Universidade Federal dos Vales do Jequitinhonha e Mucuri

Faculdade de Ciências Exatas

Departamento de Computação

Bacharelado em Sistema de Informação

AMBIENTE DE SIMULAÇÃO DE UMA FLORESTA TRIDIMENSIONAL

Pablo de Oliveira Castro

Diamantina - MG, Brasil

Abril de 2013

Pablo de Oliveira Castro

Ambiente de Simulação de uma Floresta Tridimensional

Diamantina - MG, Brasil

Abril de 2013

Pablo de Oliveira Castro

Ambiente de Simulação de uma Floresta Tridimensional

Monografia submetida à banca examinadora designada pelo Colegiado do Curso de Graduação em Sistemas de Universidade Federal dos Vales do Jequitinhonha e Mucuri, como parte dos requisitos da disciplina Projeto Orientado II e para obtenção do Grau de Bacharel em Sistemas de informação.

Orientador:
Alessandro Vivas Andrade

Co-orientador:
Alexandre Ramos Fonseca

UNIVERSIDADE FEDERAL DOS VALES DO JEQUITINHONHA E MUCURI
FACULDADE DE CIÊNCIAS EXATAS
DEPARTAMENTO DE COMPUTAÇÃO

Diamantina - MG, Brasil

Abril de 2013

Monografia de Projeto Final de Graduação sob o título *"Ambiente de Simulação de uma Floresta Tridimensional"*, defendida por Pablo de Oliveira Castro e aprovada em Abril de 2013, em Diamantina, Estado de Minas Gerais, pela banca examinadora constituída pelos professores:

Prof. Dr. Alessandro Vivas Andrade
Orientador

Prof. Dr. Alexandre Ramos Fonseca
Co-orientador

Prof. Rafael Santin
Universidade Federal dos Vales do
Jequitinhonha e Mucuri

Dedicatória

Dedico este trabalho aos meus amados e queridos Pai Rogério, Mãe Joana, Irmã Jaqueline, Namorada Adriana e amigos.

Agradecimentos

Agradeço a minha família, a minha namorada, aos meus amigos de longa data, aos amigos que fiz durante minha graduação, aos irmãos de república e aos mestres que contribuíram para que eu desse mais esse passo na caminhada da vida.

Resumo

As tecnologias de imagens 3D (três-dimensões) se tornaram um forte ramo de investimento por parte da indústria e do comércio. É fácil notar a sua difusão e aplicações, em filmes, propagandas produtos industrializados, sistemas de apoio à decisão, softwares de simulação, ferramentas CAD, etc.

Os simuladores baseados em tecnologias de visualização 3D representam uma boa parcela deste mercado. Conforme podem ser notados os investimentos na área são fortes e de alto índice de retorno. Vislumbrando estes dados, foi proposta uma pesquisa na Universidade Federal dos Vales do Jequitinhonha e Mucuri para identificar algum setor que usasse destas tecnologias. E através desta pesquisa foi descoberto o Centro Técnico de Formação de Operadores de Máquinas Florestais (CTFlor) que fornecia ao corpo acadêmico cursos de operadores de máquinas florestais, e dentre os cursos oferecidos, havia um, que utilizava da tecnologia de simulação em 3D, para ensinar aos alunos do curso o manuseio de uma colheitadeira de eucalipto.

Diante disso foi proposto o trabalho de pesquisar e desenvolver um simulador que atendesse as expectativas desse setor, utilizando a API de programação de gráficos em 2D e 3D *OpenGL* integrada com o *framework* de desenvolvimento em C++ *QT/QTCreator*. Esse trabalho possibilitou a compreensão da API e foi possível constatar sua eficiência quando utilizada para o desenvolvimento de aplicações de simulação. Também foi feita uma rápida análise de todo o universo que cercou o desenvolvimento desta aplicação, como computação gráfica, a API *OpenGL*, o *QT/QTCreator*, a metodologia de desenvolvimento *eXtreme Programming* e os conceitos de classe e objetos.

Basicamente o trabalho lida com o desenvolvimento de uma aplicação (software) capaz de trazer um novo ambiente de simulação de florestas, que fosse melhor ou mais baratos que os existentes no mercado, acrescentando ao universo acadêmico o estudo e pesquisa das ferramentas *OpenGL* e *QT/QTCreator*, aplicando as mesmas no desenvolvimento desta aplicação. Este trabalho possibilitará a abertura de uma nova linha de pesquisa na área da computação dentro da Universidade Federal dos Vales do Jequitinhonha e Mucuri.

Abstract

The technologies of 3D (three-dimensional) became a strong business investment by industry and commerce. It is easy to see the dissemination and applications, movies, advertisements industrialized products, decision support systems, simulation software, CAD tools, etc.

Simulators based on 3D display technologies, represent a good portion this market. As can be noted investments in the area are strong and high-index return. Glimpsing these data, we decided to search the Federal University of Valleys Jequitinhonha and Mucuri any sector that uses this technology. And in our research, we find that the Technical Centre for Training of Forestry Machine Operators (CTFlor) provided the body and other citizens academic courses machinists forest(as can be noted by the name of the technical center), and among the courses offered, there was one that used the technology of 3D simulation to teach students of the handling a combine eucalyptus.

Given this proposed work of researching and developing a simulator that meets expectations of this sector, using the API programming graphics in 2D and 3D OpenGL integrated with the development framework for C + + Qt / QtCreator. Through this work seek to deepen and understand the API to verify its efficiency when used for many simulation applications. We also did a quick scan of the entire universe that surrounded the development of this application, such as computer graphics, OpenGL API, the QT / QtCreator, the eXtreme Programming development methodology and concepts of class and objects.

Basically the work deals with the development of an application (software) able to bring a new simulation environment of forests, which would be better or cheaper than the on the market, adding to the academic study and research tools textit OpenGL and textit QT / QtCreator, applying the same in developing this application. This work will enable the opening of a new line of research in computing within the Federal University of the Valleys Jequitinhonha and Mucuri.

Sumário

Lista de Figuras

Lista de Tabelas

Introdução

As tecnologias de imagens 3D (três-dimensões) se tornaram um forte ramo de investimento por parte da indústria e do comércio. É fácil notar a sua difusão e aplicações, em filmes, propagandas produtos industrializados, sistemas de apoio à decisão, softwares de simulação, ferramentas CAD, etc.

Por qual motivo? Responder está pergunta não é difícil, quando é feita a analise da infinidade de recursos que essas tecnologias incorporam. Elas permitem ao usuário visualizar aspectos em uma cena, que, se visualizada sem tecnologias 3D iria esconder diversos detalhes e material que poderia ser importante para análises e conclusões. Além disso, elas permitem reproduzir com fidelidade de detalhes lugares, eventos, objetos, etc. Pesquisando informalmente na mídia é possível concluir que essa área está em constante crescimento e analisando a infinidade de elementos do cotidiano que são compostos por essa ferramenta, filmes, desenhos, simuladores, jogos, arte, tecnologias, etc. Analisando, especificamente a área das tecnologias é possível constatar o enorme crescimento desde a década de 70; uma simples investigação do quantitativo de linguagens e programas que a utilizam já é suficiente para constatar o fato. Os principais programas do mercado, atualmente, que lidam com o desenvolvimento de visualizações em 3D são segundo Falcão, Machado e Costa (2010, p. 56): MicroStation, ACIS (kernel geométrico), Softimage XSI, After Effects, Autocad, 3D Studio MAX, Blender, Cinema 4D, GIMP, Inkscape, POV-Ray, Rhino3d, Scribus, Silo (software), Wings 3D, Hexagon, Maya, Photoshop, Illustrator, Inkscape, CorelDraw, XnView, Lightwave , etc.

Os simuladores 3D representam uma boa parcela deste mercado. Conforme pode ser notado, os investimentos na área são fortes e de alto índice de retorno. Vislumbrando estes dados, foi proposta uma pesquisa na Universidade Federal dos Vales do Jequitinhonha e Mucuri para identificar algum setor que utilizasse destas tecnologias. E através desta pesquisa foi descoberto o Centro Técnico de Formação de Operadores de Máquinas Florestais (CTFlor) que fornecia ao corpo acadêmico cursos de operadores de máquinas florestais, e dentre os cursos oferecidos, havia um, que utilizava da tecnologia de simulação em 3D, para ensinar aos alunos do curso o manuseio de uma colheitadeira de eucalipto. Um simulador que fosse condizente com realidade atual das tecnologias disponíveis, que atendesse as expectativas do CTFlor e também atendesse as necessidades e exigências do mercado mostrou-se exorbitantemente caro. Basicamente as

necessidades que um simulador deste deveria atender eram:

- Simuladores mais realistas;
- Que permitissem aos usuários navegar por cenários;
- Onde os cenários possuíssem um relevos;
- E os relevos fossem editáveis;
- Bem como os parâmetros para dimensionar a floresta;

Podem ser citados como dois exemplos claros do que se trata a pesquisa dois simuladores atualmente no mercado, o **Harvester Personal Simulator** e o **Simulador John Deere de Harvester e Forwarder**. O primeiro não é demasiadamente caro (Atualmente custa USD$4,000, em reais R$8,500 aproximadamente. Dado disponível em `http://www.simlog.com/personal-harvester.html`), porem é limitado no quis diz respeito aos itens citados anteriormente, na realidade quando é feita a análise somente dos itens acima, ele não emprega nenhum na sua execução. O segundo é muito avançado mas, exorbitantemente caro (Aproximadamente R$300.000,00. Dado informal), ele atende a todos os requisitos a ainda mais uma gama de outros, mas o investimento é muito alto para pequenas empresas ou pesquisadores que desejam pesquisar sobre esse tipo de ferramenta.

As análises feitas neste trabalho tratam única e exclusivamente do ambiente (relevo e árvores) da simulação. Não é observado neste trabalho os demais componentes envolvidos nos simuladores.

A proposta consiste no desenvolvimento de um "Ambiente de Simulação de uma Floresta Tridimensional" que seja capaz de simular a conceituação inicial do ambiente florestal, de forma que seja possível ao usuário editar alguns parâmetros do ambiente como relevo e elementos da floresta (árvores). O parâmetro relevo permitirá alterar a forma do terreno de maneira a produzir a ideia de altura, desnível e ondulações de um relevo real, também será dada ao usuário a possibilidade de editar os elementos da floresta (árvores), no seu contexto de altura, diâmetro, densidade por unidade de área, espaçamento entre elementos, altura do tronco e da copa, etc. Na aplicação será possível, conforme um dos requisitos, navegar pelo ambiente simulado ampliando a visão da simulação bem como os horizontes de entendimento que ela pode prover. Não serão abordados neste trabalho, um mais ambiente realista contendo colisões, interceptações ou qualquer fenômeno físico, nem definições avançadas de textura, iluminação, sombra, ou qualquer efeito gráfico avançado. Este trabalho foi focado na concepção inicial da aplicação e no objetivo de gerar a visualização básica dos elementos propostos.

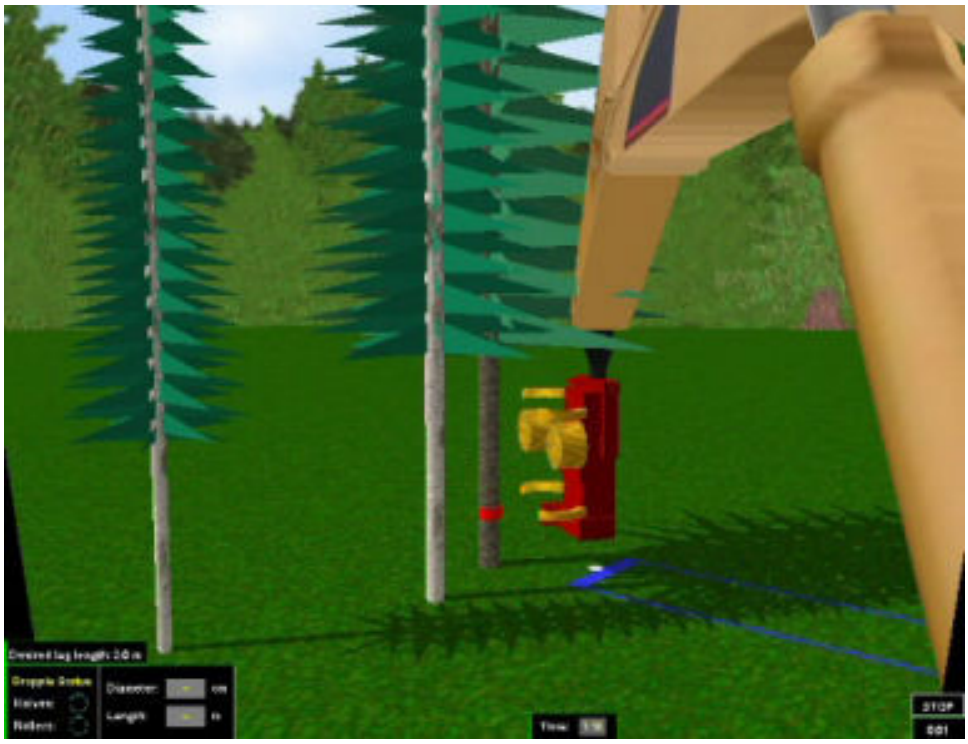

Figura 1: Cenário de interação do simulador “Harvester Personal Simulator”

Figura 2: Cenário de interação do simulador “John Deere”

No capítulo 1 será definido o conceito de computação gráfica a sua história e alguns termos fundamentais para o entendimento desta ferramenta computacional.

O capítulo 2 contém a definição de uma das duas ferramentas utilizadas. Neste capítulo é apresentado sucintamente o que é a OpenGL, como surgiu, seu histórico de versões e também uma breve descrição sobre o seu funcionamento e sua utilização. Este capítulo ainda discorre rapidamente como essa ferramenta foi incorporada ao trabalho; seu detalhamento completo de uso no trabalho estará exposto no capítulo 4.

O capítulo 3 refere-se a segunda ferramenta base utilizada o QtCreator e a linguagem Qt. Nele contem um breve relato sobre o framework Qt, e também sobre a IDE Qt Creator. É feito o relato de como a linguagem e a IDE surgiram e qual o proposito para seu surgimento. Doi detalhado um pouco do seu o seu poder de processamento, seu dinamismo e flexibilidade. Assim, no capítulo 2, é apresentada uma breve introdução da utilização da ferramenta no trabalho, que será explicada em detalhes da sua utilização mais adiante no capítulo 4.

No capítulo 4 está descrito como foi executado o trabalho, a metodologia usada (XP - eXtreme Programming) e como foram, em detalhes, incorporadas e utilizadas as ferramentas OpenGL e Qt/QtCreator no mesmo. Foi relatado também a união dessas ferramentas para o proposito principal da pesquisa, produzir um "Ambiente de Simulação de uma Floresta Tridimensional"; nele é possível vislumbrar como as duas ferramentas se encaixam de forma a produzir o efeito desejado. Além disso, é feita a descrição do programa propriamente dito, e apresentando-se, o modo como foi implementada e foram trabalhados os códigos. Não há detalhamento do código inteiro do programa neste capítulo, somente os pontos relevantes ao entendimento do seu funcionamento. Os trechos de códigos apresentados neste capítulo são fundamentais para que haja o entendimento da lógica utilizada no programa.

Por fim, na Conclusão são apresentados testes e resultados preliminares da aplicação e da sua execução, além de apresentar propostas para trabalhos futuros.

1 Computação Gráfica

"A Computação Gráfica (CG) é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a geração (síntese) de imagens através do computador" (MANSSOUR; COHEN, 2006).

Segundo Enciclopédia-Encarta (apud SANTOS et al., 2003, p. 1334) ela é responsável por gerar visualizações "bi e tridimensionais" que podem ser usadas na pesquisa científica, artes, indústria, e no comércio. Através das Interfaces Gráficas de Usuário (GUI) foi possível facilitar a utilização dos computadores, pois, elas permitem aos usuários selecionar imagens para executar ordens, não havendo mais a necessidade de decorar comandos de texto para executar suas tarefas.

1.1 Áreas da computação gráfica

Através do processamento de imagens e interfaces gráficas, a utilização dos computadores se tornaram muito mais simples. O que antes era tido como tedioso e pouco eficiente (ficar diante de uma tela preta, tendo de passar comando, e sempre visualizar textos), passou a ser fácil, além de ter tido ótima aceitação por parte dos usuários. Segundo Santos et al. (2003), a computação gráfica pode ser dividida em quatro grandes áreas: animação, processamento de imagens tridimensionais, síntese de imagens foto realística e realidade virtual.

1.1.1 Animação

É a área da Computação Gráfica responsável por simular fenômenos físicos relacionados a movimentação e deformação de corpos. Através de movimentos de câmera no cenário virtual gera uma sequência de visualizações, que chamada de animação computadorizada. Modelando-se geometricamente os objetos que compõe a cena é possível, através de simulações e movimentações de câmeras e dos objetos, gerar imagens da cena proposta (PARENT, 2001 apud SANTOS et al., 2003, p. 1334-1335).

1.1.2 Processamento de Imagens Tridimensionais

Segundo Santos et al. (2003) seria a construção de imagens em três dimensões por parte do computador. Ele deixa bem claro que, em suas análises na literatura, muitos autores citam que a síntese (elaboração ou desenvolvimento) e a análise (Estudo e pesquisa de fenômenos diversos com o apoio da visualização de imagens tridimensionais) de imagens seguiram caminhos diferente na evolução tecnológica e mesmo tendo pontos em comum, existe pouca troca de informações entre elas.

1.1.3 Síntese de Imagens Fotorealística

Ramo da Computação Gráfica que gera imagens de alta qualidade através de algoritmos, com nível de realidade idêntico ou muito próximo da imagem original (real). Suas principais aplicações são: arquitetura, design industrial e entretenimento, etc (SANTOS et al., 2003).

1.1.4 Realidade Virtual

Segundo Foley et al. (apud SANTOS et al., 2003, p. 1340) é definida como um conjunto de métodos e tecnologias relacionadas com a imersão do usuário em ambientes com aparência realística e comportamento e interação simulados pelo computador.

1.1.5 Interfaces gráficas de usuário

As GUI's (Interfaces Gráficas de Usuários) são uma das peças fundamentais na utilização de um computador no contexto atual. Conforme mostrado por PARC (2012), em 1975 a XEROX foi primeira a desenvolver uma GUI, que continha ícones de usuário, pop-ups menus e janelas sobrepostas que podiam ser controlados facilmente usando uma técnica de ponto-e-clique (mouse). O famoso GUI (ou infame) foi responsável pelo desenvolvimento de todas as interfaces de computadores pessoais subsequentes.

Para Galitz (2007) a interface de usuário é a parte mais importante do sistema computacional, pois ela é a visão do sistema para a maioria dos usuários. É a imersão do usuário no ambiente computacional de forma que ele possa ver, ouvir e "tocar" os elemento ali expostos. Ainda segundo Galitz (2007) seu objetivo é simples "tornar o trabalho com o computador fácil, produtivo e agradável".

1.2 Renderização

Processo no qual o computador cria, gera, ou expõe imagens e modelos de objetos e cenários. Os objetos e cenários são feitos a partir de geometrias primitivas: "pontos, linhas e polígonos" (SHREINER, 2009). Dá-se a imagem resultante desses processos o nome de renderização, que por sua vez, constitui-se de pixels desenhados na tela do computador. Por pixel tem-se o conceito de menor elemento visível de um hardware que pode ser exposto na tela do computador

2 *OpenGL*

OpenGL (Open Graphics Library) (SHREINER, 2009) é uma Application programming interface (API) de código livre, usada para desenvolvimento aplicações que utilizam computação gráfica. Ela permite ao seu utilizador desenvolver desde uma simples imagem em duas dimensões, até um filme em três dimensões, contendo diversos efeitos especiais e animações com efeitos muito realistas.

> Por "software livre" ou software de código livre, devemos entender aquele software que respeita a liberdade e senso de comunidade dos usuários. A grosso modo, os usuários possuem a liberdade de executar, copiar, distribuir, estudar, mudar e melhorar o software. Com essas liberdades, os usuários (tanto individualmente quanto coletivamente) controlam o programa e o que ele faz por eles (GNU, 2012).

Pode ser considerado atualmente como um dos principais ambiente de desenvolvimento de aplicações interativas portáteis de gráficos 2D e 3D (incluem-se nesta escala segundo Clua e Bittencourt (2005) DirectX, OpenAL, Java 3D, Unity 3D, Google O3D, etc).

2.1 Versões

Segundo OpenGL (2012) inicialmente a OpenGL era semelhante a IrisGL (plataforma de processamento de gráfico 3D que deu origem a OpenGL). Os autores das especificações da OpenGL Mark Segal e Kurt Akeley tentaram formalizar uma API de processamento gráfico que fosse útil e livre para todos utilizarem. Tendo isto em mente eles produziram varias implementações diferentes para essa API, e todas sempre adotando o mesmo critério, deveriam possuir licença livre para utilização de qualquer usuário. Com isso eles não utilizaram licenças como SGI ou 3rd Party (licenças de código proprietário). As versões oficiais da OpenGL lançados até hoje são 1.0, 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 4.1, 4.2,4.3 (OPENGL, 2012).

Nos sub-tópicos abaixo é feita uma breve descrição sobre as versões da OpenGL, baseadas em OpenGL (2012).

2.1.1 OpenGL 1.0

Como a versão 1.0 da OpenGL foi a primeira estável a ser lançada é fácil pensar que ela não atendia a toda a gama de utilidade a qual o conceito OpenGL foi proposto. Nesta versão foram omitidos objetos para processamento de texturas, materiais, luzes e ambientes de texturizados. Mas os autores tinham um motivo para isso, os desenvolvedores defendiam o conceito de listas de exibição, que através de mudanças incrementais seriam capazes de cobrir toda essa gama de objetos. Logo, se existisse uma lista como essa os programadores seriam capazes de exibir todos os objetos que não foram abordados, ficando a cargo do programador definir sua própria lista a partir de funções primitivas. Esta ainda é a filosofia básica da OpenGL.

2.1.2 OpenGL 1.1

Na versão 1.1 da OpenGL não foram apresentadas muitas alterações significativas em relação a versão 1.0, a única mudança significativa que mudou bastante o conceito da OpenGL foi a incorporação de um elemento para tratar texturas, que inicialmente havia sido descartado da versão 1.0. Esse elemento era o *glBindTexture* (OPENGL, 2012).

2.1.3 OpenGL 2.0

A versão 2.0 corrigiu muitos erros e incrementou diversas novas funcionalidades. A que teve maior peso e contribuição para o estado atual da arte da OpenGL foi o conceito de Shading Language, também conhecido como GLSL (OPENGL, 2012), isso incorporou a OpenGL o conceito de sombreamento. Com isso foi possível caracterizar superfícies, volumes e objetos no espaço.

2.1.4 OpenGL 3.0, 3.1 e 3.2

Na versão 3.0 as alterações mais significativas, além da remoção de erros (como as demais anteriores), foi a adição do conceito de depreciação, marcando algumas características obsoletas para remoção em versões posteriores. Como proposto na versão 3.0 a versão 3.1 teve como evolução significativa a remoção das características obsoletas. A Versão 3.2 criou o conceito de compatibilidade de contexto com outras versões OpenGL e também a noção de núcleo da API, sendo assim havia agora um ponto central de referência para as funcionalidades.

2.1.5 OpenGL 4.0 e versões atuais

A versão atual da OpenGL se tornou, por meio das versões anteriores, uma API de programação muito estável, rica em recursos para a programação e que mantem até hoje seus conceitos básicos de, liberdade de utilização, simplicidade e objetividade, não incorporando elementos específicos de renderização deixando a cargo do programador realizar tais feitos a sua maneira.

2.2 Outros conceitos da OpenGL

A OpenGL foi projetada simplificadamente para que possa ser implementada independentemente de interface de hardware, assim sendo suportada em qualquer interface desejada ou qualquer arquitetura de processamento de vídeo. Ela também foi projetada para ser independente do Sistema de Janelas (GUI), sendo assim não existem comandos puramente OpenGL que diretamente processem janelas ou similares. Essa propriedade de ser multiplataforma foi a responsável por tornar a OpenGL tão popular e difundida. Ela é executado em todos os sistemas operacionais importantes, incluindo Mac OS, OS/2, UNIX, todas versões Windows a partir do 95, Linux, OPENSTEP, e BeOS, e também trabalha com todos os grandes Sistema de Janelas, incluindo Win32 e MacOS. Porém, a maioria dos usuários constroem suas aplicações no Sistema de Janelas X-Window System.

Partindo de princípios simples é possível criar uma infinidade de formas, texturas, cores e iluminações. Tudo graças a mecanismos poderosos como, máquinas de estados, conforme Martin (1998) uma máquina de estado finito é uma abstração matemática usada para elaborar de algoritmos avançados. Em termos simples, ela recebe uma série de comandos que, farão com que a mesma mude para um estado diferente, ou não. Cada estado especifica quais são os estados que podem ser alcançados por ele dado um determinado comando, sobrecarga de funções e outros recursos implementados pela API e disponíveis na linguagem C, linguagem base da OpenGL segundo Bicho et al. (2002).

Ainda segundo Bicho et al. (2002) normalmente, utiliza-se a linguagem C++ para programação com Open Inventor, Java para programação com Java 3D e a linguagem C para programação de aplicações com OpenGL. No entanto, existem implementações destas bibliotecas e de suas APIs em outras linguagens de programação, como Delphi, Ada e Modula-3.

A OpenGL também conta com uma vasta gama de documentação, é possível encontrar na Internet ou em livrarias uma infinidade de manuais e livros de referência que possibilitam a

qualquer pessoa aprender a linguagem. Os próprios desenvolvedores da API, em seu site `http://www.opengl.org`, disponibilizam uma lista que contém os manuais de referência, facilitando a filtragem de material que seja de qualidade por parte dos programadores que trabalham com a API.

2.3 Histórico

Segundo SGI (2012), no fim da década de 80 uma empresa da cidade de Sunnyvale/Califórnia, a Silicon Graphics Inc (SGI) (SGI, 2012), produziu um padrão para uma API de programação gráfica chamado IRISGL. Esse padrão chamou muita atenção da indústria, sendo considerado o estado da arte de uma API gráfica (SGI, 2012). Ela começou a se tornou um padrão nas indústrias da época, pela sua facilidade de uso, mas ainda sofria forte resistência por parte de grandes corporações como a Sun Microsystems e a IBM, que ainda eram capazes de fabricar hardware adotando o padrão PHIGS (sigla de Programmer's Hierarchical, Graphics System (SGI, 2012)), padrão desenvolvido anteriormente e que havia sido amplamente difundido nas grande empresas de tecnologia. Como estratégia para difundir de vez a sua tecnologia a SGI pensou em manobra mercadológica, tornar seu padrão público, para que todo e qualquer fabricante de hardware pudesse usá-lo. Mas para isso era necessário rever todo o código-fonte do seu padrão IRISGL, pois possuía muitos códigos proprietários, impedindo-a de torna-lo público. Além desse detalhe, A Silicon também considerava que a IRISGL possuía muitas questões que não estavam diretamente ligadas ao processamento de gráficos 2D e 3D, "como gerenciamento de janelas, teclado e mouse" (SGI, 2012). Mesmo tendo esses empecilhos, a empresa achou interessante manter seus antigos clientes comprando seus hardwares, e também quis considerar os novos cliente que necessitavam do padrão público, e como resultado disso, o padrão OpenGL foi criado em 1992.

> "Desde 1992, o padrão é mantido pelo ARB (Architecture Review Board), um conselho formado por empresas como a 3DLab, ATI, Dell, Evans & Sutherland, HP, IBM, Intel, Matrox, NVIDIA, Sun e, logicamente, a Silicon Graphics. O papel desse conselho é manter a especificação e indicar quais recursos serão adicionados a cada versão. A versão mais atual do OpenGL até hoje é a 4.2 (SGI, 2012).

2.4 Funcionamento

Tudo em OpenGL parte dos princípios de funcionamento de uma Máquina de Estados, e de maneira bem especifica é dito ao computador suas ações a partir de um determinado ponto, ou função. Por exemplo, se quiser que algo seja pintado (mostrado) de azul na tela, teremos que usar uma diretiva (função em c) para indicar ao OpenGL que o que virá após este ponto será pintado de azul, assim a máquina de estados mantem a cor azul para tudo que é plotado tudo que vem após a diretiva, ou até que seja indicada outra cor, ou limpamos o estado da máquina. A função que executa tal funcionalidade é:

```
glColor*(argumento);
```

As seguintes diretivas definem estados na Máquina de Estados do OpenGL:

```
glClearColor();
glMatrixMode();
glEnable()/glDisable();
glPolygonMode();
glColor*();
```

A arquitetura da API do OpenGL segue a seguinte estrutura:

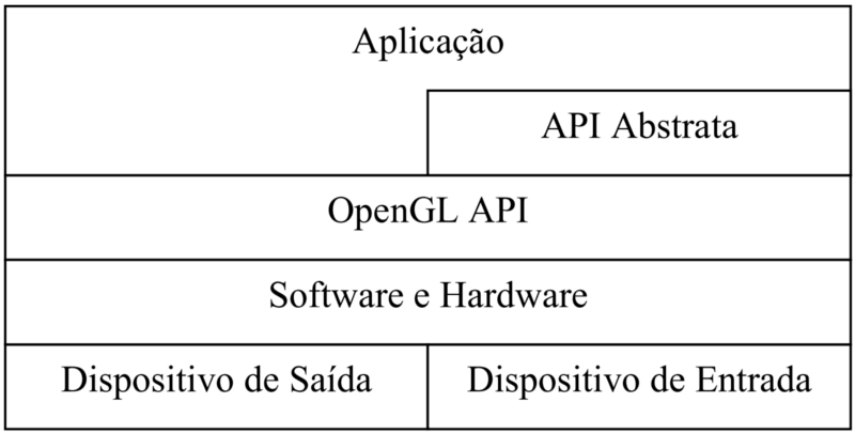

Figura 2.1: Estrututa da API OpenGL.

Aplicação define os códigos do usuário em OpenGL. Estes códigos como mostrado anteriormente, serão responsáveis por descrever os cenários e os objetos que compõem a cena. Através da diretiva:

```
glBegin(argumento);
  vertices do poligono
glEnd();
```

(onde argumento pertence aos itens descritos na Tabela 3.1 e entre as duas funções haverão códigos descrevendo os vértices que compõem o objeto) será possível desenhar o que a imaginação do programador conseguir formular. Este trabalho é árduo e um pouco cansativo, pois pode-se ter objetos muito complexos que demandariam muito tempo e esforço mental por parte do programador. Para evitar esse desgaste foram feitas outras API's que abstraem diversas formas e objetos para o programador, de forma que ele se preocupe apenas com a lógica da interação dos objetos, e não, como eles são descritos. Conforme citado por Falcão, Machado e Costa (2010, p. 56) Blender, Maya, SketchUp, 3D Studio Max, AutoCAD, são algumas das API's que abstraem a OpenGL.

Após a camada da Aplicação temos a camada da API OpenGL, que será responsável por decodificar o que foi proposto na camada da Aplicação para controladores e códigos da linguagem C, isso quando o programador utiliza diretamente a API OpenGL, em caso contrário inclui-se mais um nível no diagrama, que será o nível o qual a plataforma de abstração da OpenGL utilizada pelo programador irá se comunicação entre com a API OpenGL.

E por fim, será feita a comunicação da linguagem C com os dispositivos de entrada ou saída do computador, que irão mostrar os objetos e cenários ou receber as informações do usuário, para que seja possível o mesmo interagir com o programa.

2.4.1 Estrutura

Por possui diversas forma de ser implementada variando de API para API, a OpenGL pode ter diferente estruturas básicas para um programa qualquer. Abaixo é mostrada a estrutura mais comum e utilizada (não usada neste trabalho) a biblioteca GLUT que, segundo Shreiner (2009, p. 17-18), amplia os horizontes da OpenGL, já que a OpenGL não possui recursos para processamento entrada de dados e nem diretivas para o processamento do Sistema de Janelas, sendo assim a GLUT irá tornar a OpenGL completa, no que diz respeito aos programas que os seus usuários podem criar:

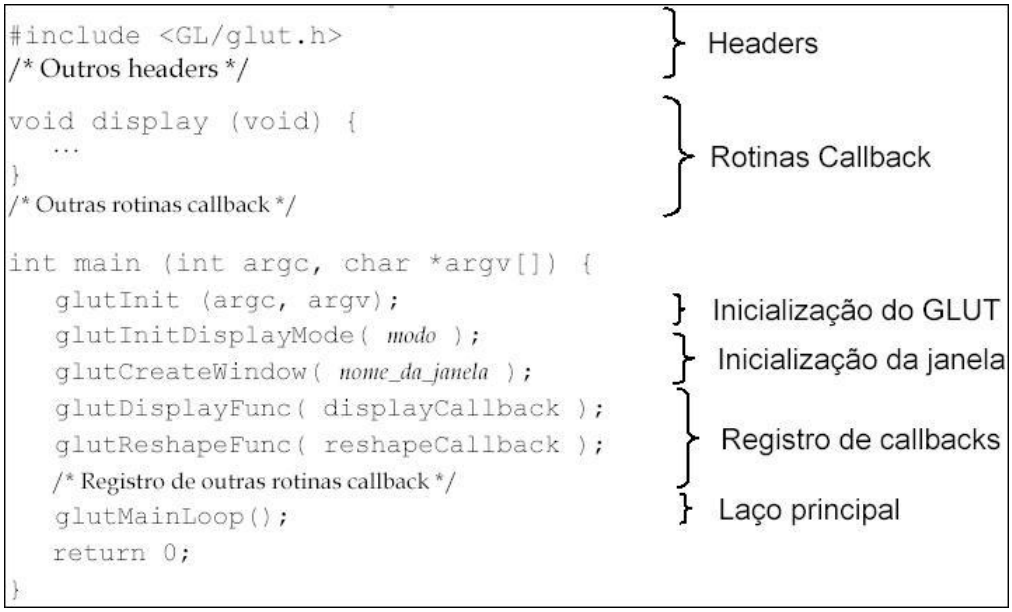

Figura 2.2: Estrutura básica de um programa OpenGL utilizando a GLUT.

Para este trabalho o Framework escolhido foi o Qt Creator (BLANCHETTE; SUMMERFIELD, 2008) que possui a sua própria forma de tratar a estrutura básica da OpenGL, com suas próprias funções básicas. Sendo assim ela deve ser seguida na criação de qualquer aplicação, desde a mais básica até as mais avançadas. Basicamente todos os programas feitos nele contem as seguintes funções:

```
initializeGL();
paintGL();
resizeGL(int width, int height);
```

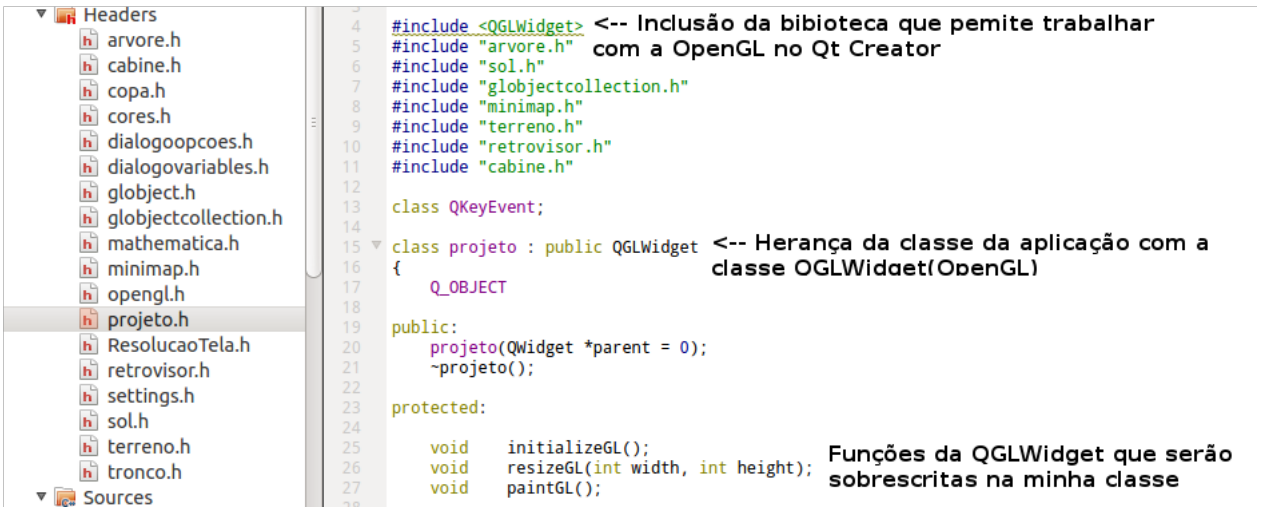

Figura 2.3: Estrutura básica de um programa OpenGL utilizando o QT Creator (Classe "projeto").

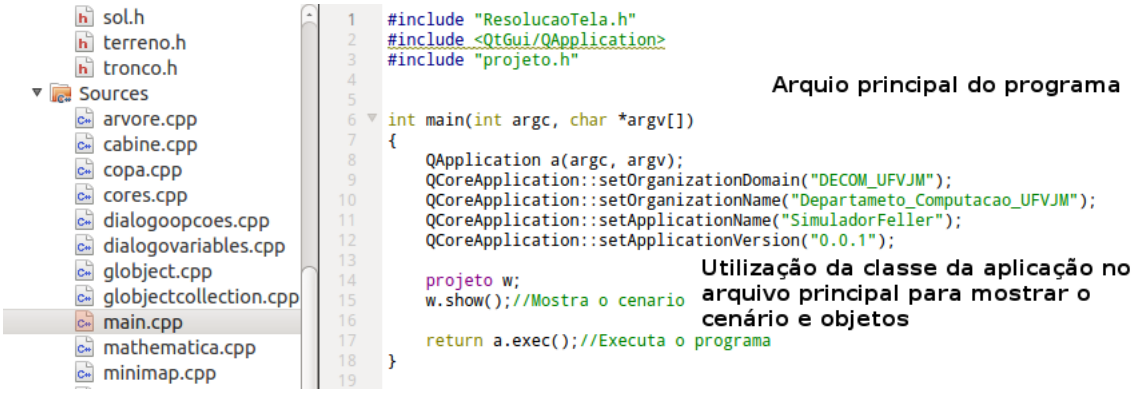

Figura 2.4: Estrutura básica de um programa OpenGL utilizando o QT Creator (Classe "main").

Na estrutura básica as funções *initializeGL( )* e *paintGL( )* são indispensáveis. Mesmo que somente façamos delas virtuais puras (pois não existe a obrigação de as sobrescrever, Blanchette e Summerfield (2008)), mas elas devem constar na aplicação. A função *resizeGL(int width,int height)* apesar de não ser indispensável deve ser usada sempre que, for relevante o comportamento da aplicação quando a janela é redimensionada. As funções citadas acima são responsáveis por:

resizeGL(int width, int height) Esta função, segundo Blanchette e Summerfield (2008) define como será o comportamento da aplicação quando a janela for redimensionada. Mantendo as proporções das figuras ou assumindo novas proporções. Deve ser usada, sempre que o comportamento da aplicação em relação ao tamanho da janela for relevante. Caso não for dito como lidar com o redimensionamento, a API assume a escala padrão que provavelmente irá distorcer os objetos e cenário.

initialize GL() Função nativa do QT/ OpenGL (normalmente sobrecarregada) Blanchette e Summerfield (2008) para indicar quais vão ser os seus estados iniciais da sua aplicação. Nela informamos como ocorrerá a renderização dos objetos, como será a o comportamento da iluminação, como será o comportamento das sobras, etc. Conforme dito não é necessário a sobrecarregar, mas se isto não for feito o programa será compilado com as configurações padrões para os itens citados acima. Mesmo em aplicações básicas pode não ser o mais adequado.

paintGL() Essa função, segundo Blanchette e Summerfield (2008) é responsável por fazer a "mágica" da OpenGL. Ela define o que e o como será pintado na tela do computador as imagens e objetos descritos na função. Em geral ela é sobrescrita, pois se não for, apenas será mostrado uma janela sem nenhum conteúdo. Nela colocamos diretivas como, *glTraslate*, glRotate** que serão responsáveis por fazer transformações geométricas nos

objetos da cena, e também a função *glBegin(*)* que desenhará os objetos (confirme citado anteriormente). As seguintes primitivas são usadas como parâmetro da função *glBegin(*)* para desenhar objetos.

2.5 Utilização no trabalho proposto

No trabalho proposto foi utilizada a API para processamento gráfico OpenGL em conjunto com a API para programação e abstração de orientação a objetos, QT Creator. A OpenGL foi escolhida por diversos motivos, como:

- Facilidade de se encontrar material de referência;
- Relativamente fácil de se aprender;
- Grande comunidade que contribui com códigos e material de referência;
- Simplicidade dos códigos;
- Estar licenciada segundo GNU/GPL, o que torna livre a utilização, e não implica em processos legais;
- Completa integração com a plataforma QT Creator;
- Aceitabilidade de mercado, e de hardware;
- Velocidade de processamento;
- Aplicações leves;

A comunidade que utiliza e apoia a OpenGL é muito grande, e por esse motivo existe maior suporte quanto a duvidas, correção de bug e a quantidade e qualidade dos materiais de referência. Esse seria um dos pontos mais críticos que motivou a escolha desta API, o outro é a velocidade do processamento. Em comparação informal com outras linguagens de processamento gráfico, como Java3D, a OpenGL aparentemente se sobressai, além de aparentar ser mais leve e mais rápida, conseguindo processar grande quantidade de formas, figuras, cenários, texturas e iluminações sem perde a agilidade e qualidade da aplicação.

3 Qt e QT Creator

> O Qt é um sistema modular de classes e ferramentas que torna fácil a escrita de pequenos módulos de código. Ele fornece uma substituição quase que completa para classes/tipos da STL, compila/roda em mais compiladores que os códigos escritos em C++ puro, e suporta vários dos mesmos recursos sem a necessidade de um compilador moderno. C++ é uma linguagem poderosa que suporta muitos estilos diferentes de programação. Os estilos de códigos usados na maioria dos programas em Qt, não são em "C++ puro". Como alternativa, é comum se ver uma combinação de macros e truques de pré-processamento para alcançar um nível maior na dinâmica da linguagem o que é muito comum em Java e Python (EZUST; EZUST, 2011).

"A Standard Template Library, ou STL, é uma biblioteca C++ de classes contêineres, algoritmos e iteradores, que fornece muitos dos algoritmos básicos e estruturas de dados de informática. A STL é uma biblioteca genérica, o que significa que seus componentes são fortemente parametrizados: quase todos os componentes do STL é um modelo" (SGI, 2012).

Conforme pode ser visto no paragrafo acima, a API Qt provem recursos ao programador que facilitam muito sua vida. Com a API foi possível fazer uma infinidade de programas conforme citado acima, além de, tornar simples a implementação de interfaces gráficas, por possuir ferramentas poderosas de edição e design o Qt Creator e seu subitem o Qt Design, ambas citadas logo abaixo.

> Qt Creator é um ambiente de desenvolvimento integrado (IDE - em inglês) multi-plataforma que, foi concebido para facilitar o desenvolvimento de aplicações em C++ que usam o Qt. Como é esperado, ele inclui um bom editor de códigos em C++ que provem um inteligente auto completador de códigos, ajuda sensível ao contexto usando o recurso de assistência do Qt, mensagens de erros/warnings durante a digitação e uma ferramenta de navegação rápida pelo código. Além do editor de textos, a IDE possui o Qt Design que, permite montar de forma visual os layouts e designs dos formulários feitos com ele com o Qt, sendo totalmente integrado ao Qt Creator. O Qt Creator possui ferramentas visuais de depuração e construção/manutenção de códigos, além de alguns geradores de códigos e recursos de navegação muito uteis (EZUST; EZUST, 2011).

A hierarquia de classes no Qt é bem definida e estruturada de forma facilitar o entendimento do desenvolvedor, o manual on-line que a IDE Qt Creatir dispõe torna ainda mais fácil navegar pelas classes e descobrir suas funcionalidades.A estrutura hierárquica de classes do Qt é a apresentada logo abaixo (note que o nome das classes é bem sugestivo a sua funcionalidade):

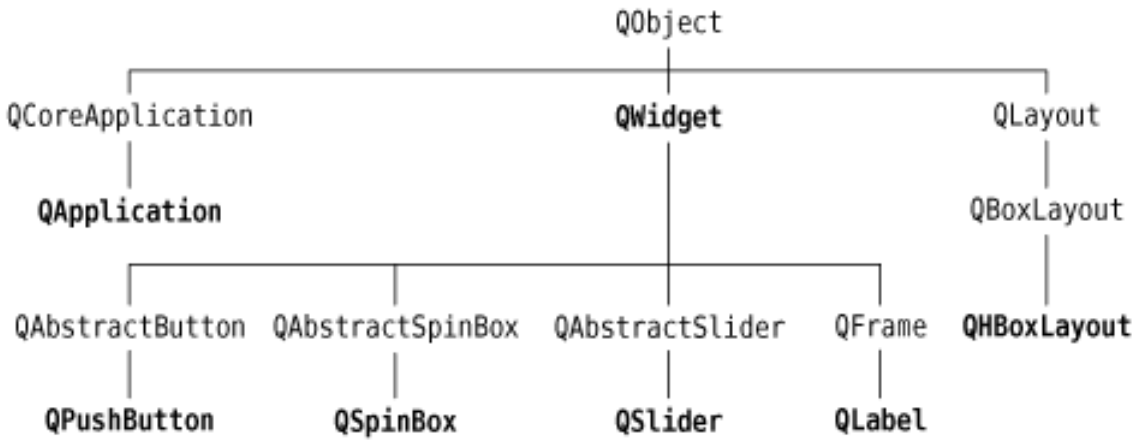

Figura 3.1: Hierarquia de classes do Qt.

3.1 Histórico

Segundo Blanchette e Summerfield (2008), a primeira versão publica do framework Qt foi oficialmente lançada em 1995. Inicialmente era desenvolvida por Haavard Nord (Trolltech's CEO) e Eirik Chambe-Eng (Trolltech's Chief Troll). Quando trabalharam juntos no verão de 1990 em uma empresa de desenvolvimento de softwares, Haavard e Eirik usavam a linguagem C++ e precisavam desenvolver uma aplicação de banco de dados para imagens de ultrasom. O sistema que eles tinham de desenvolver deveria possuir interfaces gráficas nos sistemas operacionais Unix, Macintosh, e Windows. Mais tarde naquele verão Haavard indagou a Eirik: "Precisamos de um sistema multi-plataforma orientado a objetos em C++", e o resultado disso é que eles passaram a pensar nessa hipótese e formular soluções para esse framework GUI multi-plataforma.

Foi em 1991 que Haavard começou a esboçar e escrever as classes, juntamente com Eirik, que viriam a se tornar o framework Qt. Nos anos seguintes Eirik trabalhou na ideia de "Sinal e Slot" (Em linhas gerais significa que, um sinal é emitido pelo usuário da aplicação e uma função (slot) irá atender a esse sinal (BLANCHETTE; SUMMERFIELD, 2008)), um simples e poderoso paradigma de programação para interfaces gráficas, Haavard gostou da ideia e a incorporou ao projeto.

Logo após eles decidiram que seria proveitoso se unirem de vez e criar, o que segundo eles

seria "A melhor API de programação GUI em C++ do mundo", e em 1994 os dois decidiram de vez criar uma empresa para lançar o produto (inacabado) no mercado (que já estava bem estabilizado). A empresa foi constituída em 4 de março de 1994, originalmente como Quasar Technologies,então como Troll Tech, e hoje como Trolltech.

A letra "Q" foi escolhida para ser o prefixo das classes no Qt, porque tinha visual bonito na fonte do editor de textos Emacs do desenvolvedor Haavard.

3.1.1 Versões

Em 20 de Maio de 1995 a versão 0.9 do Qt foi descarregada no servidor `sunsite.unc.edu`, e seis dias mais tarde foi anunciada na *comp.os.linux.announce*. Então está se tornou a primeira versão publica do Qt. O Qt estava disponível sob duas licenças: Uma licença comercial era exigida caso fosse utilizado para criar aplicações comerciais e outra de software livre estava disponível para desenvolvedores de código livre.

No final de Maio de 1996 foi lançada a versão 0.97 do Qt. Em 24 de Setembro do mesmo ano foi lançada versão 1.0, e no fim do ano foi anunciada e lançada a versão 1.1, oito cliente cada de países diferentes haviam comprado 18 licenças cada um. Mesmo ano que viu a fundação do projeto KDE (interface gráfica para sistemas Linux) liderado por Matthias Ettrich.

O Qt 1.2 foi lançado em Abril de 1997, foi quando Matthias resolveu usar o Qt para implementar o projeto KDE, e assim ajudar de vez ao Qt a ter um suporte a interfaces gráficas para aplicações GUI C++ para Linux.

Oficialmente a versão 2.0 do Qt foi lançada em Junho de 1999. Qt 2 possuía uma nova licença de código livre, a Q Public License (QPL) que cumpria as definições da Open Source.

Qt 3.0 foi lançado em 2001. Qt 3 possuía 42 novas classes e seu código ultrapassado 500.000 linhas. Ele foi um grande passo à frente do Qt 2, incluindo considerável melhoria de alocação de memória e suporte a Unicode, texto completamente novo, visualizador e editor de widget, e uma Perl-class para criar expressões regulares.

Finalmente no verão de 2005 o Qt 4.0 foi lançado. Contando com mais de 500 classes e mais de 9000 funções. Ele inclui-a um conjunto novo e eficiente de templates, contêineres, funcionalidade avançada de modelo/visão e um rápido e flexível framework de pinturas em 2D.

3.2 Utilização no trabalho

No trabalho foi utilizado o Qt Creator como IDE de programação e o framework Qt como API para processamento do sistema de janelas. Também foi utilizada a API Qt para fornecer o paradigma de Orientação- Objeto a aplicação (que não está disponível na OpenGL pura, por ser implementada em C), tornando mais fácil e prático o trabalho.

Essa ferramenta foi escolhida por se tratar de um framework (Qt) muito conceituado e premiado, e por possuir nativamente suporte a OpenGL. A escolha também decorreu por trabalhar com o paradigma de "Sinal-Slot", que para o trabalho seria útil no que diz respeito a interação do usuário com a aplicação. Utilizou-se a IDE Qt Creator, por prover um poderoso editor de textos, que facilita o desenvolvimento e o torna mais ágil. Com suas ferramentas de produtividade como, auto completador, navegador de classes e pré-processamento de texto informando erros/*warnings*, a produtividade teve ganho considerável.

Outras ferramentas utilizadas providas pelo Qt Creator e pelo framework Qt serão abordadas no capitulo "Materiais e métodos".

4 A Aplicação

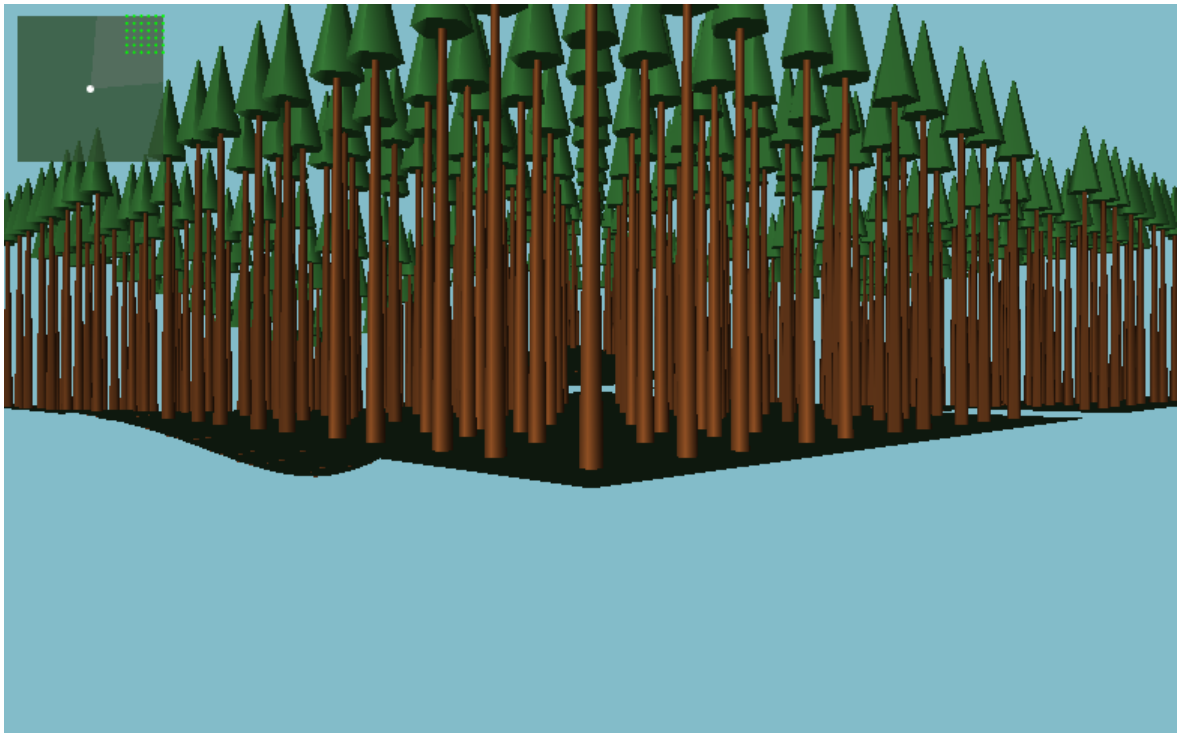

Figura 4.1: Visualização *Default* da aplicação.

Neste capítulo é apresentada a estrutura do programa. Nele é abordada mais profundamente a lógica usada no desenvolvimento do mesmo. Também é apresentado como foi realizada a união das duas ferramentas base (OpenGL e Qt/Qt Creator), para a criação da ferramenta, que iria formar o suporte necessário para o desenvolvimento da aplicação proposta pelo trabalho. Serão abordadas também, as estruturas básicas das classes que compõe a aplicação (não abordando detalhes do código e implementação, somente quando for necessário a compreensão da lógica da funcionalidade em questão). E por fim, serão apresentadas as bases da simulação, e algumas mudanças que podem ser realizadas para personalizar a simulação de forma a obter o resultado desejado.

4.1 Metodologia

No desenvolvimento deste trabalho foram utilizadas as ferramentas OpenGL e Qt/Qt Creator (conforme já apresentado nos Capítulos 2 e 3), para criar a ferramenta que, seria necessária para o desenvolvimento da proposta de trabalho. Para isso, foi aproveitado o poder de processamento gráfico 3D que a OpenGL proporciona aliado ao conceito de orientação a objetos que o framework Qt disponibiliza, tornando assim, a OpenGL ainda mais poderosa, pois seria possível abstrair de forma mais eficiente os elementos que compõem o cenário, como árvores e terreno.

Partindo da obtenção da ferramenta foi possível modelar a aplicação segundo os recursos que ele forneceria, e fazer esboços de como ficaria o conceito aplicado a ferramenta. No trabalho foi utilizada a metodologia XP (eXtreme Programming), que proporcionou uma forma de trabalho muito adequada a realidade da aplicação a ser desenvolvida, pouco foco na documentação, aprendizado rápido da ferramenta, reuniões curtas e objetivas e colaboração das partes interessadas ao invés de contratos. Assim, foi possível focar no aprendizado e desenvolvimento, agilizando a produção da aplicação e maximizando o tempo disponível.

4.1.1 eXtreme Programming

A metodologia eXtreme Programming (XP) (SOARES, 2004) foi desenvolvida no início dos anos 1990(SOARES, 2004 apud BECK, 1999, p. 23) por Kent Beck(BONA, 2002 apud WEEL, 2002, p. 23), faz parte das metodologias de desenvolvimento ágeis que " (...) dividem o desenvolvimento do software em iterações, buscando redução de riscos ao projeto. Ao final de cada iteração, uma versão (*release*) funcional do produto, embora restrita em funcionalidades, é liberada ao cliente." (BONA, 2002), criada com o intuito de acelerar o tempo de desenvolvimento focando no produto final, e não na documentação conforme Bona (2002) "as metodologias ágeis destacam aspectos humanos no desenvolvimento do projeto, promovendo interação na equipe de desenvolvimento e o relacionamento de cooperação com o cliente. Comunicação face-a-face é preferida à documentação compreensiva", claro que é possível fazer em paralelo ao desenvolvimento a documentação, mas este não é o foco do método. São pontos fundamentais das metodologias ágeis (SOARES, 2004):

- Indivíduos e interações ao invés de processos e ferramentas;
- Software executável ao invés de documentação;
- Colaboração do cliente ao invés de negociação de contratos;

- Respostas rápidas a mudanças ao invés de seguir planos;

Segundo Soares (2004), a metodologia XP consiste em uma metodologia ágil desenvolvida para pequenas e médias equipes de desenvolvimento de softwares, que tem como principal empecilho requisitos mal definidos ou vagos que tendem a modificar-se com muita frequência e rapidamente. Nele são necessários *Feedback* constante para melhorar a comunicação entre as equipes e também facilitar o levantamento de mudanças e novos requisitos, abordagem incremental de forma a partir do básico e a cada nova iteração acrescentar novas funcionalidades e o encorajamento da comunicação entre as partes envolvidas no projeto de forma a sempre haver comunicação clara e objetiva.

Segundo Vasco, Vithoft e Estante (2005), as iterações do XP costumam ser curtas, por isso o alto índice de versões (release) liberados para o cliente, assim é possível o cliente dar suas opiniões em pontos específicos do software, realimentando o processo de produção com um *feedback* daquela versão, que será usado pela equipe de desenvolvimento para corrigir e continuar o desenvolvimento. Seu objetivo é flexibilizar o processo de desenvolvimento do software, amortecendo o impacto de mudanças continuas no produto a ser desenvolvido, assim o próprio código é usado como indicar de progresso do projeto de desenvolvimento. Segundo Vasco, Vithoft e Estante (2005) o XP tem seu ciclo de vida dividido nas seis seguintes etapas:

1. **Exploração**: nessa fase o cliente escreve cartões de estórias, cada um contendo uma funcionalidade desejada para o primeiro *release*.
2. **Planejamento**: ocorre definição de prioridades entre as estórias junto com o cliente. Nesta etapa os programadores estimam o esforço e o cronograma para cada uma das estórias.
3. **Iterações para *Release***: nessa fase ocorrem diversas iterações até o primeiro *release* ser completado. Na primeira iteração é criado o sistema com toda a arquitetura, nas iterações seguintes serão adicionadas às funcionalidades de acordo com as prioridades estabelecidas.
4. **Validação para Produção (*Productionizing*)**: nessa fase são feitos testes extensivos e verificações para validação do software para ser utilizado em ambientes de produção.
5. **Manutenção**: após o primeiro *release* para produção, há novas edições do sistema com novas funcionalidades.
6. **Morte**: quando não há mais estórias a serem implementadas, quando o cliente está satisfeito com o código.

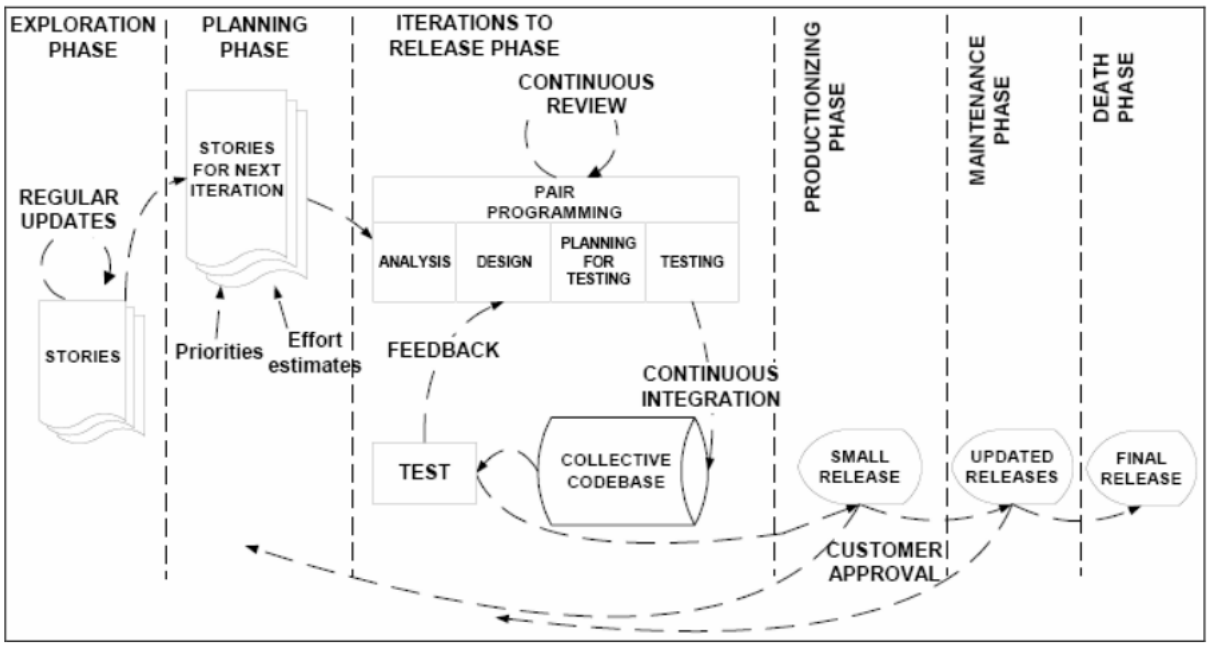

Figura 4.2: Fases do XP - Retirado de (VASCO; VITHOFT; ESTANTE, 2005).

4.1.2 Estrutura das classes

Segundo Eckel (2006) Orientação a objetos é um paradigma de programação, fundamentado no conceito de classes e objetos que abstraem para o computador os componentes do mundo real, fornecendo ferramentas para o programador representar elementos no espaço do problema. Assim, OO (orientação a objetos) permite descrever o problema mais próximo da realidade humana, em vez de descreve-lo em termos do computador onde o solução será executado, códigos de máquina. Abstração é a transformação de objetos reais em componentes computacionais, preservando seu conceito, sendo assim a abstração vai garantir que as principais características do objeto sejam mantidas no objeto abstraído. Um objeto é composto por características (variáveis), métodos ou comportamentos (funções) e interface (definição do objeto), e cada objeto pertence a uma classe. As classes são a descrição geral dos objetos. Pode-se pensar na classe como um molde do objeto, e o objeto como um elemento gerado por esse molde. Cada objeto se parece um pouco com um pequeno computador, tem um estados, e tem operações que você pode pedir a ele para executar. No entanto, esta não parece ser uma analogia ruim para objetos no mundo real, todos eles têm características e comportamentos. Além disso, existe o conceito de interface e encapsulamento de um objeto, que deixa visível somente elementos que são necessários para sua utilização, "escondendo" como os seus comportamentos são implementados, assim não é necessário saber como são feitos os comportamentos, é necessário saber apenas como ele age e quais são seus resultados.

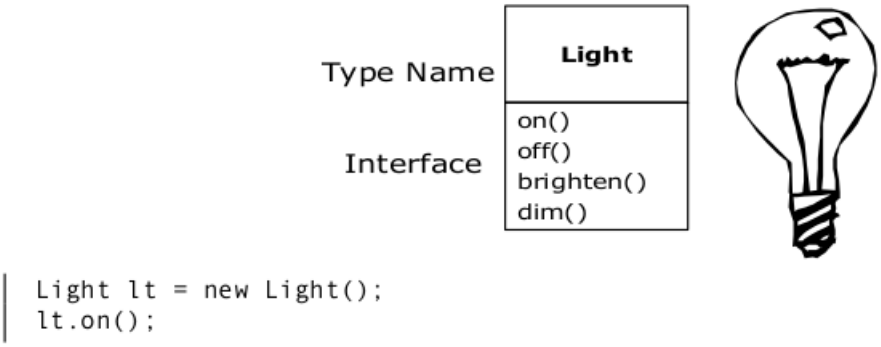

Figura 4.3: Exemplo de composição - Retirado de (ECKEL, 2006).

Ainda segundo Eckel (2006), existem mais alguns conceitos como "herança" e "composição". A diferença entre Herança e Composição é que a herança existe entre classes, sendo que uma classe herda características e comportamentos de outra classe, assim é possível criar classes derivando-as de outras, já composição lida também com as classes, porem, não existe a questão de herança e de características e comportamentos passados de uma classe apara outra, na realidade a composição só diz que uma classe possui em suas definições ou como parte de suas características outras classes.

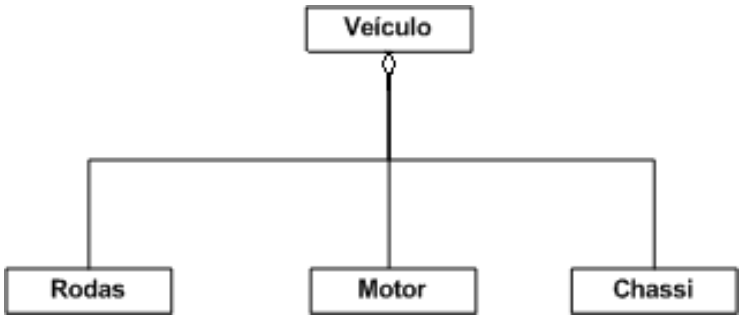

Figura 4.4: Exemplo de composição.

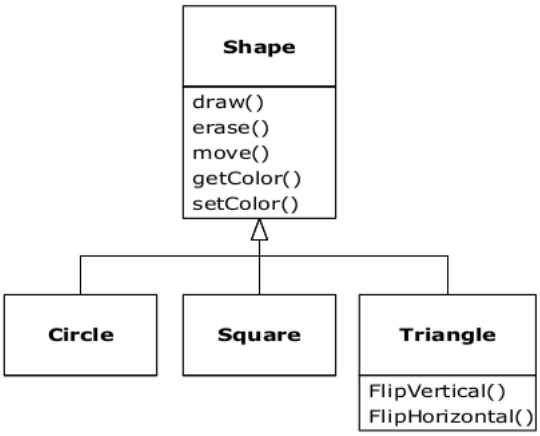

Figura 4.5: Exemplo de herança - Retirado de (ECKEL, 2006).

A OpenGL puramente não trabalha com o conceito de orientação a objetos, pois é implementada em C (BICHO et al., 2002), linguagem que não possui esse paradigma.

O trabalho para conseguir implementar de forma eficiente a aplicação seria muito grande, pois, não seria possível intercambiar e manipular os elementos (objetos) que compõem a cena facilmente.

4.1.2.1 As classes *glObject* e *glObjectCollection*

Foram elaboradas através do paradigma OO algumas classes fundamentais a aplicação como, *glObject*, *glObjectCollection*. A classe *glObject* tornava possível criar quais quer modelos de objetos de forma padrão e simples, sempre adotando a premissa que um objeto deve possuir um local no espaço indicado por coordenadas tridimensionais "x","y"e "z" tornando simples localizar um objeto no espaço de simulação, essa classe ainda possui alguns métodos puramente virtuais (devem ser implementados nas classes derivadas dela) que são básicas a todos os objetos derivados dela, *paint()*, *draw_xy()* e *clone()*, métodos que definem respectivamente como desenhar o objeto, como plotá-lo no ambiente tridimensional e como criar um clone a partir dele. Além da *glObject* foi criada também a *glObjectCollection*, que encapsulava uma coleção (lista) de objetos, tornando viável manipular vários objetos de um mesmo tipo ao mesmo tempo, por exemplos árvores. Na aplicação, basicamente todos as demais classes derivam, por meio de herança ou composição, de uma dessas duas classes como as classes "tronco" e "copa" que compõem a classe "árvore" e ambas são derivadas de glObject, bem como as classes "terreno" e "minimapa" que também derivam da glObject. O protótipo das classes *glObject* e *glObjectCollection* pode ser visualizado abaixo:

```
#ifndef GLOBJECT_H
#define GLOBJECT_H

class glObject
{
    //Coordenadas de posicionamento
        double coord_x;
        double coord_y;
        double coord_z;

    public:
        glObject(double pos_x = 0, double pos_y = 0, double pos_z = 0);

        virtual ~glObject();

        virtual void        draw(int qualidade = 13) const;
        virtual int         qualidade(double pos_x, double pos_y);
        virtual double      get_x() const;
        virtual double      get_y() const;
        virtual double      get_z() const;
        virtual void        set_x(int pos_x);
        virtual void        set_y(int pos_y);
        virtual void        set_z(int pos_y);
        virtual bool        interceptObject(int x, int y) const;

        //Virtuais pura a ser implementadas nas classes filhas
        virtual void       paint(int qualidade = 13) const = 0;
        virtual void       draw_xy() const = 0;
        virtual glObject*  clone()   const = 0;

};

#endif // GLOBJECT_H
```

Figura 4.6: Classe glObject.

```
#ifndef GLOBJECTCOLLECTION_H
#define GLOBJECTCOLLECTION_H

#include "globject.h"
#include <vector>

class glObjectCollection : public glObject
{
private:
    std::vector<glObject*> m_objects;

public:
    glObjectCollection();
    glObjectCollection(const glObjectCollection&);

    void        addObject(const glObject& obj);
    void        clear();
    void        draw(int qualidade = 13) const;
    void        draw_xy() const;
    glObject*   clone() const;

    void        paint(int qualidade) const;
    int         quantObjetos() const;
    glObject*   getObject(int pos) const;
    bool        empty();
};

#endif // GLOBJECTCOLLECTION_H
```

Figura 4.7: Classe glObjectCollection.

Além das classes derivadas da *glObject* e *glObjectCollection* existem mais algumas classes fundamentais ao sistema, as classes *"protejo"* e *"settings"*.

4.1.2.2 As classes *settings* e *projeto*

A classe *settings* segue um padrão de projeto denominado *Singleton*, que segundo Horstmann (2006), define um único objeto daquele tipo para todo o programa, e a cada vez que é feita uma referência a ele sempre será chamando o mesmo objeto com os mesmo atributos e valores definidos, ou seja, os valores que a ele forem atribuídos continuaram os mesmos independente de qual parte do programa o chamarem, a menos que seu valor seja alterado em algum ponto, essa alteração será válida para todo o programa, e o atributo mudado passa a ter aquele valor. Com isso foi possível definir algumas variáveis de ambiente que deveriam ser acessíveis a qualquer momento e em qualquer parte do código, como por exemplo, "a quantidade de árvores em um único terreno".

Neste trabalho existe, também, uma classe denominada projeto, que implementa o "core" (núcleo) da aplicação. Ela é responsável por gerar a renderização das cenas, nela está inserido o conceito principal da OpenGL para o framework Qt, as funções:

```
initializeGL();
paintGL();
resizeGL(int width, int height);
```

Assim pode-se afirmar que a classe *projeto*, tem papel paralelo a classe *main*, que no caso deste trabalho é bem simples e somente existe como classe principal por padrão e obrigação do framework (BLANCHETTE; SUMMERFIELD, 2008). Na classe projeto está toda a lógica de execução do programa implementada, nela estão as funções responsáveis por calcular o posicionamento, tamanho e comportamento dos terrenos e das árvores em acordo com os parâmetros da classe *settings*. Esta classe também responsável por gerenciar eventos do teclado (acionamento de teclas, e resposta a elas) e o comportamento da câmera no cenário, de forma que, a união de teclado e câmera proporcionem ao usuário certo nível de realidade de acordo com a definição de simulação.

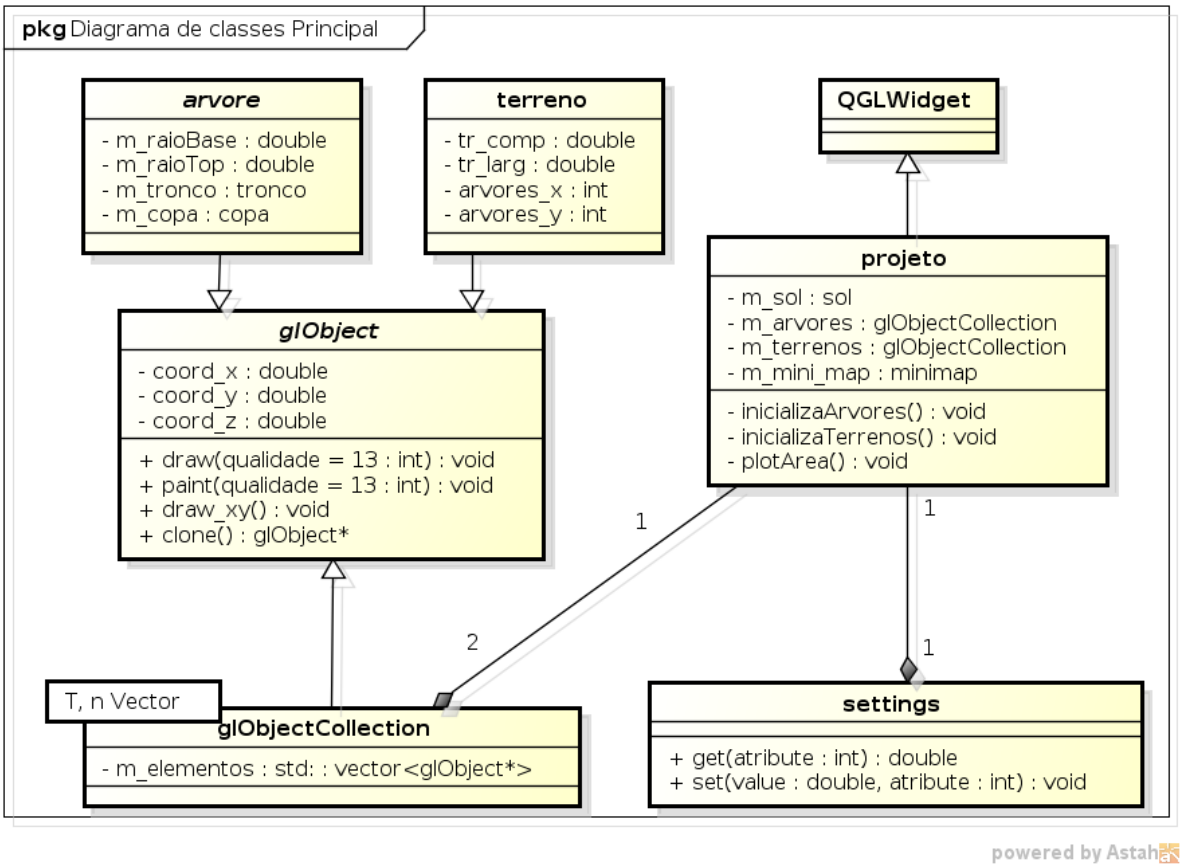

Figura 4.8: Estrutura principal das classes do Simulador.

4.1.3 Lógica de execução da aplicação

A aplicação "Ambiente de Simulação de uma Floresta Tridimensional" implementa uma lógica de execução muito simples, ela apenas interpreta os dados fornecidos pelo usuário para criar um ambiente em acordo com o especificado pelo mesmo nas variáveis de ambiente, ou seja, o simulador interpreta o que está contido nos parâmetros do sistema num dado momento do tempo. Ele inicia com uma configuração *defautl* (padrão), e a partir desta configuração o usuário tem total liberdade para alterar uma infinidade de parâmetros que o ajudaram a personalizar o ambiente da simulação.

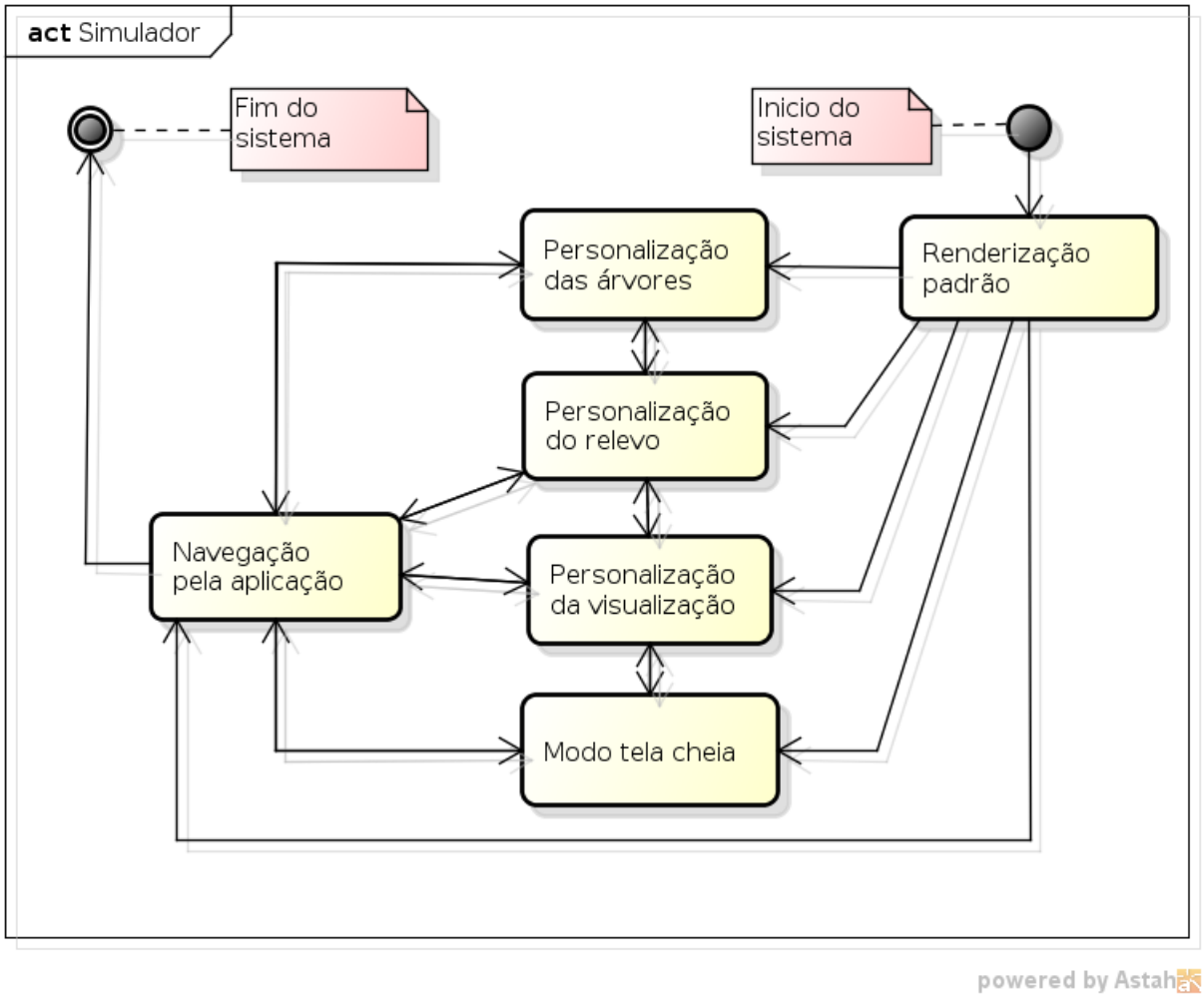

Figura 4.9: Lógica de execução da aplicação.

A lista de parâmetros editáveis pelo usuário são:

Nome do parâmetro	**Função do parâmetro na aplicação**
utilArea_comp	Tamanho dos talhões da floresta em unidades de medidas (U);
utilArea_larg	Largura dos talhões da floresta em unidades de medidas (U);
arvoresY	Define a quantidade de árvores no eixo Y;
arvoresX	Define a quantidade de árvores no eixo X;
arvoresAlturas	Define a geral das árvores;
arvoresRaiobase	Estipula o raio da base de todas as árvores;
arvoresRaiotopo	Estipula o raio do topo de todas as árvores;
arvoresDistx	Distância entre arvores no eixo X;
arvoresDisty	Distância entre arvores no eixo Y;
terrenosX	Quantidade de terrenos no eixo X;
terrenosY	Quantidade de terrenos no eixo Y;
m_show_minimap	Expressa se deve ou não ser mostrado o minimapa;
velox	Velocidade de movimentação da visão pelo ambiente;
rotate	Quantidade em graus rodadas ao utilizar as teclas direcionais

Tabela 4.1: Parâmetros editáveis do sistema

Depois de renderizada a cena, o usuário pode navegar por ela usando algumas teclas especificas do seu teclado, sendo que a sua mobilidade é total, podendo ele entrar no meio da floresta e sobrevoa-la para visualizar melhor a cena gerada.

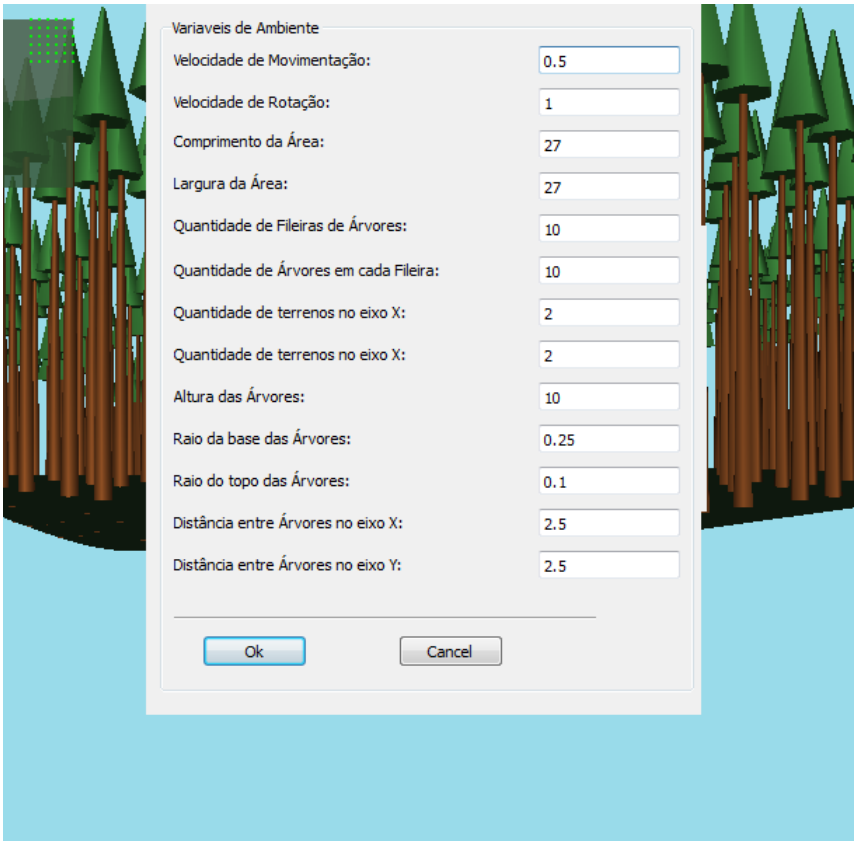

Figura 4.10: Menu de variáveis do sistema.

4.2 A simulação

A simulação, consiste na interação do usuário com o ambiente gerado de acordo com as especificações do mesmo. Esse ambiente é composto, como já informado anteriormente pelos objetos "Árvore" e "Relevo". Para ficar mais claro como a simulação serão detalhadas as participações destes objetos no cenário, bem como a interação da visão (câmera) com esse cenário e os objetos. A cada alteração dos parâmetros do ambiente um novo relevo é carregado, ele está em constante atualização, pois a função paintGL(), conforme dito no capítulo 3, está a cada iteração do *loop* de execução do programa, gerando uma nova visualização em acordo com os parâmetros passados a ela.

4.2.1 Objeto Árvore

O modelo do objeto "Árvore" é constituído basicamente de duas figuras geométricas, "Cilindro" e "Cone", sendo que o cilindro representa o tronco da árvore e o cone a copa da árvore assim, a figura cone e plotada 'n' unidades acima do solo nas mesmas coordenadas (x,y) onde o tronco foi plotado, gerando assim o efeito esperado, de uma conífera. É possível ver um exemplo desta aplicação na imagem abaixo:

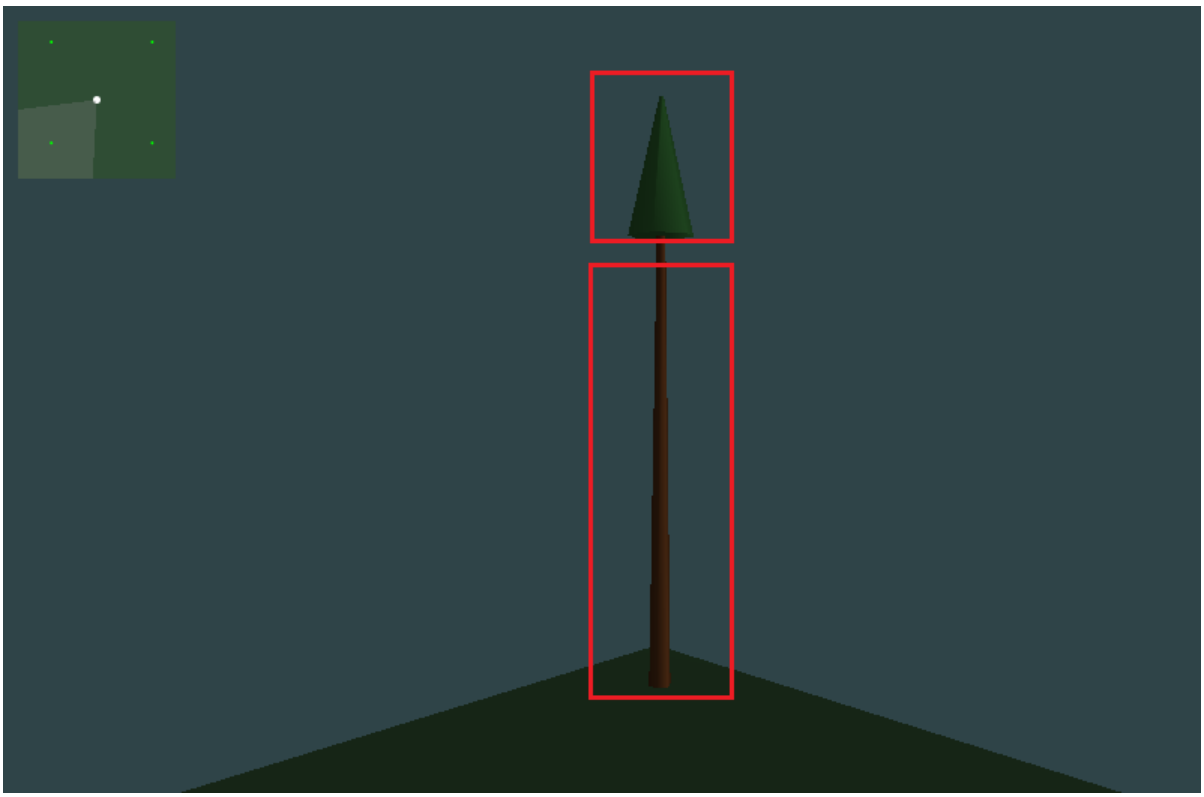

Figura 4.11: Cilindro e cone unindo-se e formando o aspecto de uma conífera.

Os códigos para gerar o objeto árvore são bem simples, por a biblioteca *glu* (SHREINER, 2009) prover funções que já plotam figuras como, cilindros, esferas, cubos, etc. Assim, a constituição de um modelo de árvore pode ser simplificado. No caso especificado da árvore, foi utilizada a mesma função para criar as duas figuras que a compõe (cilindro e cone), a função da *glu* (SHREINER, 2009) responsável por plotar um cilindro na tela, porem, no caso do "Cone" foi utilizada uma propriedade interessante da função, os parâmetros referentes aos raios da base e raio do topo da figura.

```
gluCylinder(Objeto GLUquadric,raio base,raio topo,altura,
fatias,pilhas);
```

A definição dos parâmetros de *gluCylinder* segundo Shreiner (2009) são:

- **Objeto *GLUquadric*** - O objeto quádrica (criado com gluNewQuadric).
- **Raio base** - O raio do cilindro, na altura z = 0.
- **Raio topo** - O raio do cilindro, na altura z = **altura**
- **Fatias** - O número de subdivisões em torno do eixo z.
- **Pilhas** - O número de subdivisões ao longo do eixo z.

Quando os parâmetros dos raios da base e do topo não são idênticos a figura formada se afunila em direção ao menor dos dois parâmetros informado, no caso o raio do topo da figura, logo foi possível ter o efeito desejado, para a copa da árvore um cone. O usuário tem através desse sistema de parâmetros (altura, raio da base e raio do topo) das duas figuras que compõem a árvore, existe a possibilidade do usuário gerar diversas visualizações diferentes para o objeto árvores, conforme as imagens abaixo mostram:

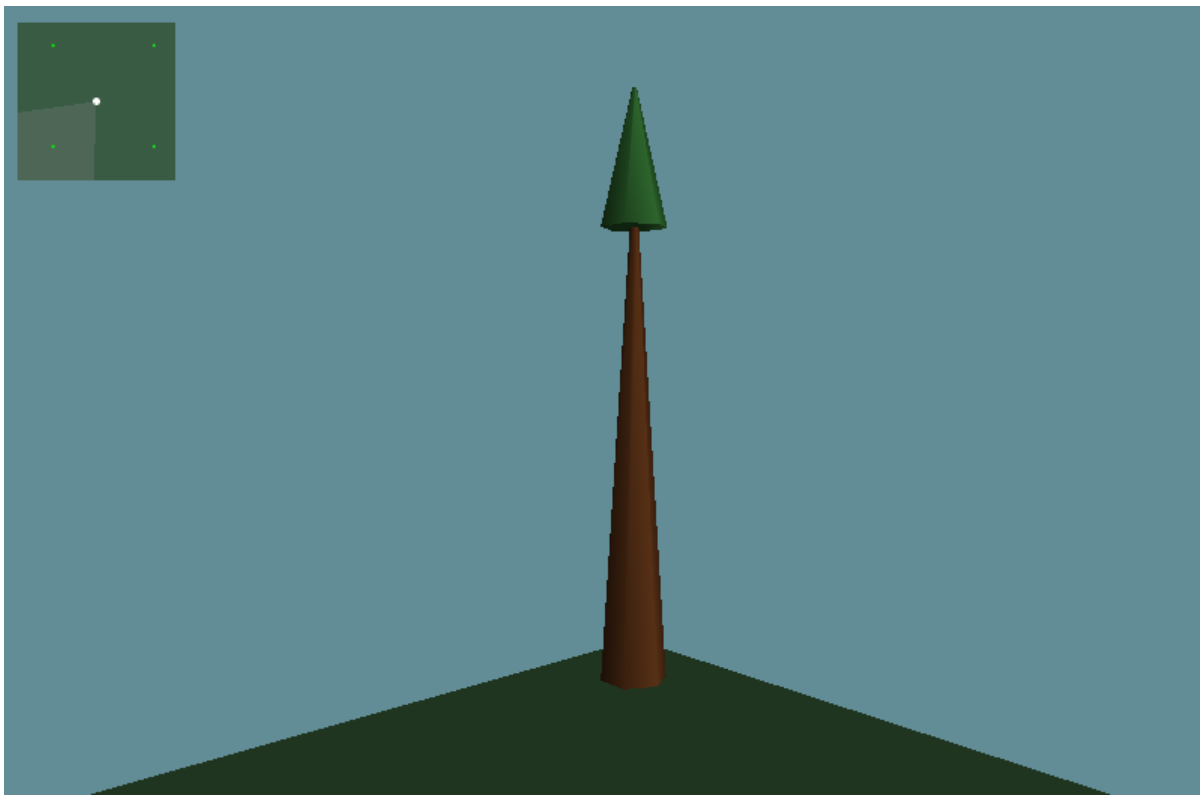

Figura 4.12: Aumento do parâmetro "Raio da base".

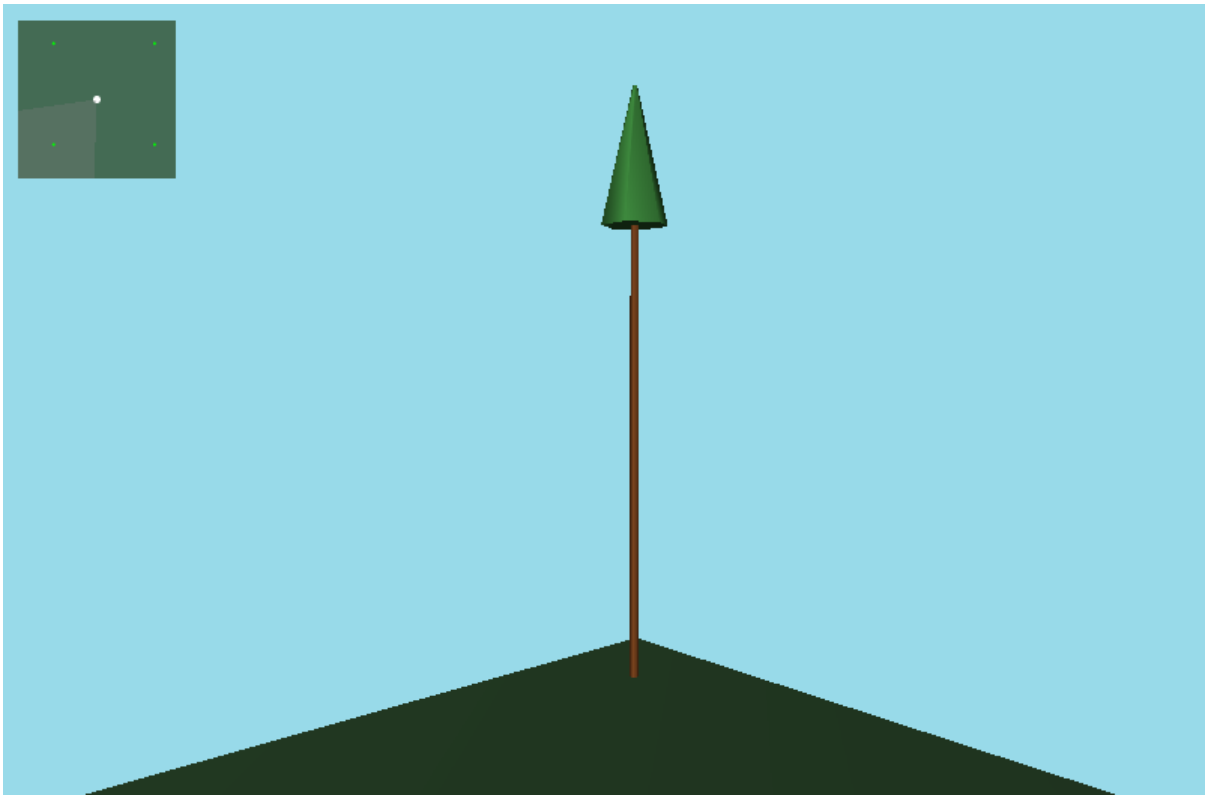

Figura 4.13: Redução do parâmetro "Raio da base".

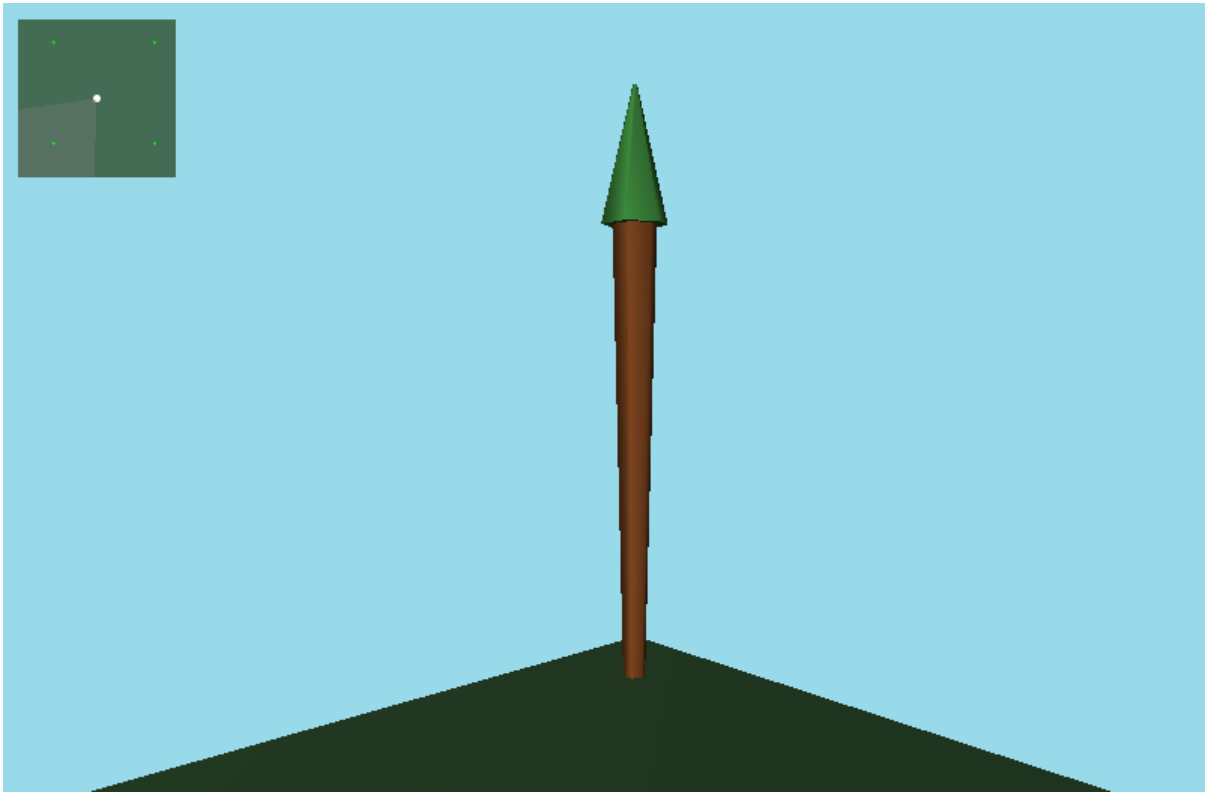

Figura 4.14: Aumento do parâmetro "Raio do Topo".

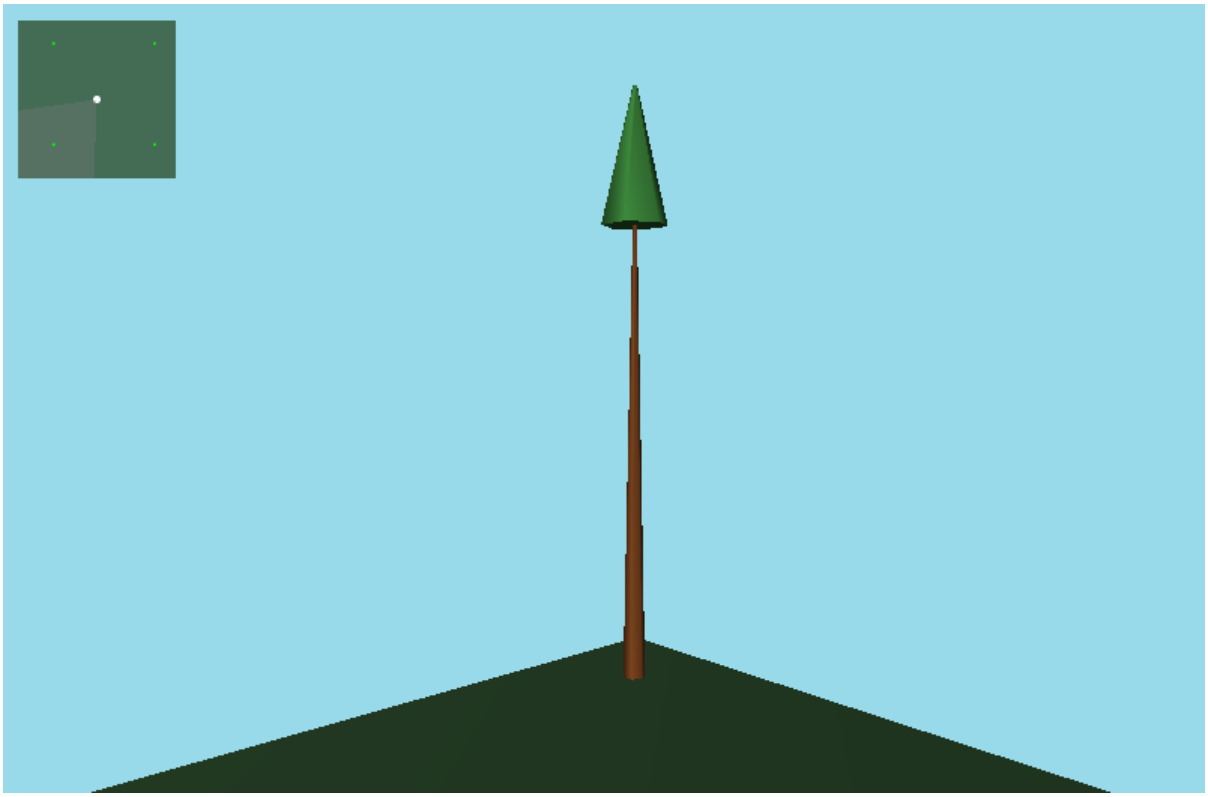

Figura 4.15: Redução do parâmetro "Raio do Topo".

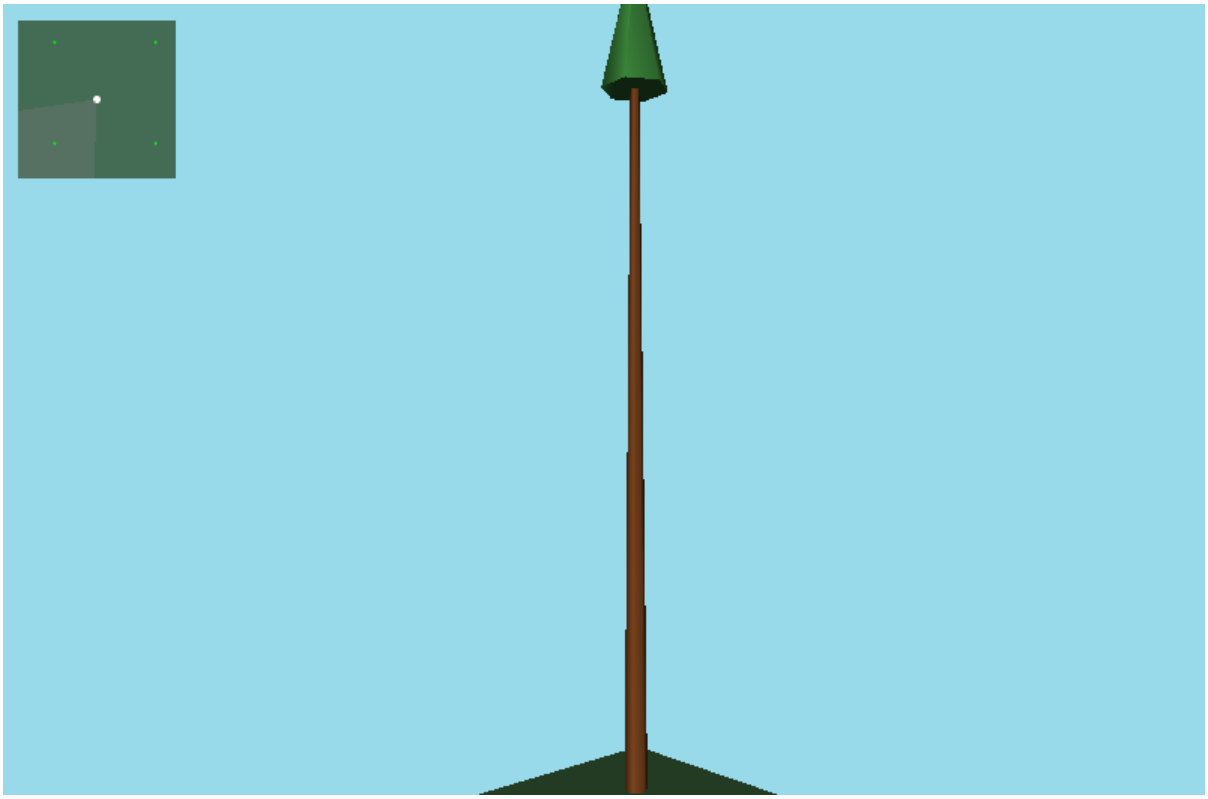

Figura 4.16: Aumento no parâmetro "Altura".

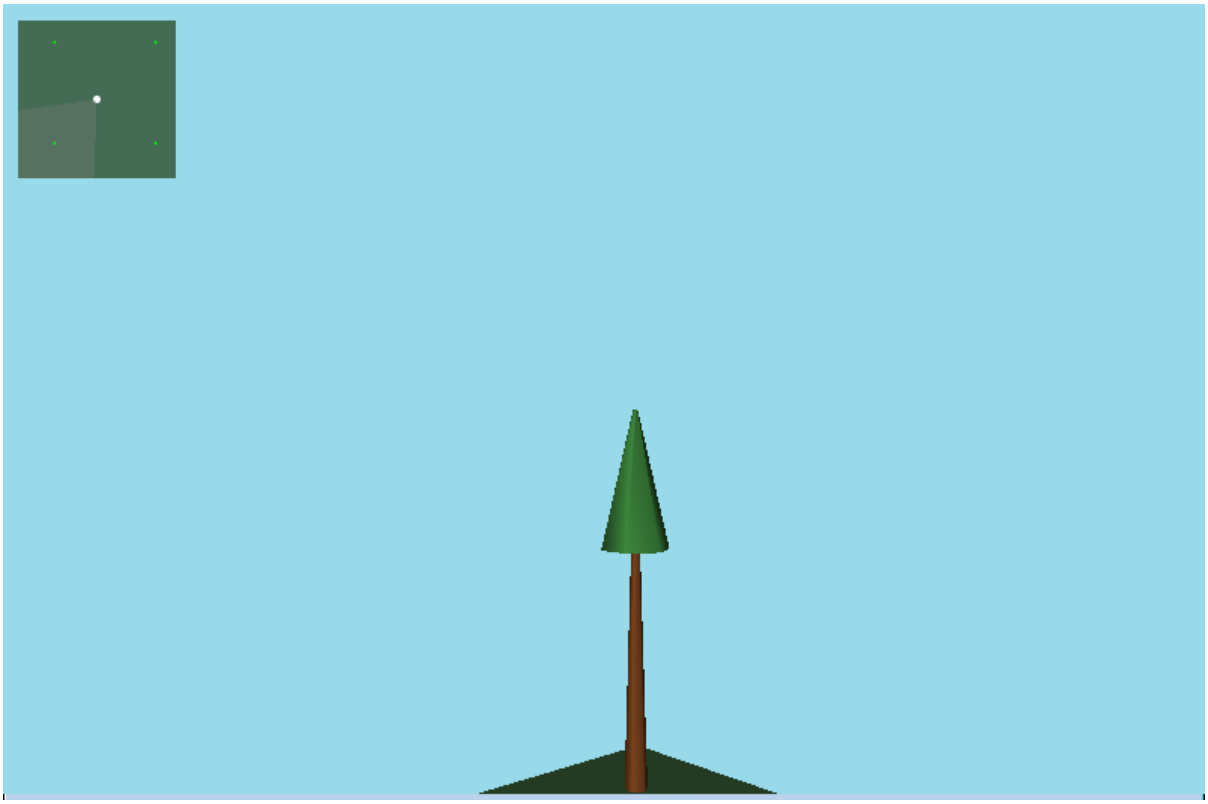

Figura 4.17: Redução no parâmetro "Altura".

Além, da personalização do modelo do objeto "Árvore" o usuário ainda pode definir outras propriedades ligadas ao objeto para aumentar a realidade da visualização gerada. É permitido ao usuário alterar o parâmetro "Distância entre árvores" e "Quantidade de arvores". No caso da distância o usuário pode indicar tanto no eixo X quanto no eixo Y quantas unidades de medida que cada árvore terá em relação a sua vizinha, tornando viável simular florestas mais densas ou mais esparsas. Não é diferente com o parâmetro quantidade de árvores. Com ele pode-se definir, como no parâmetro distância entre árvores, a quantidade de árvores no eixo X e no eixo Y que cada talhão (unidade quadrada de relevo, usada por Engenheiros florestais para indicar um quadrado de grandes proporções que no seu interior está repleto de árvores) irá comportar, podendo o usuário gerar talhões mais povoados de árvores ou menos povoado de árvores.

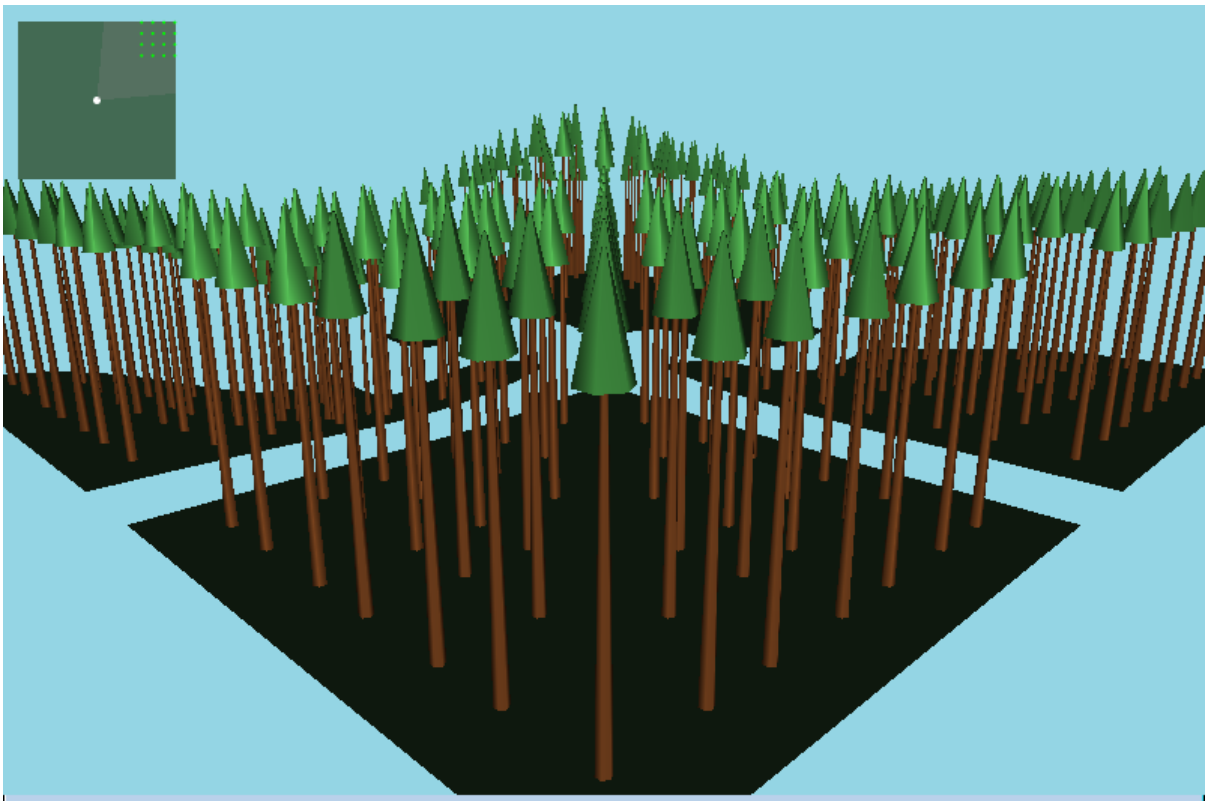

Figura 4.18: Aumento no parâmetro "distância entre árvores", tanto no eixo x como no eixo y.

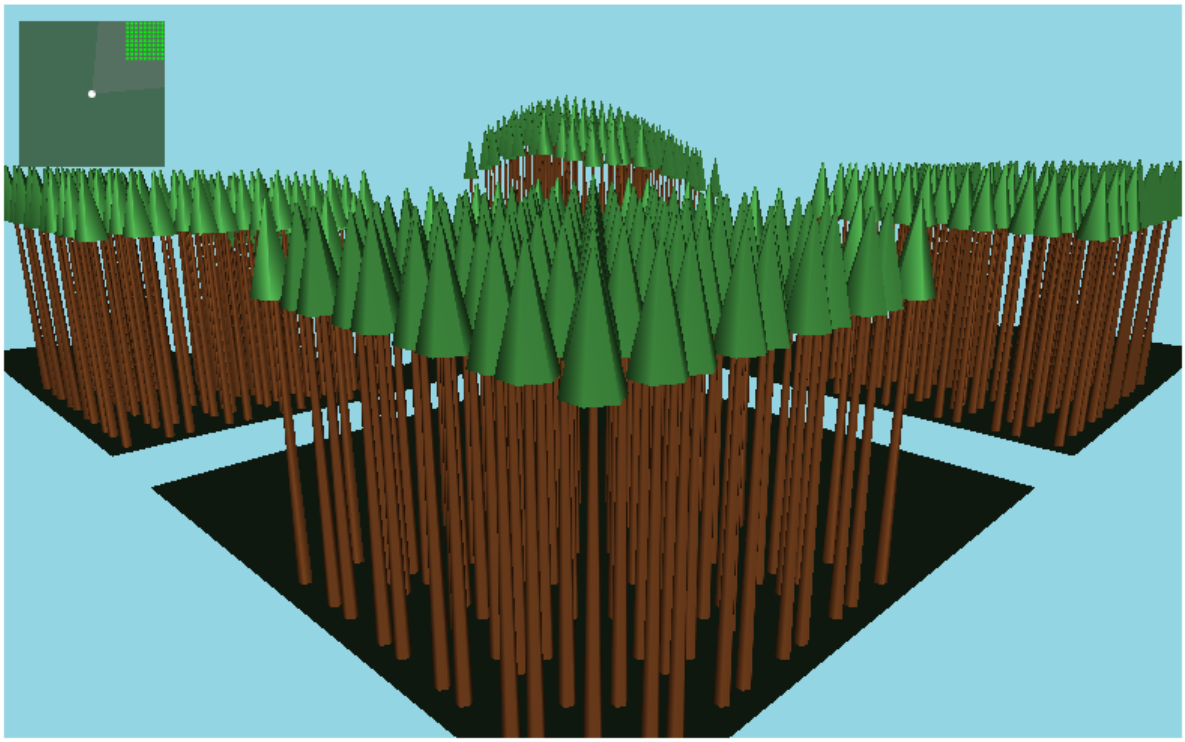

Figura 4.19: Redução no parâmetro "distância entre árvores", tanto no eixo x como no eixo y.

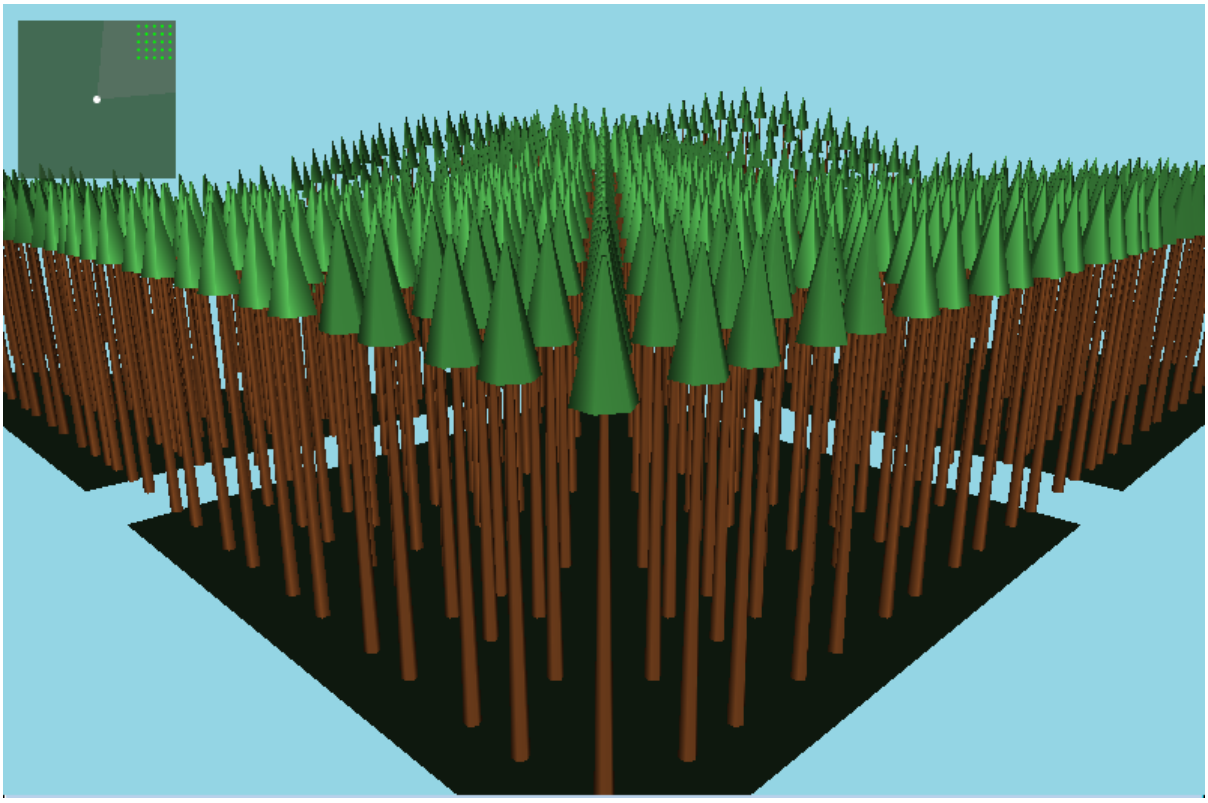

Figura 4.20: Aumento no parâmetro "quantidade de árvores", tanto no eixo x como no eixo y.

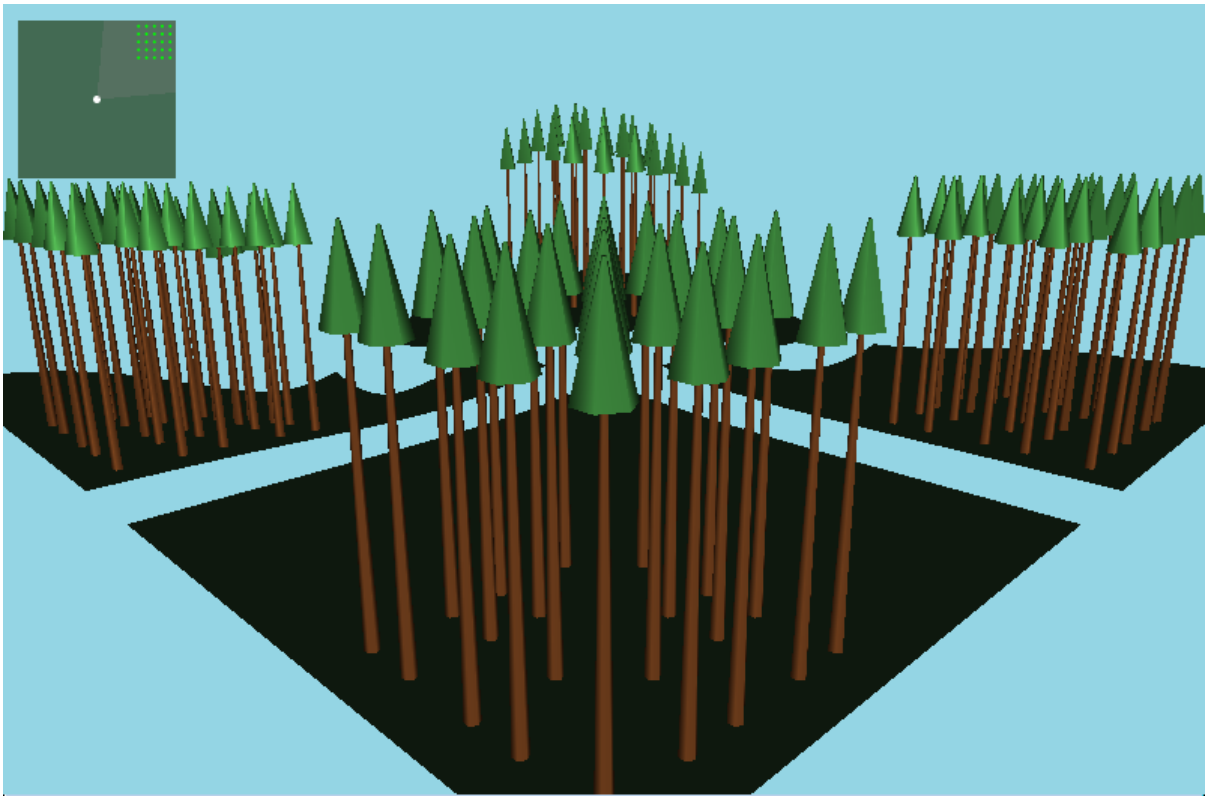

Figura 4.21: Aumento no parâmetro "quantidade de árvores", tanto no eixo x como no eixo y.

4.2.2 Objeto Relevo

O objeto "Relevo" tem por finalidade simular, como o próprio nome sugere, o relevo do cenário. Na aplicação ele é constituído unicamente de pequenos quadriláteros, que se encaixam formando os talhões da simulação. Esse pequenos quadriláteros são implementados no sistema diretamente pela função da OpenGL *glBegin(GL_QUADS)*, cujo o efeito produz quadriláteros a partir de quatro pontos fornecidos a ela pela função *glVertex3f()* antes do termino da função *glBegin(GL_QUADS)* (terminada por *glEnd()*). Ao contrário do objeto "árvore", que era constituído de objetos já implementados pela biblioteca *glu* (SHREINER, 2009), o objeto relevo foi implementado diretamente por uma função da OpenGL. A função que gera os quadriláteros pode ser vista abaixo:

```
glBegin(GL_QUADS);
  glVertex3f(x,y,z);
  glVertex3f(x,y,z);
  glVertex3f(x,y,z);
  glVertex3f(x,y,z);
glEnd();
```

Assim, cada função *glVertex3f()* será responsável por gerar um ponto no espaço com as coordenadas (x,y,z), cada coordenada é representada por um valor em ponto flutuante (no formato 0.0). A função *glBegin(GL_QUADS)* dará inicio a máquina de estados da OpenGL, colhendo os pontos informados anteriormente, unindo-os, para formar o quadrilátero desejado. Caso desejássemos um relevo totalmente plano, bastava que, a cada talhão fosse usado somente um quadrilátero com grandes proporções (cada ponto incluído na máquina de estados seria um vértice do talhão), e com todos os parâmetros "z" indicados com o valor '0.0'. Produzindo o aspecto mostrado abaixo:

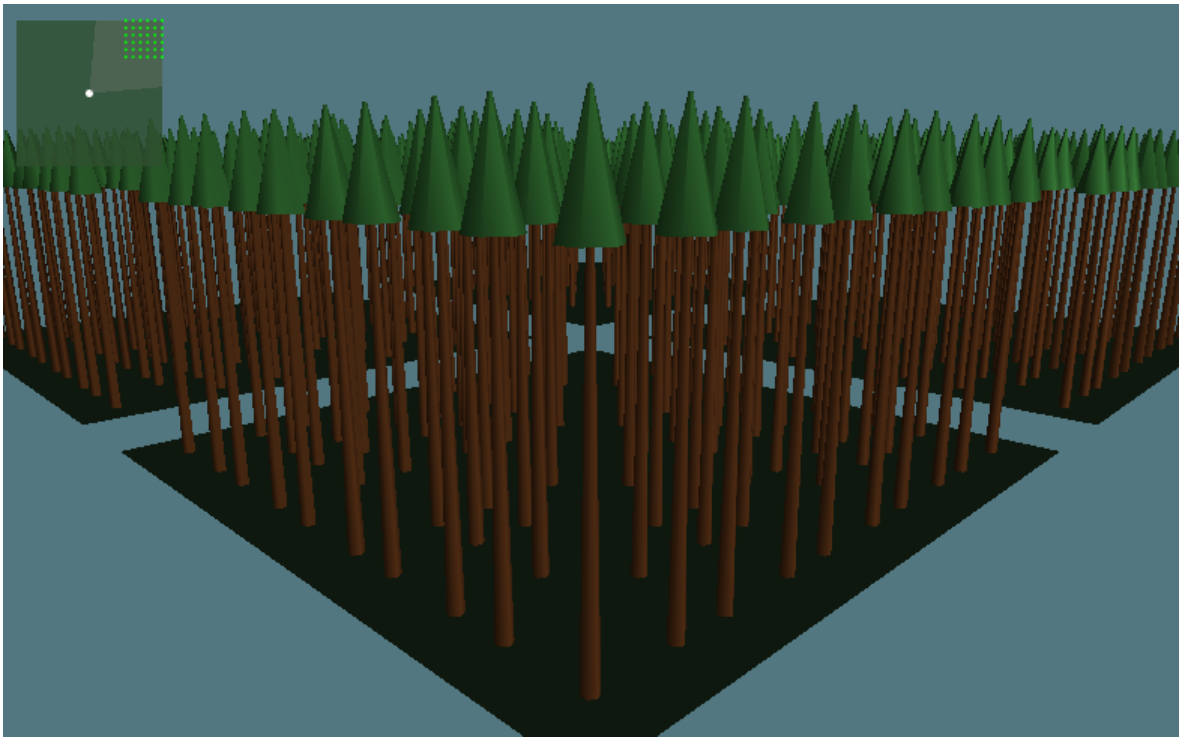

Figura 4.22: Demonstração do relevo plano.

Diferentemente da imagem anterior no caso do simulador proposto foi usada uma matriz de relevo, com o intuito de aumentar a realidade proporcionada pelo simulador. Essa matriz funciona da seguinte forma, ao invés de usar um único quadrilátero representando o talhão, utiliza-se diversos quadriláteros o constituindo. Com isso é possível atribuir a cada um deles, em cada um dos seus vértices, diferentes valores para o parâmetro "z", que ao unir lado-a-lado os quadriláteros, darão ao relevo o aspecto de deformação, ondulação, depressão, etc. Para que isso seja possível foi implementado um sistema que, recebe uma matriz de alturas, que será processada e embutida no simulador para gerar o efeito citado. Essa matriz deve ser no seguinte formato:

Linhas			
Colunas			
$altura_{11}$	$altura_{12}$	$altura_{1...}$	$altura_{1n}$
$altura_{21}$	$altura_{22}$	$altura_{2...}$	$altura_{2n}$
$altura_{...1}$	$altura_{...2}$	$altura_{.....}$	$altura_{...n}$
$altura_{m1}$	$altura_{m2}$	$altura_{m...}$	$altura_{mn}$

Tabela 4.2: Formato do arquivo de alturas para o relevo.

Tem-se que "Linhas" e "Colunas" são respectivamente a quantidade de linhas e colunas contidas no arquivo, necessárias para saber até qual talhão esse arquivo abrange. E cada elemento da matriz($altura_{mn}$) representa a altura do ponto (mn) no sistema de coordenadas em relação a todo o cenário, assim é possível não somente alterar a altura em que os vértices dos quadriláteros que se encontram como também, pode-se alterar a altura dos objetos árvores em relação ao relevo, conforme mostrado na imagem abaixo:

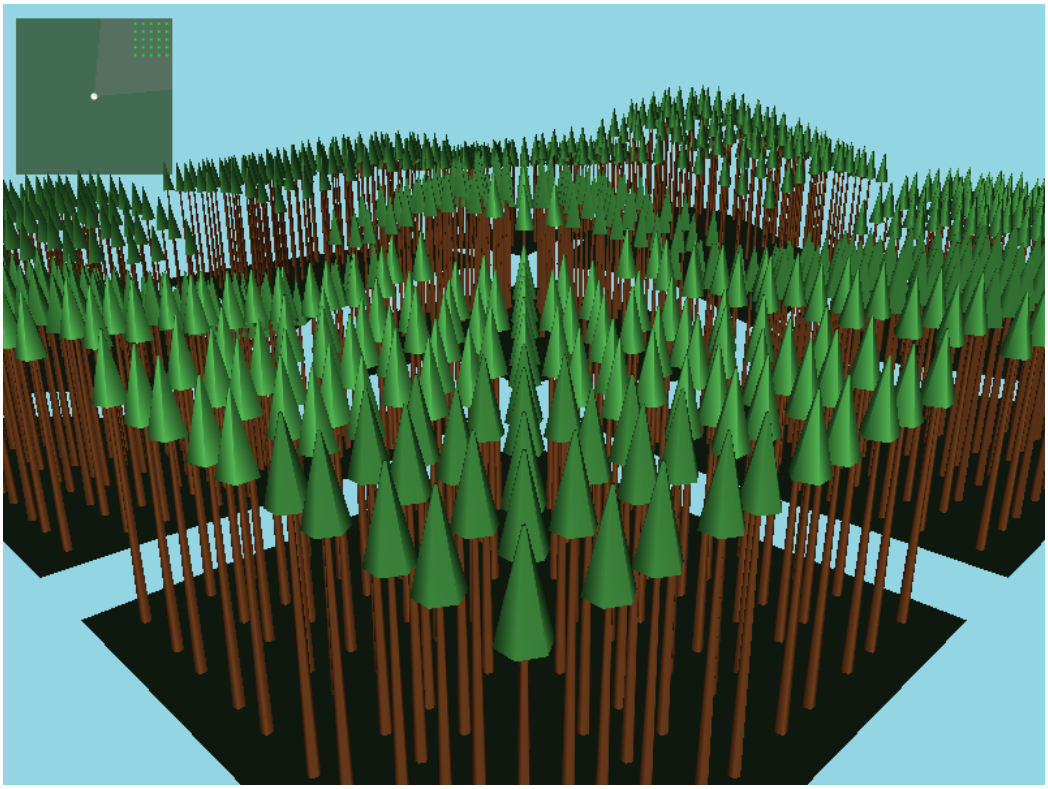

Figura 4.23: Demonstração do relevo Deformado(ondulado, tentando simular a realidade).

Assim como o objeto árvore tem-se alguns parâmetros ligados diretamente ao relevo, com o intuito de tornar a simulação o mais realista o possível, permitindo ao usuário personaliza-la da forma que mais lhe interessar. Os parâmetros ligados diretamente ao relevo são "Tamanho dos talhões","Largura dos talhões","Quantidade de talhões no eixo X" e "Quantidade de talhões no eixo Y". O tamanho do talhão vai parametrizar no sistema a distância em unidades de medida (U) do talhão no eixo X em relação ao seu ponto inicial, bem como a largura irá indicar no

sistema qual a distância em unidades de medida (U) do talhão no eixo Y em relação ao seu ponto inicial. Abaixo exemplos de alterações destes dois parâmetros para os talhões:

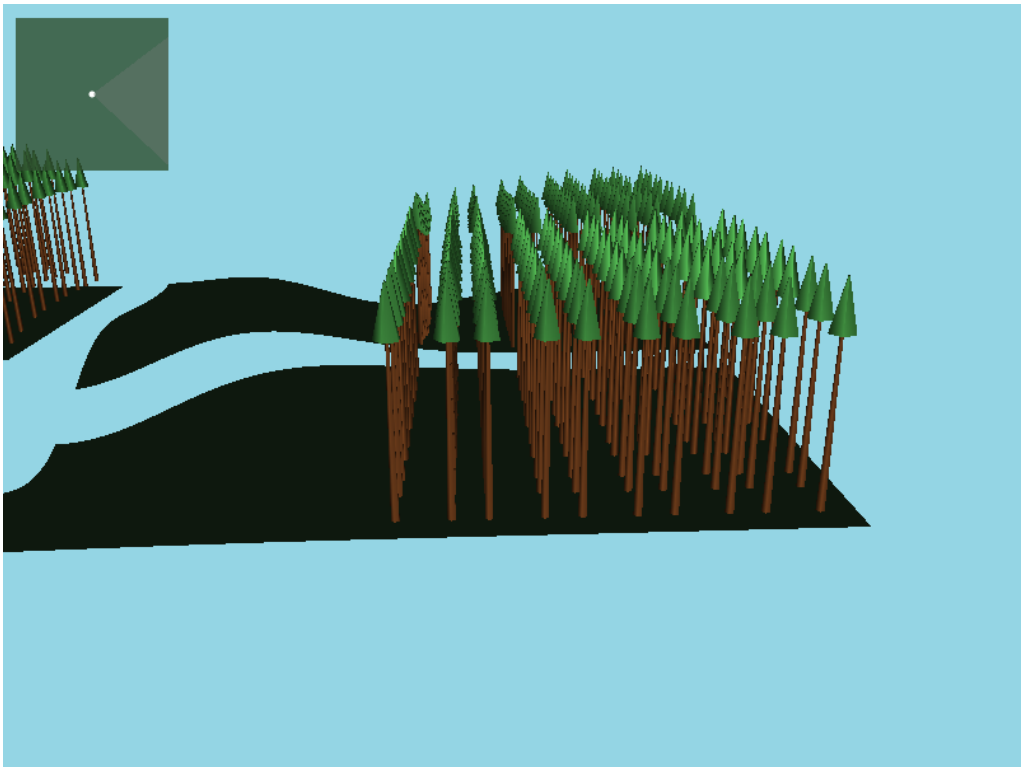

Figura 4.24: Alteração no parâmetro "tamanho do talão" (eixo X).

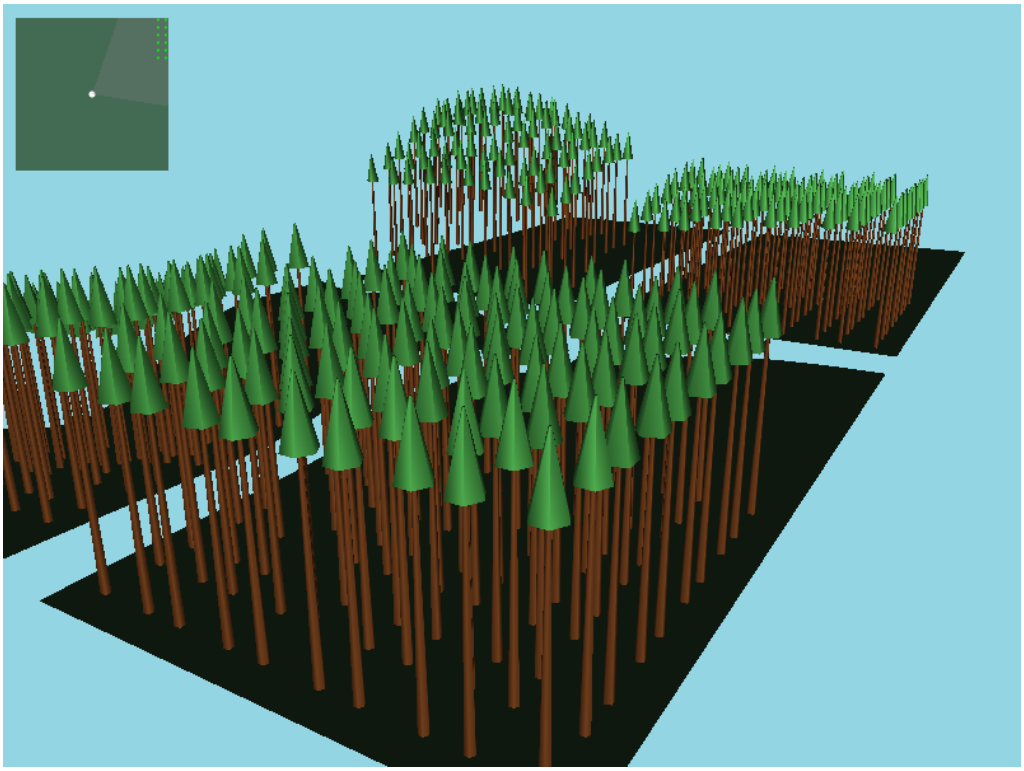

Figura 4.25: Alteração no parâmetro "largura do talão" (eixo Y).

Além do tamanho dos talhões, também é possível personalizar a quantidade de cada um deles tanto no eixo X quanto no eixo Y, a fim de elaborar a floresta com as dimensões desejadas pelo usuário. É possível visualizar exemplos dessas mudanças abaixo:

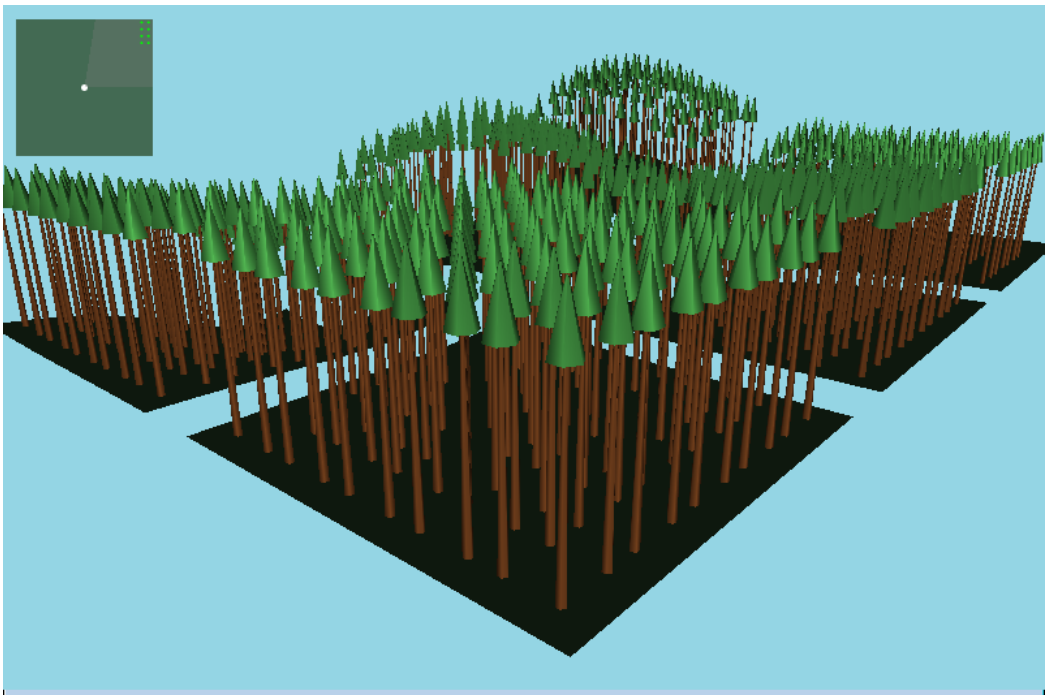

Figura 4.26: Alteração no parâmetro "largura do talão" (eixo Y).

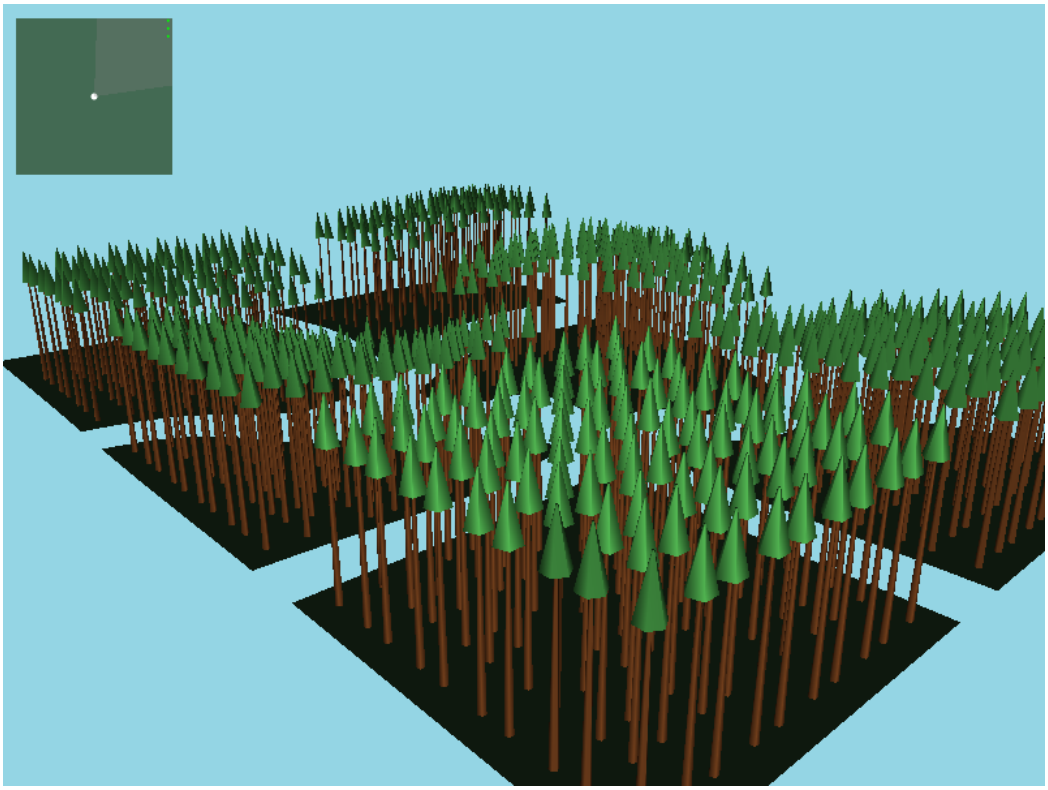

Figura 4.27: Alteração no parâmetro "largura do talão" (eixo Y).

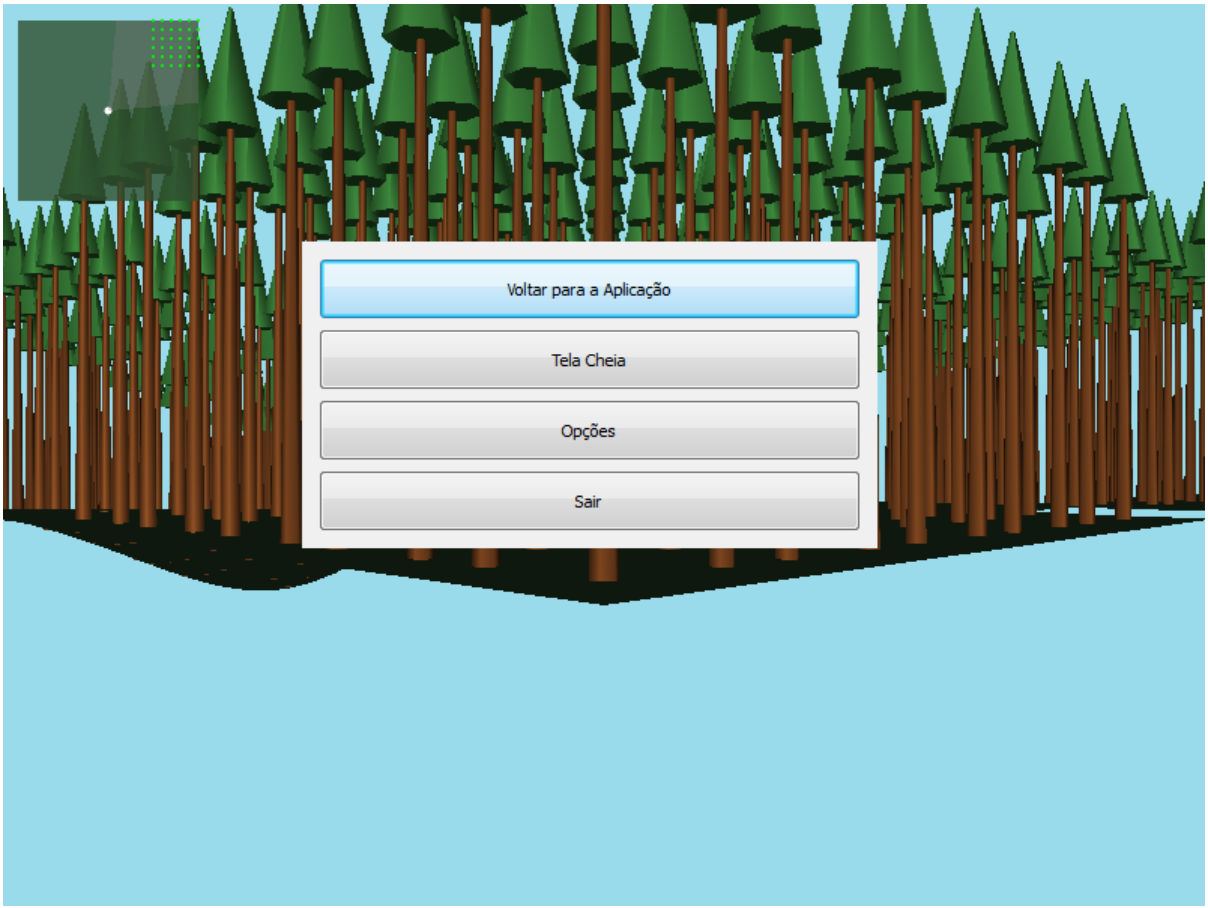

Figura 4.28: Menu de contexto do sistema.

4.2.3 A visualização

Na aplicação " Ambiente de Simulação de uma Floresta Tridimensional" o ultimo componente da simulação é a visualização. Ela é responsável por simular o observador da cena no programa, mas diferentemente de uma simples câmera que acompanha a cena do mesmo ponto, no caso desta aplicação ela é ajustável da forma mais interessante para o visualizador, sendo assim, ele pode posicionar esta visualização no ponto, e anglo que quiser, para aumentar sua compreensão e visão da cena. Pode-se resumir a visualização, como sendo uma câmera não estática, que flutua no cenário. Ao visualizador é permitido, conforme dito anteriormente, os seguintes movimentos:

- Mover-se para frente e para trás;
- Mover-se lateralmente para esquerda e direita;
- Girar a visualização para a esquerda e direita;

- Angular a visualização para cima e para baixo;
- Aumentar ou diminuir a altura da visualização em relação ao cenário;

Ela é implementada pela função *gluLookAt* (SHREINER, 2009) da biblioteca *glu* (SHREINER, 2009). É constituída de três conceitos principais, o "olho" (Observador), o "centro da visão" (ponto observado) e o "sentido da câmera". O olho define a posição atual de onde se está observando a cena, e o centro da visão define para qual ponto do espaço o olho está observado, resumidamente funciona como o olho humano, que possui o campo de visão e o ponto para qual a visão está focada.

> gluLookAt cria uma matriz de visualização derivada do ponto do "Olho (observador)", de um ponto de referência indicando o centro da cena, e um vector de "Sentido da visualização". A matriz de pontos de referência mapeia o eixo z na sua porção negativa e o ponto do olho na origem. Quando uma matriz de projeção típica é utilizada, o centro da cena, é portanto, mapeado no centro da janela. Da mesma forma, a direção descrita pelo vetor "Sentido da visualização" é projetado no plano de visualização, e é mapeado para o eixo y na sua porção positiva para que ele aponte para cima no visor (SHREINER, 2009).

Esta figura resume basicamente o conceito:

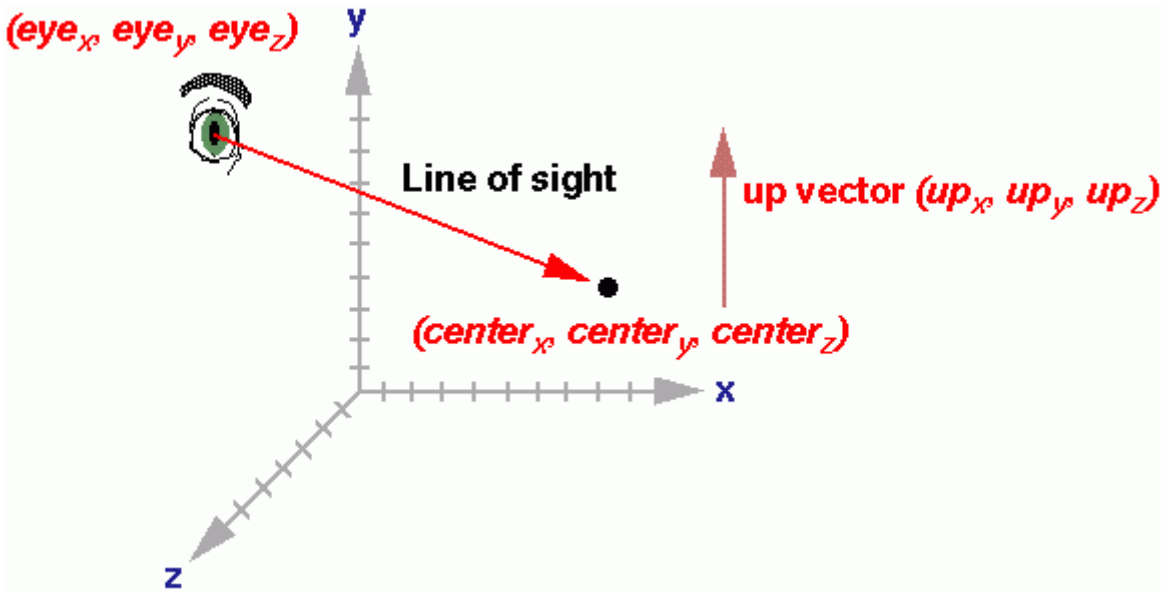

Figura 4.29: Exemplificação da função *gluLookAt*. Retirado de http://profs.sci.univr.it/ colombar/html_openGL_tutorial/en.

Como pode ser observado na figura são expostos, duas coordenadas, no espaço tridimensional, uma representa a posição do olho, a outra o centro da visão e vetor o outro o sentido da câmera, respectivamente, representandos pelos parâmetros (eye_x,eye_y,eye_z), $(center_x,center_y,center_z)$ e (up_x,up_y,up_z). No sistema a função está desta forma:

```
gluLookAt(xEye, yEye, zEye, xCenter, yCenter, zCenter,
          0.0, 0.0, 1.0);
```

No sistema existem as variáveis **xEye, yEye, zEye, xCenter, yCenter, zCenter** que corresponde respectivamente aos parâmetros citados acima, e os últimos três valores representam que a visão está no sentido do eixo . Esses parâmetros que permitem a movimentação da visualização ao longo do cenário, bastando somente alterar seus valores, para que na próxima iteração do *loop* da função *paintGL()* eles sejam alterados, gerando uma nova visualização em uma posição diferente do espaço tridimensional. Quando é apertado o botão correspondente a ação mover-se para frente são alterados todos os parâmetros que estão em função do parâmetro "velocidade de movimentação". Para garantir que a movimentação corresponda ao esperado, foram necessárias realizar transformações geométricas de seno no parâmetro **yEye** e cosseno parâmetro **xEye** para que eles não se comportassem de forma errada. Além destas alterações também é observada a posição da visualização em relação ao relevo a cada movimentação, se o relevo aumentar a altura no novo ponto que a visualização encontra-se ela também terá sua altura aumentada, de forma semelhante para reduções de tamanho, dando a visualização o aspecto de percorrer o terreno, como ocorre na realidade.

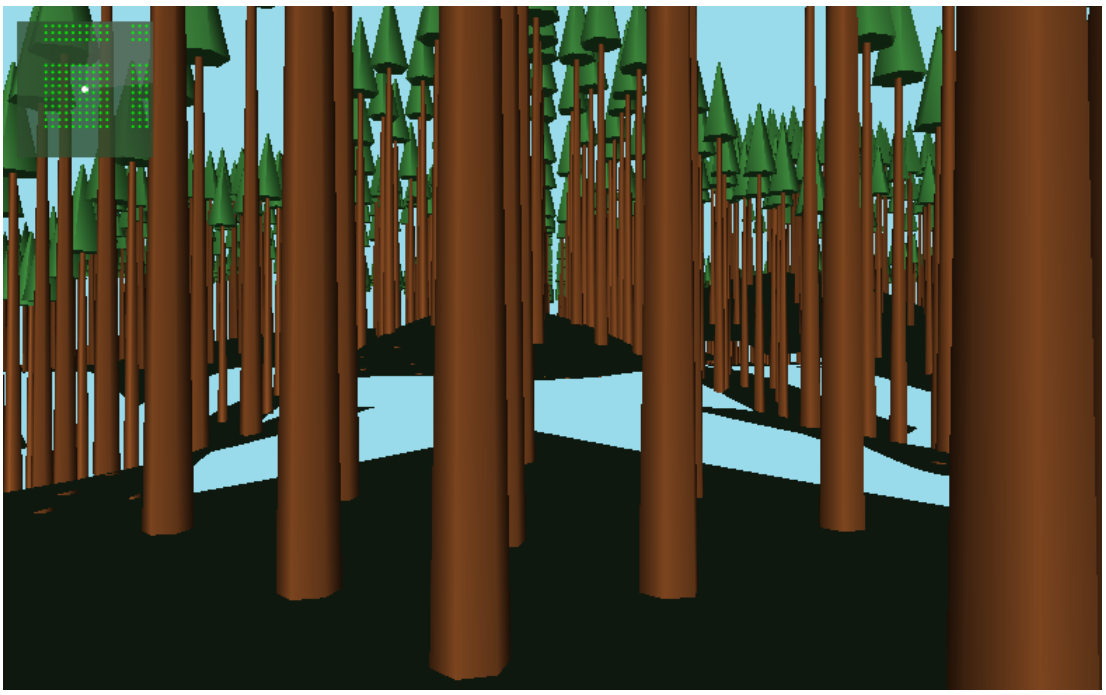

Figura 4.30: Visualização deslocada até um ponto no meio da floresta.

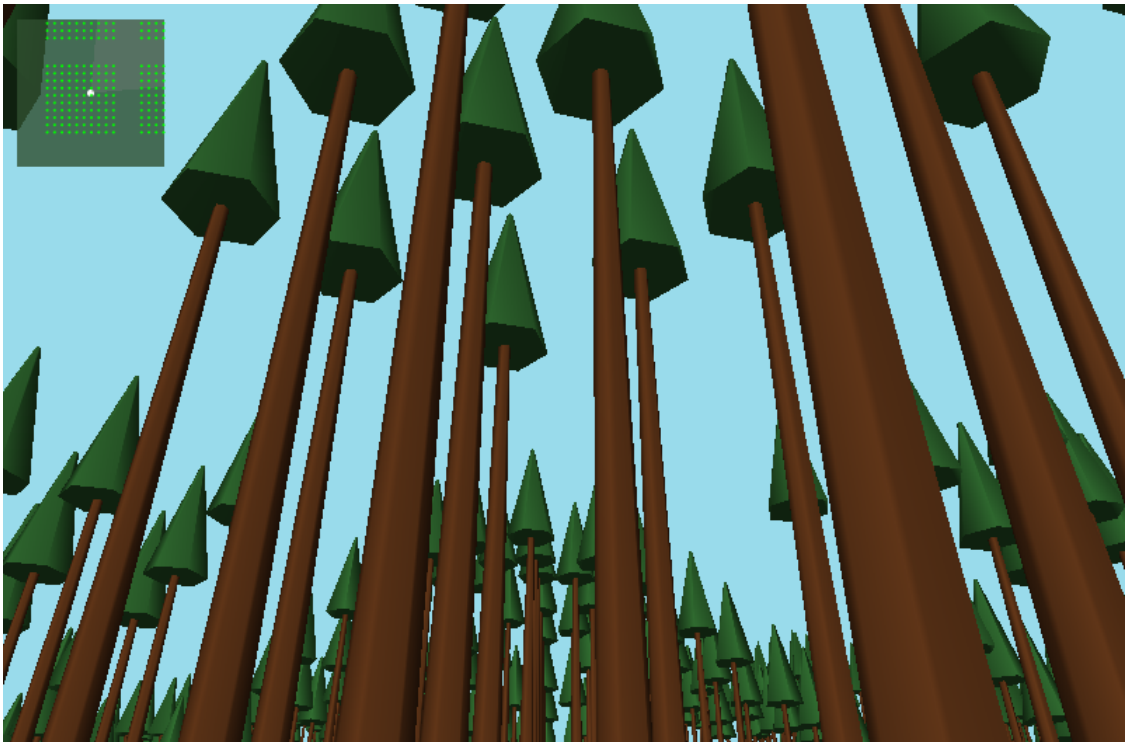

Figura 4.31: Centro de visualização angulado para cima.

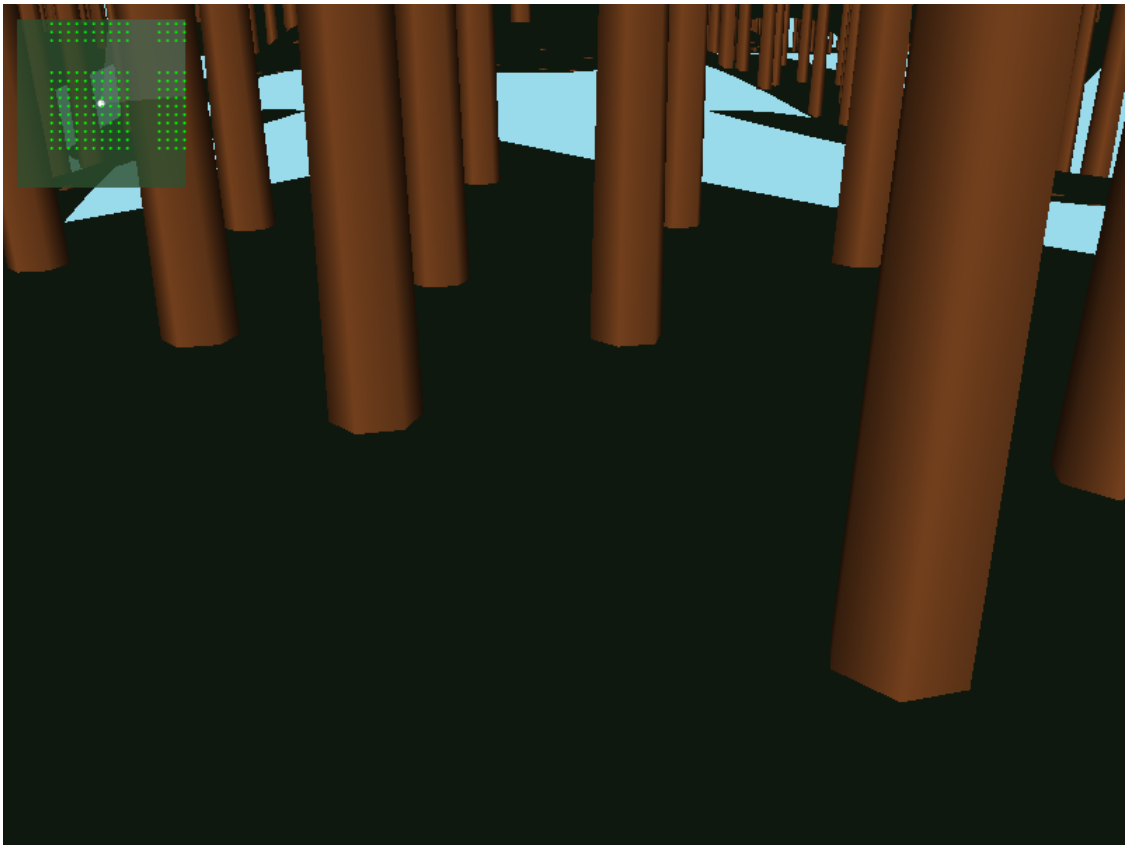

Figura 4.32: Centro de visualização angulado para baixo.

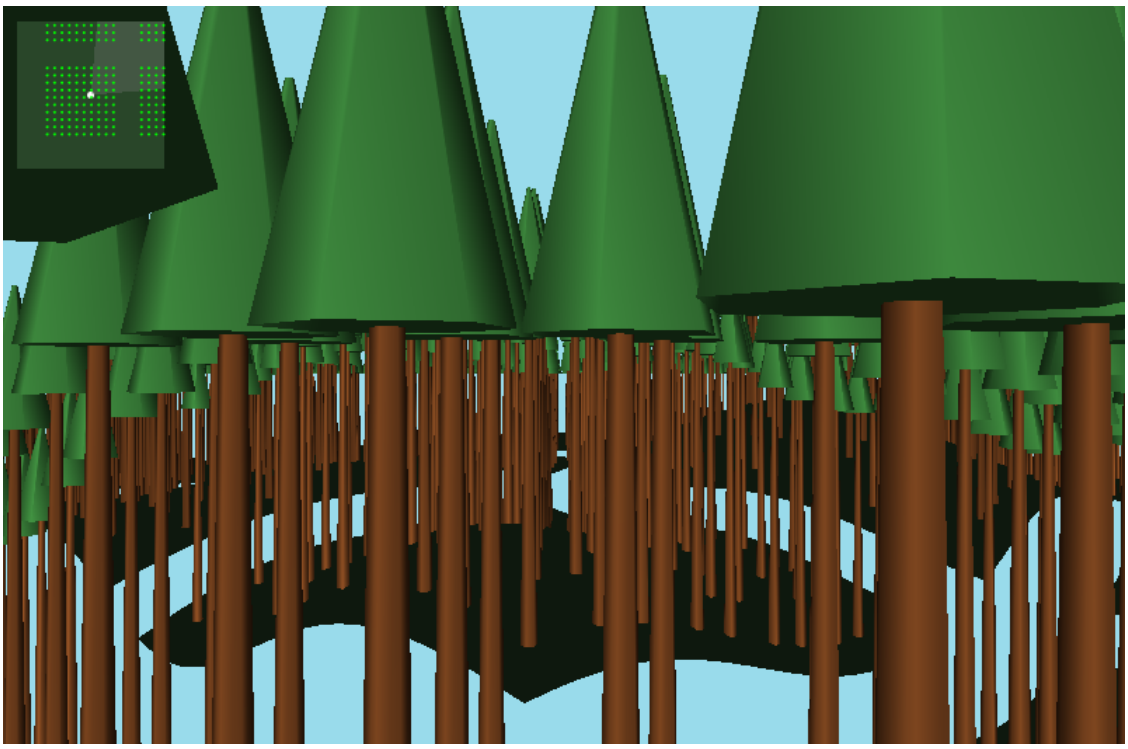

Figura 4.33: Aumento no parâmetro "z" da visualização, causando aumento da altura em relação ao solo.

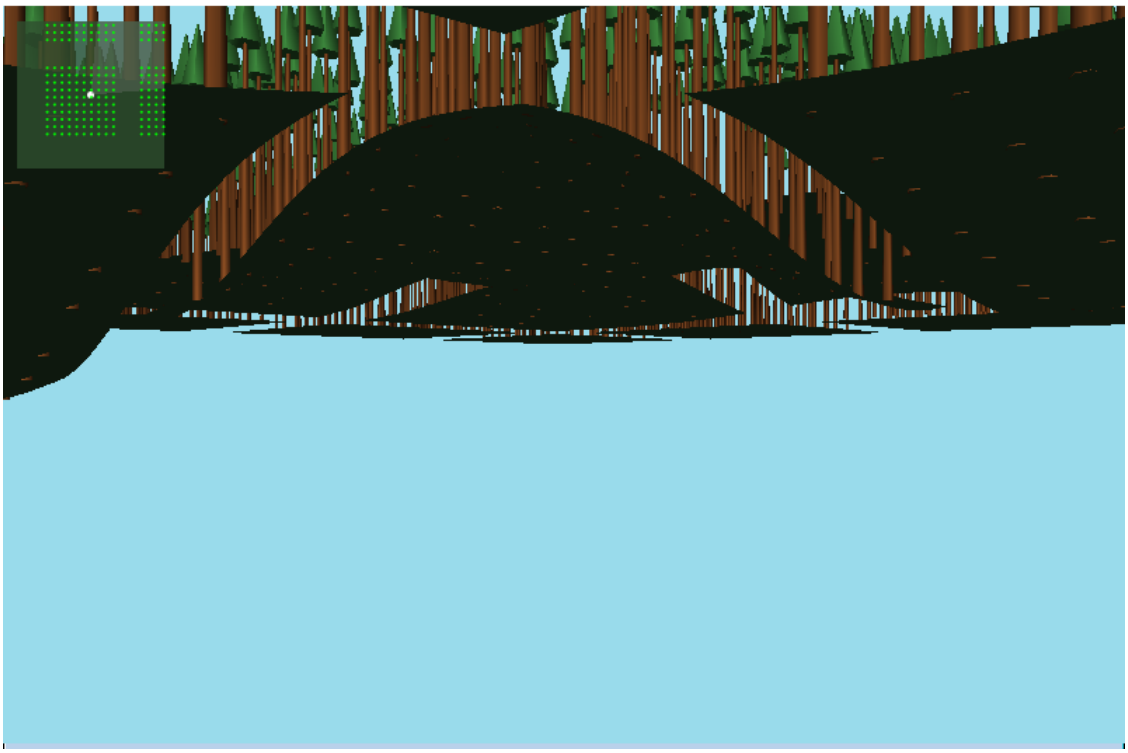

Figura 4.34: Redução no parâmetro "z" da visualização, causando redução da altura em relação ao solo.

Alguns artifícios foram utilizados para tornar aplicação menos lenta, facilitando o seu processamento mesmo em máquinas sem muito poder de processamento. Essas alterações foram, "Fator de visualização", "Fator de qualidade" e "Redução da resolução".

- **Fator de visualização** - Esse fator define a qual distância do observador, os objetos da cena serão visíveis, assim, objetos além deste ponto são omitidos da cena, mas não são excluídos do sistema, caso o observador mova-se em direção a eles e a distância seja reduzida a ponto de serem exibidos eles passam a aparecer na cena.

- **Fator de qualidade** - Esse fator define a qualidade dos objetos "árvore" da cena em função da distância que o observador encontra-se dos objetos, sendo que, quanto mais perto maior a qualidade e quanto mais distante menor a qualidade, ele é integrado com o item anterior para ampliar os resultados de otimização do sistema.

- **Redução da resolução** - Esse parâmetro não está ligado a distância e sim a resolução da tela do computador. Ele é ativado e desativado pelo parâmetro "FullScreen", quando ativado a resolução da tela do usuário passa a ser de *800x600*, o que facilita o processamento gráfico, pois serão menos pixels por cada unidade de área da tela.

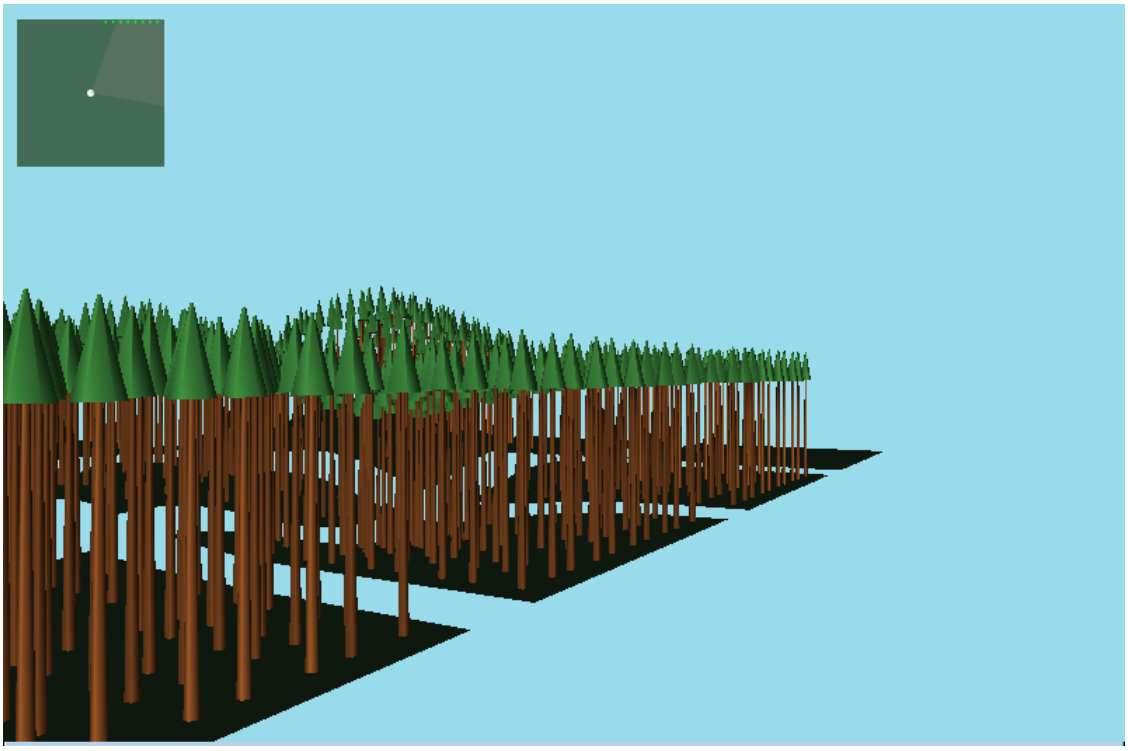

Figura 4.35: A direita ao fundo pode ser observado um "talão" que não contém todas as árvores, aquele é o limite imposto pelo "Fator de visualização".

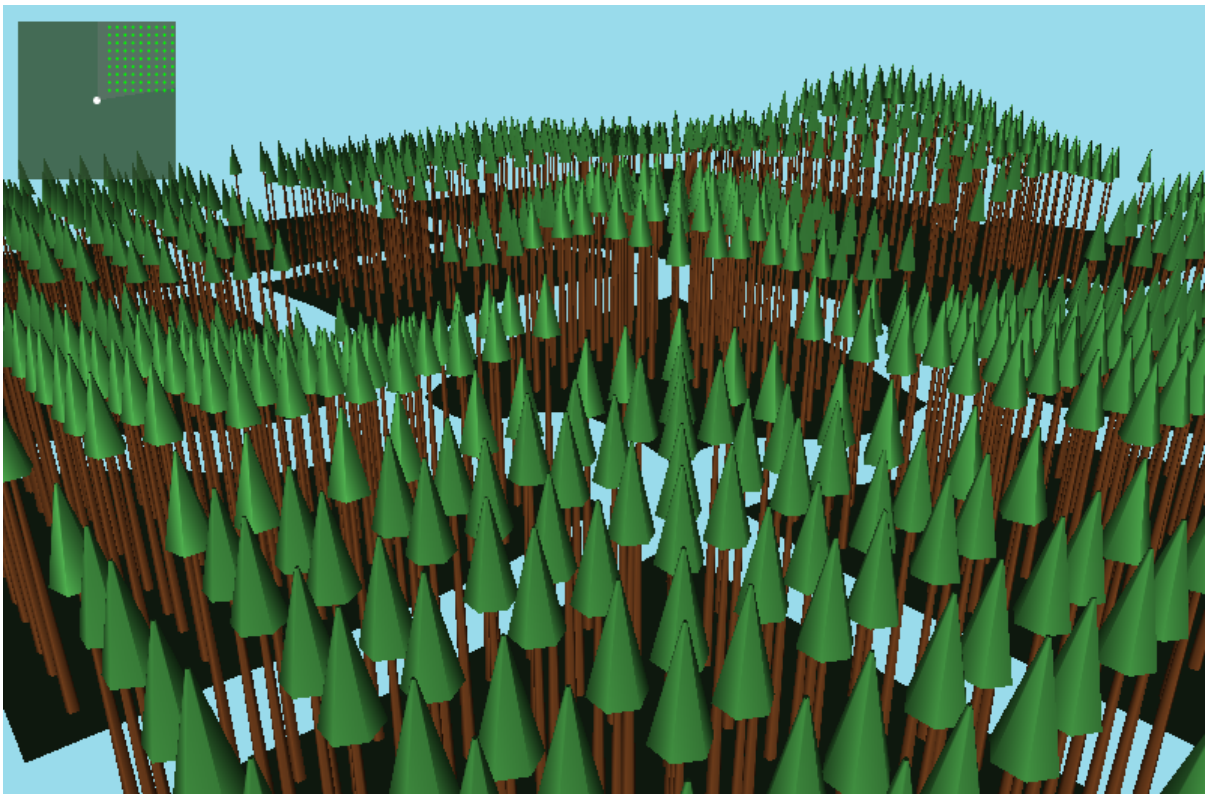

Figura 4.36: Exemplificação do "Fator de qualidade". No centro da imagem podem ser observadas árvores com aspecto triangular.

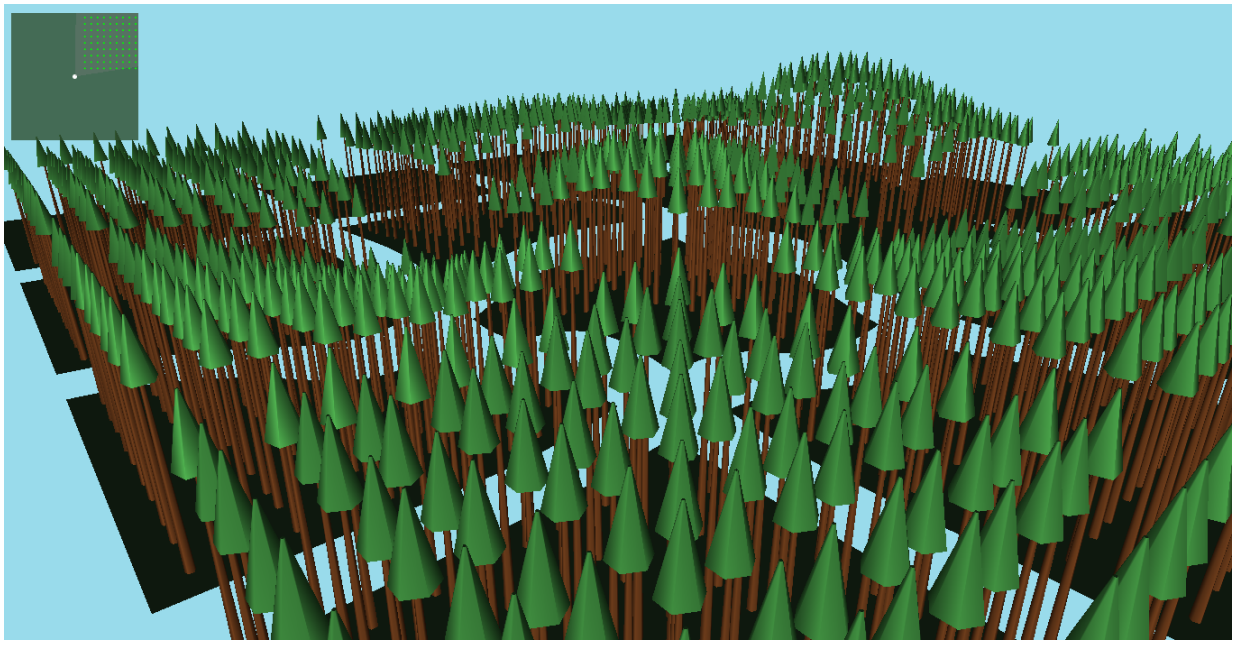

Figura 4.37: Tela normal do programa, com a resolução máxima suportada pelo monitor.

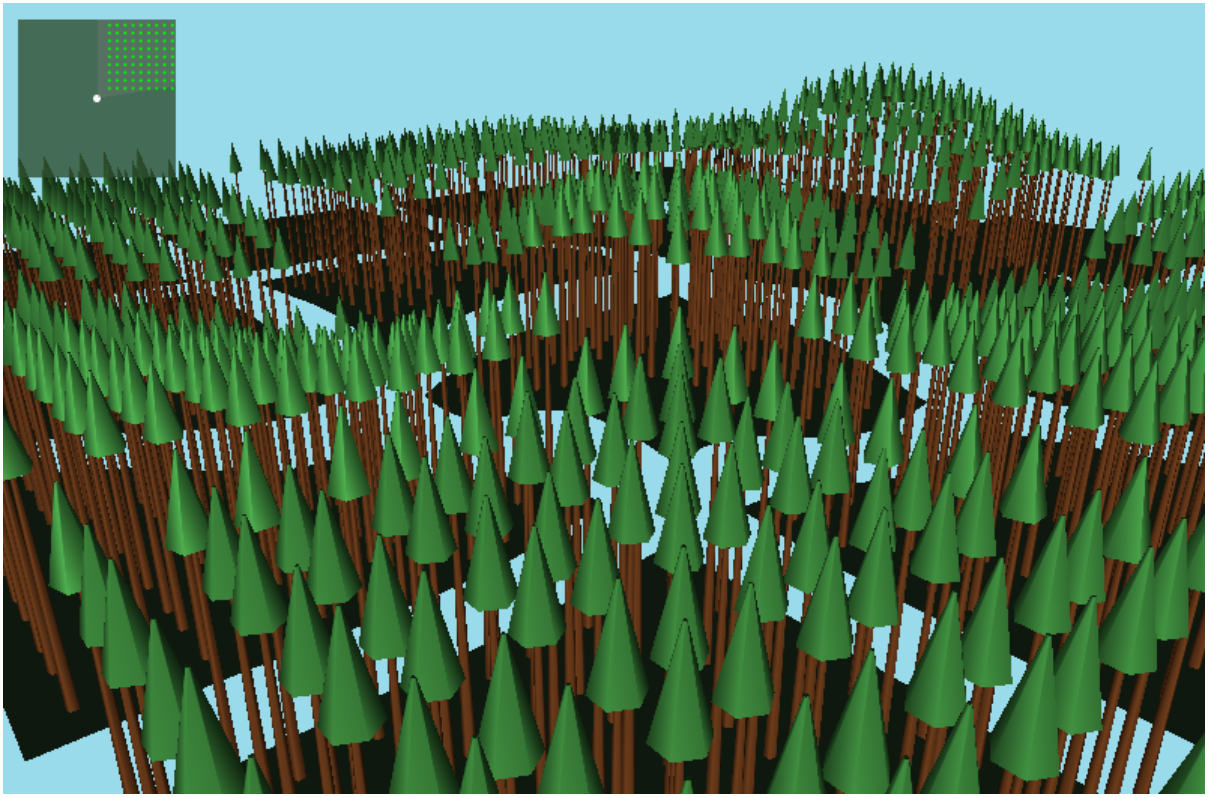

Figura 4.38: Tela em modo *"FullScreen"*. Resolução reduzida obrigatoriamente para 800x600.

Conclusão

O trabalho realizado teve como objetivo a construção de um "Ambiente de Simulação de uma Floresta Tridimensional" para atender as necessidades do Centro Técnico de Formação de Operadores de Máquinas Florestais (CTFlor) por uma ferramenta que fosse melhor que a utilizada atualmente. A partir disto foi proposto a realização da construção de um novo ambiente de simulação que atendesse a essas necessidades. O aplicativo desenvolvido atendeu a todos os requisitos iniciais propostos:

- Simulador mais realista;
- Que permitisse ao usuário navegar por um cenário;
- Cenário possuísse um relevo;
- E o relevo fosse editável;
- Bem como os parâmetros de dimensionam a floresta;

Proporcionou a pesquisa e a análise de uma ferramenta para a elaboração desta aplicação, a API de programação gráfica *OpenGL*, de forma a ampliar as percepções sobre computação gráfica e o ambiente tridimensional computadorizado. Foi possível implementar de forma satisfatória o software proposto, atendendo ao escopo determinado.

A ferramenta desenvolvida foi capaz de gerar a visualização do ambiente florestal, conforme especificações, e possibilitou a total interação do usuário com o ambiente. Ele permite a ele navegar pelo cenário gerado e visualizar todos os pontos do espaço renderizado pela aplicação. É possível, também, o usuário customizar o ambiente conforme achar mais satisfatório, ele pode alterar diversos parâmetros das árvores e do relevo.

Perante dados colhidos na execução da aplicação, constatou-se que, ela é capaz de gerar em um computador pessoal de bom rendimento (nos parâmetros atuais, uma máquina com processador de 4 núcleos a uma frequência de 2,0 GHz com 4 Gigabytes de memória RAM e sem placa de vídeo dedicada) um cenário bem grandioso. Quantitativamente foi possível gerar uma floresta composta de 1.000.000 de árvores e 10.000 talhões (terrenos). Neste caso cada

talhão tinha 25 metros quadrados e 100 árvores, divididas em 10 fileiras com 10 árvores com o espaçamento uniforme de 1,5 metros. Entre talhões existe um espaço de 5 metros. Ao fim do teste obtivemos uma floresta com 2995 metros quadrados ou seja aproximadamente 3 hectares de terra processados.

Conforme citado ao longo do trabalho, esta ferramenta tem poder de processamento para gerar gráficos 3D muito avançados. Todas as afirmações constatadas, em geral na mídia, apontam a *OpenGL* como uma ferramenta muito poderosa. Conforme esperado ela é utilizada nas mais diversas áreas de animação como: filmes, desenhos, animações computadorizadas, simuladores, etc. Diversos sites especializados e revistas citam a *OpenGL* como a ferramenta preferencial do mercado de animações, além de destacar a todo momento o seu carácter de ferramenta livre, segundo sua licença de utilização (SHREINER, 2009). Isso torna simples e prático para qualquer pessoa desenvolver suas aplicações, utilizando-a como base de desenvolvimento, como foi feito neste trabalho. Ainda destacando as qualidades observadas desta ferramenta, pode-se concluir que o seu poder de processamento gráfico é muito avançado, pois na aplicação proposta, mesmo sem utilizar a aceleração gráfica de placas de vídeo, foi possível gerar visualizações de cenários com alto grau de densidade de informações, objetos e elementos gráficos, sendo assim, a princípio, não existe limitação quanto a renderização dos cenários, a única limitação existente é relativa ao hardware utilizado.

Em face do material apresentado, o comportamento do software se mostrou melhor ou mais barato em comparação aos existentes. Entretanto a aplicação ficou restrita a apresentar os o ponto inicial de um simulador, por não possuir componentes avançados de processamento gráfico nem mesmo aprimoramentos de visualização com efeitos visuais avançados para gerar um ambiente ainda mais realista. Ela representa o ponto inicial para a concepção de um simulador de ambiente florestal realmente condizente com a realidade das aplicações atuais. Ainda é necessário a implementação de opções avançadas, tanto no âmbito dos gráficos e visualizações geradas, como no âmbito da interação do usuário com o ambiente.

Exemplos dessas implementações seriam:

1. Possibilidade de carregar para o programa arquivos contendo especificações de modelos avançados para as árvores;
2. Câmera de visualização com os objetos do cenário;
3. Aplicação de texturização de objetos e cenário;
4. Transformações geométricas de movimentação mais aprimoradas e avançadas;

5. Melhoramento do cenário de fundo que compões a visualização;

6. Aprimoramento das ferramentas otimização do processamento da aplicação;

7. Integração do processamento central do computador com a unidade de processamento de vídeo disponível;

Referências Bibliográficas

BICHO, A. et al. Programação gráfica 3d com opengl, open inventor e java 3d. Março 2002.

BLANCHETTE, J.; SUMMERFIELD, M. *C++ GUI Programming with Qt 4*. 2ª. ed. New York, NY, USA: Prentice Hall. Inc, 2008.

BONA, C. Avaliação de processos de software: Um estudo de caso em xp e iconix. Florianópolis, SC - Brasil, 2002. Dissertação de mestrado.

CLUA, E. W. G.; BITTENCOURT, J. R. Desenvolvimento de jogos 3d: Concepção, design e programação. *XXIV Jornada de Atualização em Informática (JAI) Part of XXIV Congresso da Sociedade Brasileira de Computação*, p. 22–29, 2005.

ECKEL, B. *Thinking in Java*. 4ª. ed. New York, NY, USA: Prentice Hall. Inc, 2006.

EZUST, A.; EZUST, P. *Introduction to Design Patterns in C++ with Qt*. 2ª. ed. New York, NY, USA: Prentice Hall. Inc, 2011.

FALCãO, E. de L.; MACHADO, L. dos S.; COSTA, T. K. de L. Programando em x3d para integração de aplicações e suporte multiplataforma. In: L.S. MACHADO AND R.A SISCOUTTO. *Tendências e Técnicas em Realidade Virtual e Aumentada*. Porto Alegre, RS. Brasil: SBC, 2010. cap. 2, p. 35–63.

GALITZ, W. O. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques*. 3ª. ed. Hoboken, NJ. USA: Wiley Publishing . Inc, 2007.

GNU. *Definicao de Software Livre*. 2012. [Acessado em 17 Jan 2012]. Disponível em: <http://www.gnu.org/philosophy/free-sw.html>.

HORSTMANN, C. *Padrões e projetos orientados a objetos*. 2ª. ed. Porto Alegre, RS, Brasil: Bookman editora S.A, 2006.

MANSSOUR, I. H.; COHEN, M. Introdução a computação gráfica. *Anais do Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, v. 13, n. 2, p. 1–25, 2006.

MARTIN, R. C. Uml tutorial: Finite state machines. *C++ Report - Engineering Notebook Column*, v. 10, n. 5, p. 316–319, Junho 1998.

OPENGL. *History of OpenGL*. 2012. [Acessado em 20 Jan 2012]. Disponível em: <http://www.opengl.org/wiki/Version>.

PARC, X. *Xerox PARC history: Graphical User Interface(GUI)*. 2012. [Acessado em 16 Jan 2012]. Disponível em: <http://www.parc.com/about/>.

SANTOS, E. T. et al. Computação gráfica: Estado da arte e a pesquisa na usp. *SIMPÓSIO NACIONAL DE GEOMETRIA DESCRITIVA E DESENHO TÉCNICO*, v. 15, p. 1333–1363, 2003.

SGI. *OpenGL*. 2012. [Acessado em 16 Jan 2012]. Disponível em: <`http://www.sgi.com/products/software/opengl/`>.

SHREINER, D. *OpenGL, Programming Guide*. 7ª. ed. Boston, Massachussets: Addison-Wesley. Inc, 2009. 857 p.

SOARES, M. dos S. Comparaçao entre metodologias ageis e tradicionais para o desenvolvimento de software. *INFOCOMP Journal of Computer Science*, v. 3, n. 2, p. 8–13, 2004.

VASCO, C. G.; VITHOFT, M. H.; ESTANTE, P. R. C. Comparação entre metodologias rup e xp. Curitiba, PR - Brasil, 2005.

ANEXO A – Comandos

Esses são os comandos possíveis no simulador:

Tecla	**Função da tecla na aplicação**
Seta para cima	De acordo com o parâmetro "rotate"acrescenta X graus no angulo de visualização no eixo Z;
Seta para baixo	De acordo com o parâmetro "rotate"diminui X graus no angulo de visualização no eixo Z;
Seta para esquerda	De acordo com o parâmetro "rotate"acrescenta X graus no angulo de visualização no eixo X;
Seta para direita	De acordo com o parâmetro "rotate"diminui X graus no angulo de visualização no eixo X;
W	Movimenta a visão na direção da visualização em X unidade de acordo com o parâmetro "velox";
A	Movimenta a visão na direção contraria da visualização(ré) em X unidade de acordo com o parâmetro "velox";
S	Movimenta a visão lateralmente para a esquerda em X unidade de acordo com o parâmetro "velox";
D	Movimenta a visão lateralmente para a direita em X unidade de acordo com o parâmetro "velox";
PageUp	Acrescenta X unidades ao parâmetro que estipula a altura(Eixo Z) da visualização de acordo com o parâmetros "velox";
PageDown	Diminui X unidades ao parâmetro que estipula a altura(Eixo Z) da visualização de acordo com o parâmetros "velox";
M	Mostra/Esconde o minimapa da tela;
F	Coloca/Retira a aplicação do modo FullScreen(tela cheia);
X	Para o objeto "sol"para que a cena fique sempre clara;
C	Continua o tempo normal, havendo períodos de dia e noite;
ESC	Acessa os menus de contexto que permitem editar o ambiente de simulação(editar os parâmetros);

Tabela A.1: Teclas do sistema e suas funcionalidades