary, 2005, Cambridge University Press, ISBN 0-521-85554-3. (1786).

ROTH, S. F., e MATTIS, J. “*Data Characterization for Intelligent Graphics Presentation*”. Proceedings of CHI`90, ACM Conference on Human Factors in Computing Systems, New York. 193-200. (1990).

SAEDE. “Região Administrativa de Marília”. Fundaçao Sistema Extadual de Analise de Dados, Secretaria de Econimia e Planejamento. Edição 2007.(2007).

SELMAN, D. “*Java3D programming*”.Editora *Manning Publication Cpmpany*. (2002).

SEBRAE. FIPE. “Relatório com dados regionais”. Área de abrangência do escritório regional de Bauru. 2008. (2008) .

SHNEIDERMAN, B. “*Tree Visualization with Tree-Maps: A 2 –Dimensional Space Filling Approach*”. ACM Transactions on Graphics, 11(1), 92-99. (1992).

SILVA, N. R. “Visualização 3D de Dados Oceanográficos Simulados”. Pontifica Universidade Católica do Rio Grande do Sul, Faculdade de Informática, Programa de Pós Graduação em Ciência da Computação. (2006).

SUN MICROSYSTEMS, inc. “*Java 3D 1.5.1 API Documentations*”. Sun MicroSystems.(2009).

TEYLINGEN, R. V., RIBARSKY, W., e MAST, C. V. D. “*Virtual Data Visualizer*”. IEEE Transactions on Visualization and Computer Graphics, Vol.3, no.1, January-March 1997. (1997).

TUFTE, Edward R. “*The Visual Display of Quantitative Information. Cheshire*”, CT: Graphics Press. (1983).”

TUKEY, J. W. “*Exploratory Data Analysis*”. Reading, MA: Addison-Wesley. (1977).

WINCHESTER B. “*Data Visualization Through GIS*” Coordinator, Data Analysis & Research, Center for Small Towns, University of Minnesota, Morris November 13, 2003. (2003).

ZHAO, D., ZHOU, N. “*Analysis of the Present Applications of Information*. *Visualization in E-commerce Websites*”. School of Information Management, Wuhan University, Wuhan, 430072, Department of Information Management, HuaZhong Normal University, Wuhan, 430079, China, IEEE 978-1-4244-2108-4/08. (2008).