**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

# VISEDU-MAT: VISUALIZADOR DE MATERIAL EDUCACIONAL, MÓDULO DE MATEMÁTICA

**JOSÉ RICARDO KRAUSS**

**BLUMENAU**
**2013**

**2013/2-13**

**JOSÉ RICARDO KRAUSS**

# VISEDU-MAT: VISUALIZADOR DE MATERIAL EDUCACIONAL, MÓDULO DE MATEMÁTICA

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, M. Sc. - Orientador

**BLUMENAU**
**2013**

**2013/2-13**

# VISEDU-MAT: VISUALIZADOR DE MATERIAL EDUCACIONAL, MÓDULO DE MATEMÁTICA

Por

**JOSÉ RICARDO KRAUSS**

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

| | |
|---|---|
| | _____ |
| Presidente: | Prof. Dalton Solano dos Reis, M. Sc. – Orientador, FURB |
| | _____ |
| Membro: | Prof. Joyce Martins, Mestre – FURB |
| | _____ |
| Membro: | Prof. Antonio Carlos Tavares, Mestre – FURB |

Blumenau, 10 de dezembro de 2013

Dedico este trabalho aos meus familiares e todos os amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

## AGRADECIMENTOS

Agradeço primeiramente a minha família, por todo o apoio necessário durante toda a jornada deste curso, desde o seu inicio até o presente momento.

Ao meu orientador, Dalton Solano dos Reis, por ter acreditado e depositado confiança mesmo quando nem eu acreditava na conclusão deste trabalho.

Ao professor Evandro Felin Londero, pelas contribuições no desenvolvimento deste trabalho.

A todos os meus amigos, pelo companheirismo e por fazer convites irrecusáveis nas horas mais inconvenientes do desenvolvimento deste trabalho.

A imaginação é mais importante que o conhecimento.

Albert Einstein

# RESUMO

Este trabalho apresenta o desenvolvimento de uma aplicação web para visualização de funções matemáticas explícitas usando WebGL. A aplicação tem como principal objetivo a criação, validação e exibição de uma função em um espaço delimitado por um elemento HTML5 canvas. Este espaço compreende uma área tridimensional (3D) ou bidimensional (2D) permitindo que o usuário use nas funções as variáveis de domínio x, y, z para 3D ou x e y para 2D. Para criação dos objetos gráficos é usada a biblioteca Three.js que abstrai a complexidade da implementação WebGL. A aplicação ainda permite que o usuário interaja com o objeto gráfico previamente selecionado, podendo alterar sua cor, material e tamanho. Também possibilita o usuário visualizar algumas funções matemáticas usando um material de exemplos didáticos.

Palavras-chave: Computação gráfica. Funções matemáticas. WebGL. HTML5.

# ABSTRACT

This paper presents the development of a web application for visualizing explicit mathematical functions using WebGL. The application's main objective is the creation, validation and display of a function in a space delimited by a HTML5 canvas element. This space area comprises a three-dimensional (3D) or two dimensional (2D) allowing the user to use in the functions the field's variables x, y, z for 3D or x, y for 2D. For creation of graphic objects is used to Three.js library that abstracts the complexity of implementing WebGL. The application also allows the user to interact with the previously selected graphic object, altering its color, material and size. It also allows the user to visualize some mathematical functions using a didactic material.



Keywords: Computer graphics. Mathematical functions. WebGL. HTML5

# LISTA DE ILUSTRAÇÕES

# LISTA DE SIGLAS

2D – 2 Dimensões

3D – 3 Dimensões

ADUBOGL – Aplicação Didática Usando a Biblioteca OpenGL

API – *Application Programming Interface*

ASCII – *American Standard Code for Information Interchange*

CSS –*Cascading Style Sheets*

FPS – *Frames Per second*

GLSL – o*penGL Shading Language*

HTML – *Hyper Text Markup Language*

HTML5 – *Hyper Text Markup Language*, versão 5

IDE – *Integrated Development Environment*

MathML – *Mathematical Markup Language*

OPENGL ES – *OpenGL for Embedded Systems*

RAM – *Random Access Memory*

RF – Requisito Funcional

RNF – Requisito Não Funcional

SVG - *Scalable Vector Grapchis*

UML – *Unified Modeling Language*

UC – *Use Case*

WebGL – *Web Graphics Library*

W3C – *World Wide Web Consortium*

WHATWG – *Web Hypertext Application Technology Working Group*

# SUMÁRIO

# 1 INTRODUÇÃO

A matemática é uma disciplina utilizada em nossa vida todos os dias e em diversas situações. Mas, apesar de sua importância, a matemática sempre foi considerada uma disciplina complexa e de difícil compreensão por grande parte dos alunos (MALAQUIAS, 2012).

O ensino de funções pode ser citado como um exemplo dessa dificuldade de compreensão, uma vez que nem sempre se consegue achar um modelo na vida real, no qual possa ser assimilada a função exposta.

Os educadores de matemática enfrentam os desafios impostos pela profissão no sentido de responder a seguinte questão: como conduzir o ensino de matemática de forma a contribuir para formar sujeitos competentes, criativos, autônomos e capazes de resolver problemas do mais diversos contextos? Grande esforço tem sido feito no sentido de atribuir significados aos conteúdos matemáticos e ao mesmo tempo motivar o aluno, despertando neste o gosto pelo saber. Os educadores lançam mão de artifícios que proporcionem uma interação do aluno com os objetos. Neste contexto, a informática tem sido uma aliada nas escolas como elemento complementar no ensino de inúmeras matérias (CARRIJO, 2007).

No contexto da informática, um dos principais softwares responsáveis por prover essa enorme facilidade de acesso foram os navegadores de internet, fornecendo toda a acessibilidade dos sistemas web, uma vez que várias plataformas hoje em dia são capazes de interpretar código *HyperText Markup Language* (HTML), desde típicos computadores até celulares e *tablets* (GROSSKURTH; GODFREY, 2006). Mais recentemente observa-se que os navegadores também permitem visualizar representações em 3D utilizando recursos gráficos através da *Application Programming Interface* (API) *Web Graphics Library* (WebGL).

A tecnologia computacional, é um recurso que pode ser utilizado como alternativa para auxiliar instrumentos usados no cotidiano escolar (quadro, caderno, livro etc) e que deveria atender a esses alunos no âmbito escolar, promovendo a inclusão dos mesmos, independente de suas diferenças (MALAQUIAS, 2012).

Diante do exposto, com o intuito de auxiliar no ensino de funções matemáticas, este trabalho demonstra um visualizador gráfico de funções para a web, usando a API WebGL, relacionando assim as funções com os recursos da computação gráfica e a facilidade de acesso de um sistema web.

## 1.1 OBJETIVOS DO TRABALHO

Este trabalho tem como objetivo principal desenvolver um visualizador gráfico de funções matemáticas para web, utilizando a API WebGL.

Os objetivos específicos do trabalho são:

a) visualizar a representação da função digitada em um espaço gráfico 2D e 3D;
b) controlar os movimentos das câmera para poder explorar a representação gerada, bem como inspecionar valores desta representação;
c) utilizar efeitos de cores e iluminação para facilitar a visualização da representação gerada.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos, sendo que o primeiro é feito uma introdução ao tema e em seguida são apresentados os objetivos e a estrutura do trabalho.

O segundo capítulo contempla a fundamentação teórica que auxilia no entendimento deste trabalho.

O capítulo três contém o desenvolvimento da aplicação, onde são apresentados os requisitos da especificação, os casos de uso elaborados e também os diagramas de pacotes, de classes e de sequência. Neste capítulo são apresentadas também as ferramentas e técnicas utilizadas no desenvolvimento e, por último, é mostrada a operacionalidade da aplicação.

O quarto e último capítulo expõe os resultados obtidos e as conclusões, finalizando com sugestões de melhorias e extensões para projetos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é conceituado o HTML e em seguida é apresentado o HTML5 e algumas de suas novas funcionalidades. A seção 2.2 apresenta a principal biblioteca gráfica deste trabalho, que é WebGL. Na seção 2.3 é feita uma rápida abordagem sobre equações matemáticas e, por último, a seção 2.4 descreve alguns trabalhos correlatos ao assunto da proposta.

## 2.1 HTML5

HTML é uma linguagem de marcação para descrever páginas web. Ela possui um conjunto de *tags* que servem para descrever o conteúdo de uma página web, também chamada de documento HTML.

Uma *tag* de marcação HTML é uma palavra-chave da linguagem cercada pelos símbolos de maior e menor que possui uma função qualquer, como formatação de textos e tabelas. Os navegadores web têm como propósito ler um documento HTML e exibi-lo na forma de página web. Portanto, um navegador web não exibe as *tags*, ele as usa para interpretar, de forma correta, o conteúdo da página (W3SCHOOLS, 2013a).

A versão 5 da linguagem será o novo padrão HTML, também chamada de HTML5, nascido em 2006 de uma cooperação entre duas grandes empresas, a *World Wide Web Consortium* (W3C) e a *Web Hypertext Application Technology Working Group* (WHATWG) (W3SCHOOLS, 2013b).

Um dos principais objetivos do HTML5 é facilitar a manipulação dos elementos possibilitando o desenvolvedor modificar as características dos objetos de forma não intrusiva e de maneira que seja transparente para o usuário final. Ao contrário das versões anteriores, o HTML5 fornece ferramentas para o *Cascading Style Sheets* (CSS) e o JavaScript.

O HTML5 permite por meio de suas APIs a manipulação das características destes elementos, de forma que o website ou a aplicação continue leve e funcional. O HTML5 também cria novas *tags* e modifica a função de outras. As versões antigas do HTML não continham um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, *sidebar*, menus, entre outros. Não havia um padrão de nomenclatura de *ids*, classes ou *tags*. Também não havia um método de capturar de maneira automática as informações localizadas nos rodapés dos websites. O HTML5 modifica a forma como se escreve código e como se organizam as informações na página. Seria mais semântica com menos código sem perda de performance e sem a necessidade de instalação de *plug-ins* extras (W3C BRASIL, 2013).

As características mais marcantes da nova versão da linguagem começam pela estrutura básica, como a instrução *DOCTYPE*. Em HTML5 há somente uma forma de declará-lo que é da seguinte forma `<!DOCTYPE html>`. Outra diferença é a *metatag charset* que em HTML5 declara-se assim `<meta charset="utf-8">`, além da facilidade de tornar "arrastável" um elemento da página, somente adicionando no elemento o atributo `draggable="true"` (MICROSOFT, 2013).

Algumas das outras novas características do HTML5 são:

a) o elemento `<canvas>` para desenho 2D;
b) os elementos `<video>` e `<audio>` para reprodução de midias;
c) suporte a armazenamento local;
d) novos elementos de conteúdo específico, como `<article>`, `<footer>`, `<header>`, `<nav>`, `<section>`;
e) novos formulários de controle como `calendar, date, time, email, search`;
f) novos elementos para formulários como `<datalist>`, `<keygen>` e `<output>`.

Outra característica evidente foi a remoção de alguns elementos da versão anterior como: `<acronym>`, `<applet>`, `<basefont>`, `<big>`, `<center>`, `<dir>`, `<font>`, `<frame>`, `<frameset>`, `<noframes>`, `<strike>` e `<tt>`.

### 2.1.1 Javascript

JavaScript é uma linguagem de programação interpretada com capacidades orientadas a objetos. Sintaticamente o núcleo da linguagem assemelha-se com linguagens como C, C++ e Java (FLANAGAN, 2008). Tal semelhança termina por aí, uma vez que a linguagem é fracamente tipada, o que significa que suas variáveis não precisam ter um tipo especificado, diferente das outras linguagens citadas.

Objetos em JavaScript mapeiam nomes de propriedades em valores de propriedades arbitrários. Neste caso, eles são mais parecidos com tabelas hash e vetores associativos (em Perl) do que parecidos com struct (em C) e objetos (em Java e C++) (FLANAGAN, 2008).

JavaScript é mais comumente usada em navegadores web e nesse contexto o núcleo do propósito geral é estendido com objetos que permitem usar scripts para interagir com os usuários, controlar páginas web e alterar o conteúdo de documentos que aparecem nas janelas dos navegadores web.

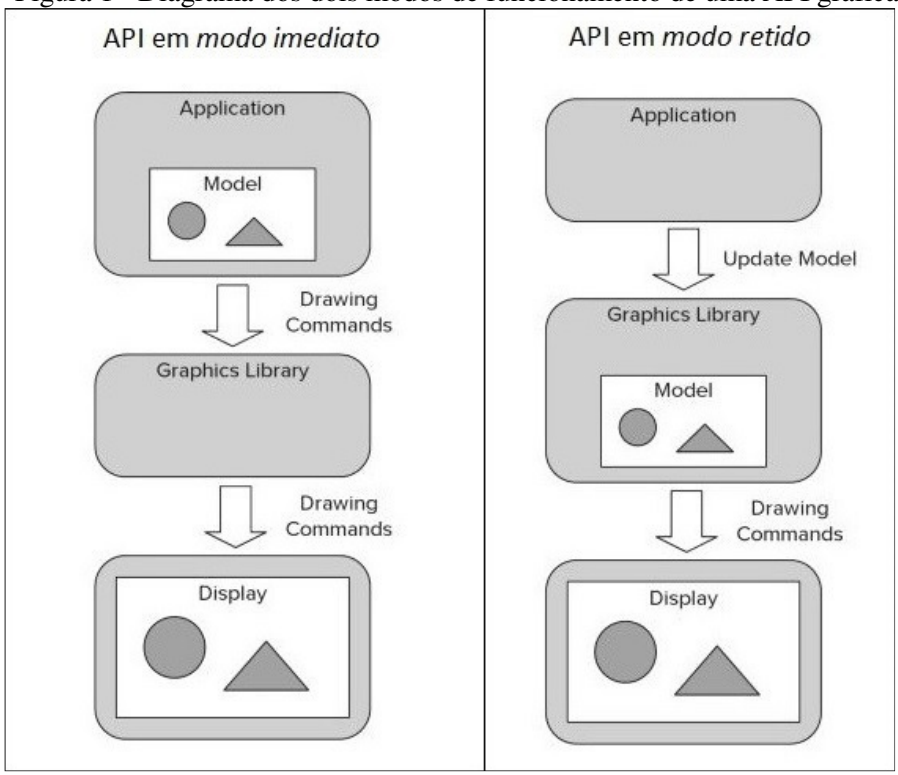
## 2.2 WEBGL

WebGL é uma API para renderização 3D de modo imediato desenvolvida para web em um contexto HTML, derivada da biblioteca gráfica *OpenGL for Embedded Systems* (OpenGL ES) 2.0, que é uma API livre e multi-plataforma para funções gráficas 2D e 3D em sistemas embarcados, baseada no sistema gráfico do OpenGL 2.0, mas concebida principalmente para hardware gráficos rodando em dispositivos móveis e embarcados. A API WebGL foi desenvolvida basicamente para renderizar conteúdo do elemento `<canvas>` do HTML5 (KHRONOS, 2011).

Uma aplicação web que usa WebGL consiste basicamente de arquivos HTML, CSS e JavaScript que executam diretamente no navegador. E também alguns tipos de dados que representam modelos 3D que exibe. WebGL é suportada nativamente pela maioria dos navegadores web (Firefox, Chrome, Safari) e é através do JavaScript que ocorre a chamada para a API WebGL, que envia informações para seu *pipeline* de como os modelos 3D devem ser renderezidados (ANYURU, 2012, p. 7).

Segundo Anyuru (2012, p. 3), existem duas maneiras fundamentalmente diferentes de desenvolver uma API gráfica, usando modo imediato ou retido, conforme a Figura 1.

Figura 1 - Diagrama dos dois modos de funcionamento de uma API gráfica




Fonte: Anyuru (2012, p. 4).

Para uma API de modo imediato, toda a cena precisa ser redesenhada a cada *frame*, independente de ela ter sido ou não alterada. A biblioteca gráfica que expõe a API não salva

nenhum modelo interno da cena que deveria ser desenhada. Em vez disso, a aplicação precisa ter sua própria representação da cena que é mantida em memória.

Uma biblioteca gráfica que expõe uma API em modo retido contém um modelo interno ou um grafo de cena com todos os objetos que devem ser renderizados. Quando a aplicação chama a API, o modelo interno é atualizado e a biblioteca pode então decidir quando o desenho atual deve ser feito na tela. Isso significa que a aplicação que usa a API não precisa emitir comandos de desenho para desenhar a cena completa em todos os *frames*. Um exemplo de API em modo retido é a *Scalable Vector Grapchis* (SVG) (ANYURU, 2012, p. 4).

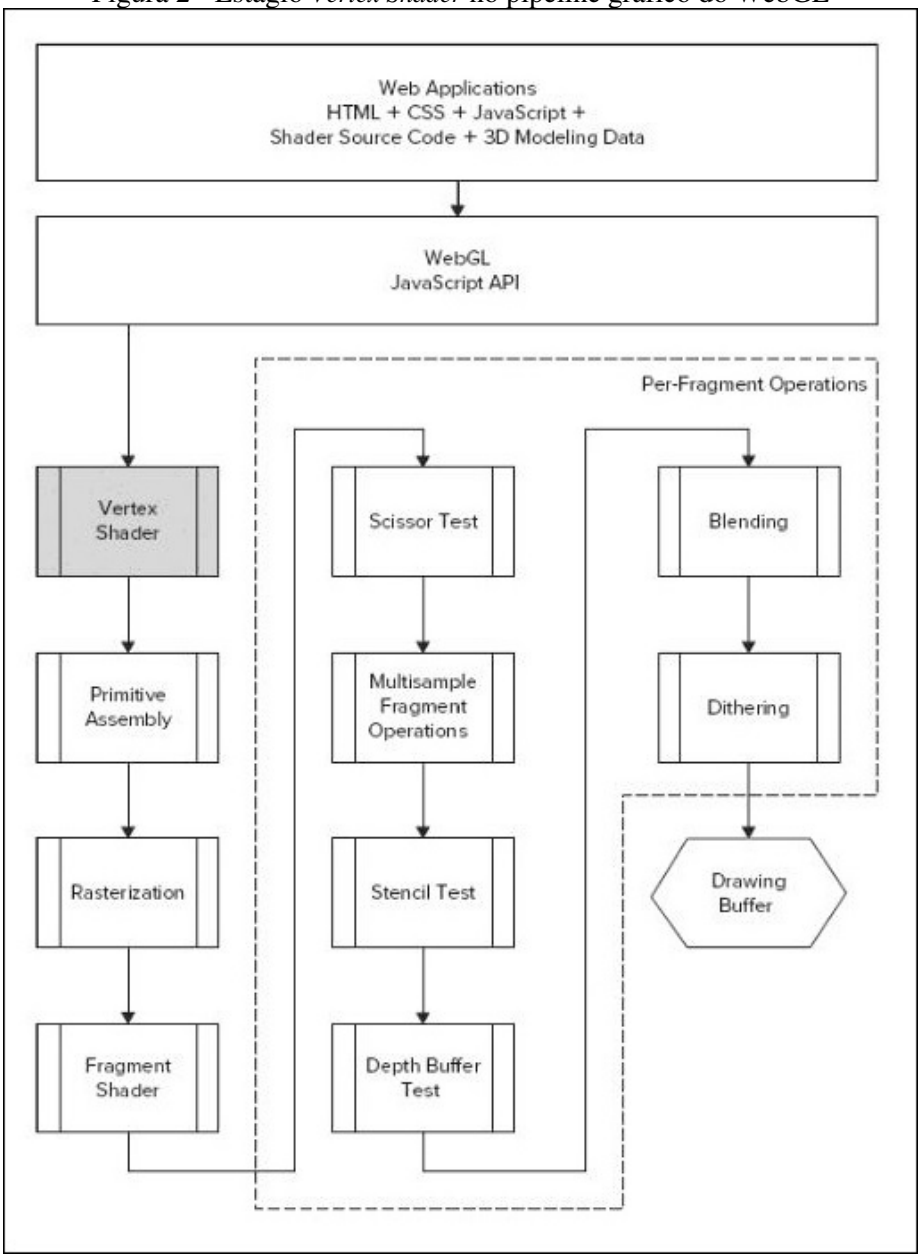
### 2.2.1 *Pipeline* gráfico WebGL

Para obter cenas realísticas em 3D, não é suficiente renderizar objetos em determinadas posições. É preciso também levar em conta como o objeto ficará quando fontes de luzes brilharem sobre ele. O termo geral usado para o processo que determina os efeitos da luz em diferentes materiais é chamado de *shading* (ANYURU, 2012, p. 8).

Em WebGL, o processo de *shading* é feito em dois estágios principais conhecidos como *Vertex Shader* e *Fragment Shader*. O primeiro estágio é o *Vertex Shader* no qual executa o *shading* para um vértice. O código fonte do *Vertex Shader* é escrito em *OpenGL ES Shading Language* (GLSL) (ANYURU, 2012, p. 9). A Figura 2 mostra onde o *Vertex Shader* se encontra no *pipeline* gráfico WebGL.

O *Vertex Shader* é onde os dados da modelagem 3D, como os vértices por exemplo, são processados pela primeira vez depois de serem enviados através da API JavaScript. Antes do *Vertex Shader* iniciar, ele geralmente transforma o vértice multiplicando-o por uma matriz de transformação. Os dados de entrada do *Vertex Shader* consistem basicamente em: variáveis do tipo *attribute* que normalmente contém dados específicos para cada vértice (por exemplo, cor ou posição de um vértice), variáveis do tipo *uniform* que contém dados comuns a todos os vértices (por exemplo, matrizes de transformações) e o código fonte que consiste o *Vertex Shader* escrito em GLSL (ANYURU, 2012, p. 10). O Quadro 1 mostra um exemplo de código fonte escrito em GSLS que contempla os dados básicos de entrada do *Vertex Shader*.

Figura 2 - Estágio *Vertex Shader* no pipeline gráfico do WebGL




Fonte: Anyuru (2012, p. 9).

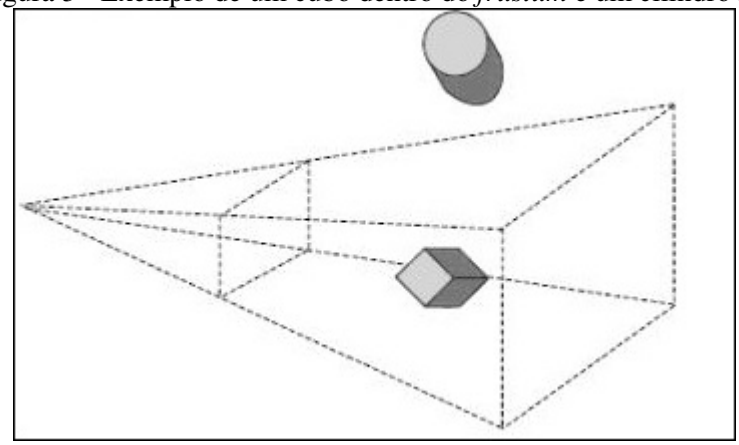
Quadro 1 - Exemplo de código fonte do *Vertex Shader*

```
attribute vec3 vertexPosition;
attribute vec3 vertexColor;

uniform mat4 uMVMatrix;
uniform mat4 uPMMatrix;

varying vec4 vColor;

void main(){
  gl_Position = uMVMatrix * uPMMatrix * vec4(vertexPosition, 1.0);
  vColor = vertexColor;
}
```

Fonte: Anyuru (2012, p. 11).

Entre o primeiro e o segundo estágio existem duas etapas fundamentais no processo de *shading*. A primeira etapa é chamada de *Primitive Assembly* (Montagem primitiva). Nessa etapa o WebGL precisa montar os vértices processados em primitivas geométricas individuais como triângulos, linhas ou pontos. Então para cada triângulo, linha ou ponto ele precisa saber se a primitiva está na região 3D visível na tela naquele momento. Essa região também é conhecida como *frustum* e se a primitiva está dentro dessa região então é enviada para a próxima etapa do processo, caso contrário ela é removida (ANYURU, 2012, p. 13). A Figura 3 mostra o exemplo de um cubo dentro do *frustum* e um cilindro na parte de fora.

Figura 3 - Exemplo de um cubo dentro do *frustum* e um cilindro fora

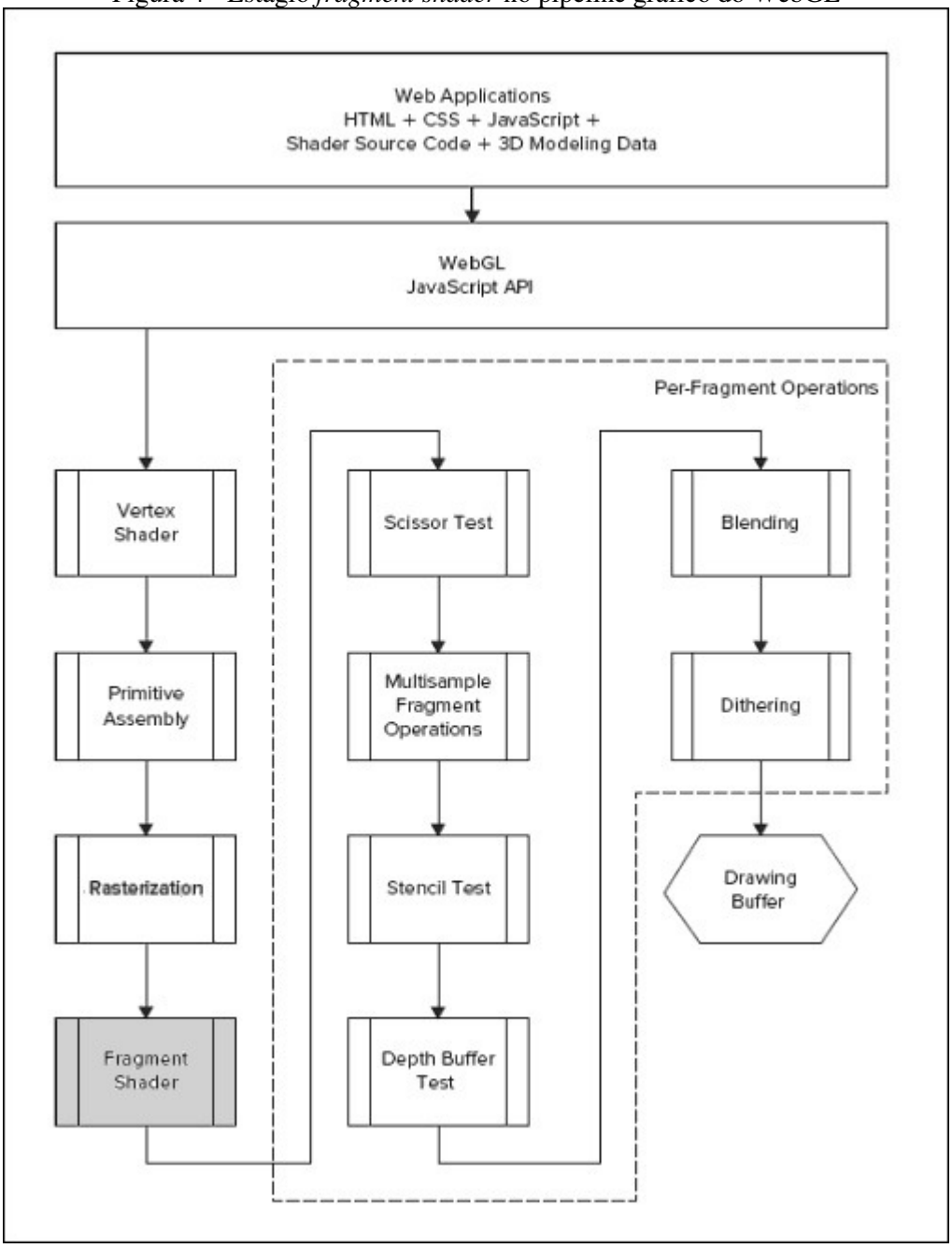


Fonte: Anyuru (2012, p. 13).

A segunda etapa entre os estágios principais do processo de *shading* é conhecida como *Rasterization* (Rasterização). Nesta etapa as primitivas recém processadas são convertidas em fragmentos que serão enviados para o *Fragment Shader*. Nessa etapa um fragmento é equivalente a um pixel que será desenhado na tela (ANYURU, 2012, p. 14).

O segundo estágio principal do processo de *shading* é o *Fragment Shader.* Nesse estágio são processados os fragmentos vindos da etapa de rasterização. Os dados de entrada do *Fragment Shader* consistem de algumas variáveis especiais embutidas no WebGL por exemplo, `gl_PointCoord`, as variáveis definidas pelo usuário no *Vertex Shader*, variáveis do tipo *uniform,* e também o código fonte do *Fragment Shader* escrito em GLSL (ANYURU, 2012, p. 16). A Figura 4 mostra o estágio *Fragment Shader* no pipeline gráfico WebGL.

Figura 4 - Estágio *fragment shader* no pipeline gráfico do WebGL




Fonte: Anyuru (2012, p. 17).

Depois de processados, cada fragmento passa por uma série de sub-etapas onde são feitos alguns testes e ajustes até que seja enviado para o *Drawing Buffer* para ser desenhado na tela. A primeira etapa é *Scissor Test* que determina se o fragmento em questão está dentro da área visível definida. A próxima etapa é *Multisample Fragment Operations* que modifica o valor alpha do fragmento através da técnica *anti-aliasing* que torna as arestas e vértices mais suaves na tela. Após essa operação, ocorre a *Stencil Test* que verifica se o fragmento deve ou não ser descartado.

Na sequência ocorre o *Depth Buffer Test* que é responsável por determinar quais pixels devem ser exibidos na tela quando ocorre a sobreposição de fragmentos. Essa verificação é controlada através do z-buffer; fragmentos com valores menores estão mais perto da tela, portanto devem ser desenhados, enquanto os mais distantes devem ser descartados. Após esta etapa, ocorre a etapa *Blending*, que permite combinar a cor do fragmento atual com a cor de um fragmento já existente naquela mesma posição. Isso é o que permite tornar objetos transparentes. Por último, a etapa *Dithering* organiza as cores de uma maneira que se cria a ilusão de existirem mais cores do que realmente há (ANYURU, 2012, p. 19).

### 2.2.2 `Three.js`

`Three.js` é um biblioteca escrita em JavaScript, independente de navegador, usada para criar e exibir gráficos 3D em navegadores web. Ela permite a criação de animações 3D aceleradas por hardware gráfico sem a necessidade de instalação de *plug-ins* adicionais (WIKIPEDIA, 2013b).

Primeiramente seu código fonte foi desenvolvido em *ActionScript*, em seguida em 2009, foi portado para JavaScript. Sua primeira verão oficial foi liberada por Ricardo Cabello em um repositório on-line (*Git-Hub*) por volta de abril de 2010. O motivo para a transferência para JavaScript foi que não precisavam mais compilar o código fonte a cada vez que o executassem e também pelo fato de ser independente de navegador (WIKIPEDIA, 2013b).

A criação desta biblioteca só foi possível com o advento do WebGL, quando Paul Brunt adicionou facilmente um renderizador para a biblioteca, uma vez que o código fonte do renderizador da `Three.js` foi desenvolvido mais como um módulo do que como seu próprio núcleo.

Algumas das características da biblioteca incluem suporte a rederizadores Canvas, SVG e WebGL, adição e remoção de objetos na cena em tempo real, uso de câmeras ortográficas e perspectiva, controles de evento de mouse e teclado, uso de luzes ambientes e direcionais, captura e reprodução de sombra, uso de materiais básicos, sólidos e reflexivos, uso de *shaders* personalizados, objetos pré-definidos como malha, partículas, linhas e *sprite,* uso de geometrias pré-definidas como plano, cubo, esfera e cilindro (WIKIPEDIA, 2013b).

## 2.3 EQUAÇÕES MATEMÁTICAS

Segundo Garbi (2009, p. 1), “as equações – algébricas, exponenciais, diferenciais, trigonométricas ou de qualquer outra natureza – constituem, pelo menos do ponto de vista prático, a parte mais importante da matemática”. Qualquer problema que possa ser

solucionado através dos números certamente será tratado, direta ou indiretamente, por meio de equações. Resolver uma equação é, através do que ela expressa, encontrar alguma coisa que se desconhece e que se costuma denominar incógnita[1].

A solução de uma equação pode ser um ou mais números, mas pode também, ser a medida de uma grandeza física, como uma distância, um peso, um intervalo de tempo, entre outros. Resolver uma equação pode significar o encontro não de um número, mas de uma forma. Por exemplo, foi através da solução de certo tipo de equação que Newton provou que as órbitas dos planetas são elipses (GARBI, 2009, p. 3).

Equações algébricas são aquelas em que a incógnita aparece apenas submetida às chamadas operações algébricas. São elas: adição, subtração, multiplicação, divisão, potenciação inteira e radiciação (GARBI, 2009, p. 5). Por exemplo, as equações mostradas no Quadro 2, são todas algébricas.

Quadro 2 - Equações algébricas

| |
|---|
| $ax + b = c$ |
| $ax^2 + bx + c = 0$ |
| $mx^5 + \sqrt{7x^3} + k = 8$ |

Fonte: Garbi (2009, p. 5).

Já as equações exponenciais são todas aquelas em que a incógnita encontra-se no expoente (MATEMÁTICA DIDÁTICA, 2013). Conforme o Quadro 3.

Quadro 3 - Equações exponenciais

| |
|---|
| $2^x = 8$ |
| $7^x - 2 = 344$ |
| $3^{x+2} + 10 = 82$ |

Fonte: Matemática didática (2013).

Uma equação que contém as derivadas (ou diferenciais) de uma ou mais variáveis dependentes em relação a uma ou mais variáveis independentes é chamada de equação diferencial (ZILL, 2011, p. 2). Elas podem ser classificadas por tipo, ordem ou linearidade.

### 2.3.1 Funções Matemáticas

Função é um dos conceitos mais importantes da matemática. Existem várias definições, dependendo da forma como são escolhidos os axiomas. Pode ser definida como uma relação entre dois conjuntos, onde há uma relação entre cada um de seus elementos. Também pode ser uma lei que para cada valor `x` é correspondido por um elemento `y`, também denotado por `f(x)`. Cada função é definida por leis generalizadas e propriedades específicas.

[1] A letra `x` é a mais habitual representação das incógnitas, embora outras letras possam ser usadas.

Por causa de sua generalidade, as funções aparecem em muitos contextos matemáticos e muitas áreas da matemática baseiam-se no estudo de funções (WIKIPEDIA, 2013a).

O conceito de uma função é uma generalização da noção comum de fórmula matemática. As funções descrevem relações matemáticas especiais entre dois elementos. Intuitivamente, uma função é uma maneira de associar a cada valor do argumento `x` (às vezes denominado variável independente) um único valor da função `f(x)` (também conhecido como variável dependente). Isto pode ser feito através de uma equação, um relacionamento gráfico, diagramas representando os dois conjuntos, uma regra de associação, uma tabela de correspondência. Cada par de elementos relacionados pela função determina um ponto nesta representação, a restrição de unicidade da imagem implica um único ponto da função em cada linha de chamada do valor independente `x` (WIKIPEDIA, 2013a).

#### 2.3.1.1 Funções explícitas

O tipo de função mais comum é aquele onde o argumento e o valor da função são ambos numéricos, o relacionamento entre os dois é expresso por uma fórmula e o valor da função é obtido através da substituição direta dos argumentos. Em uma função explícita é fornecida uma prescrição para a determinação do valor de saída da função `y` em termos do valor de entrada `x` (WIKIPEDIA, 2013a).

O Quadro 4 mostra o exemplo de uma função explícita que associa cada valor de `x` o seu quadrado.

Quadro 4 - Função explícita

$$f(x) = x^2$$

Fonte: Wikipedia (2013)

## 2.4 TRABALHOS CORRELATOS

Para a concepção deste trabalho, foram realizadas pesquisas em trabalhos acadêmicos e foram selecionados os que abordam temas sobre WebGL e computação gráfica. Dentre estes, está o trabalho sobre um motor de jogos 3D usando WebGL, de Pereira (2012), e também o trabalho sobre uma aplicação didática usando a biblioteca OpenGL de Araújo (2012).

### 2.4.1 Desenvolvimento de motor de jogos 3D utilizando WebGL

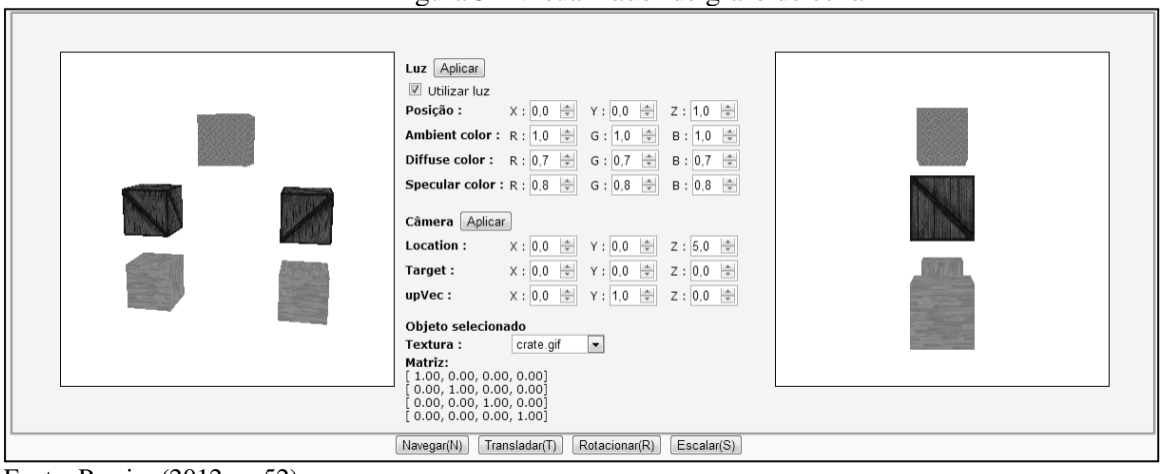
O trabalho desenvolvido por Pereira (2012) consiste na implementação de um motor de jogos 3D para web, utilizando a biblioteca gráfica WebGL e as linguagens HTML5 e JavaScript.

Em resumo, ele fornece um *framework* para desenvolvimento em WebGL, com gerenciador de grafo de cena, gerenciador de movimentos da câmera, da luz e também dos movimentos dos objetos.

O motor de jogos foi construído com o propósito de permitir que o desenvolvedor não utilize as APIs de baixo nível disponíveis na especificação do WebGL. O desenvolvedor pode desenvolver uma aplicação sem ter conhecimento de como funciona o WebGL.

Para testar todos os recursos desenvolvidos, ele criou um visualizador de grafo de cena, onde a aplicação carrega um arquivo do servidor previamente configurado e permite que o usuário edite parâmetros do objeto, da câmera e da luz ambiente. Desta forma foi possível testar e validar a funcionalidade de todos os requisitos levantados. A Figura 5 mostra o visualizador de grafo de cena, criado por Pereira (2012). A figura mostra dois elementos *canvas* exibindo os mesmos objetos, porém através de duas câmeras diferentes, sendo que só é possível movimentar a câmera da direita (PEREIRA, 2012, p. 52).

Figura 5 - Visualizador de grafo de cena




Fonte: Pereira (2012, p. 52).

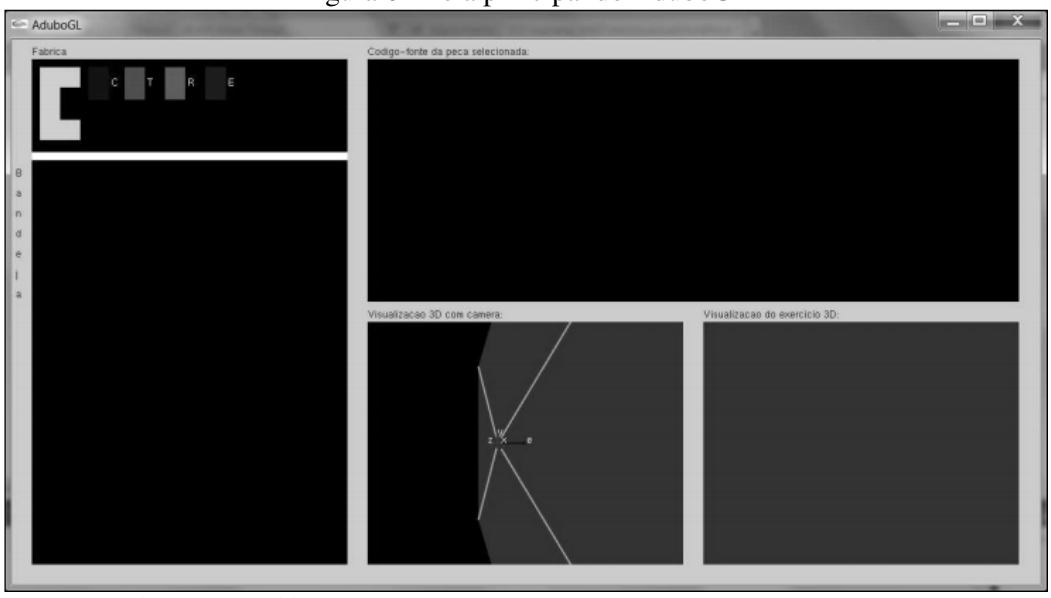
### 2.4.2 ADUBOGL

O trabalho desenvolvido por Araújo (2012) apresenta a implementação de uma aplicação voltada ao aprendizado de computação gráfica, focando mais no que diz respeito às transformações geométricas. Ela utiliza a biblioteca OpenGL como ponto fundamental para montar o cenário 2D e 3D da aplicação.

A parte fundamental deste trabalho é formada por peças que representam uma forma gráfica ou comando importante da biblioteca OpenGL para realizar uma interpretação no espaço 3D que será gerado pela junção das peças (ARAÚJO, 2012, p. 44).

Ao iniciar a aplicação, é aberta uma tela, conforme a Figura 6 que é dividida em quatro janelas. A primeira janela (esquerda), por sua vez, é dividida em duas partes, onde a parte de cima é a fábrica e a parte de baixo é a bandeja. A fábrica apresenta as possíveis peças a serem utilizadas no desenvolvimento do exercício e a bandeja é usada para a montagem do exercício. A segunda janela, na parte superior a direita, mostra o código-fonte correspondente a peça selecionada, sendo da fábrica ou da bandeja. As outras duas janelas de baixo são utilizadas para a apresentação da cena em 3D, resultante do exercício montado (ARAÚJO, 2012, p. 55). Toda a interação do usuário com a aplicação se restringe as duas primeiras janelas. As demais servem somente para exibição.

Figura 6 - Tela principal do AduboGL




Fonte: Araújo (2012, p. 55).

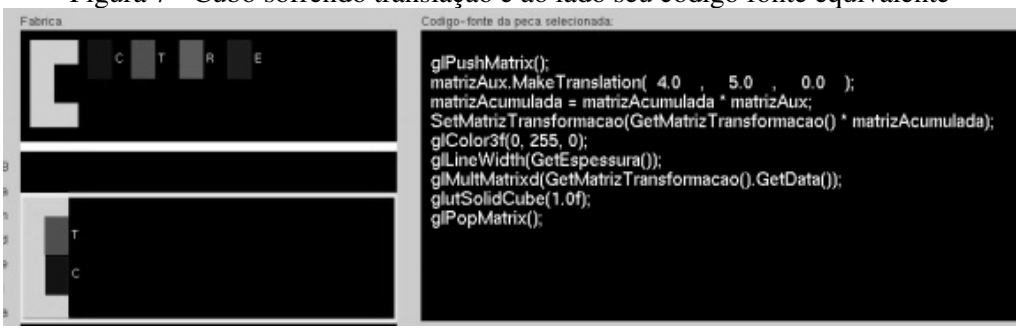
Na fábrica ficam as formas gráficas criadas para representar respectivamente os comandos `glPushMatrix()` e `glPopMatrix()` (primeira peça da esquerda para direita), um cubo (peça com a legenda C), comando de translação (peça com a legenda T), comando de rotação (peça com a legenda R) e comando de escala (peça com a legenda E).

Todas essas peças podem ser selecionadas e arrastadas para a bandeja. Uma vez que uma peça é selecionada, seu código fonte é mostrado na janela superior da direita e sua exibição é feita no espaço 3D na janela abaixo.

Para adicionar um cubo e transladá-lo por exemplo, é preciso adicionar na bandeja primeiramente a peça correspondente aos comandos `glPushMatrix()` e `glPopMatrix()`, em

seguida adicionar a peça correspondente ao movimento de translação e por último a peça equivalente ao objeto cubo, conforme a Figura 7.

Figura 7 - Cubo sofrendo translação e ao lado seu código fonte equivalente




Fonte: Araújo (2012, p. 56).

# 3 DESENVOLVIMENTO

Neste capítulo são apresentados as principais etapas para o desenvolvimento do visualizador de material educacional matemático. São apresentados os requisitos principais do trabalho, a especificação, a implementação e por último os resultados e discussões.

## 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O visualizador de material educacional matemático deverá:

a) permitir a digitação de uma função matemática (Requisito Funcional – RF);
b) validar a função digitada (RF);
c) exibir o resultado gráfico gerado a partir da função digitada (RF);
d) permitir a alteração do modo de exibição, 2D ou 3D (RF);
e) permitir interação com a câmera (RF);
f) permitir alteração de cores e texturas (RF);
g) armazenar localmente os dados gerados pela aplicação (RF);
h) utilizar JavaScript e HTML5 como linguagens para implementação (Requisito Não Funcional – RNF);
i) utilizar a biblioteca gráfica WebGL (RNF);
j) utilizar o Eclipse Juno como *Integrated Development Environment* (IDE) para desenvolvimento (RNF).

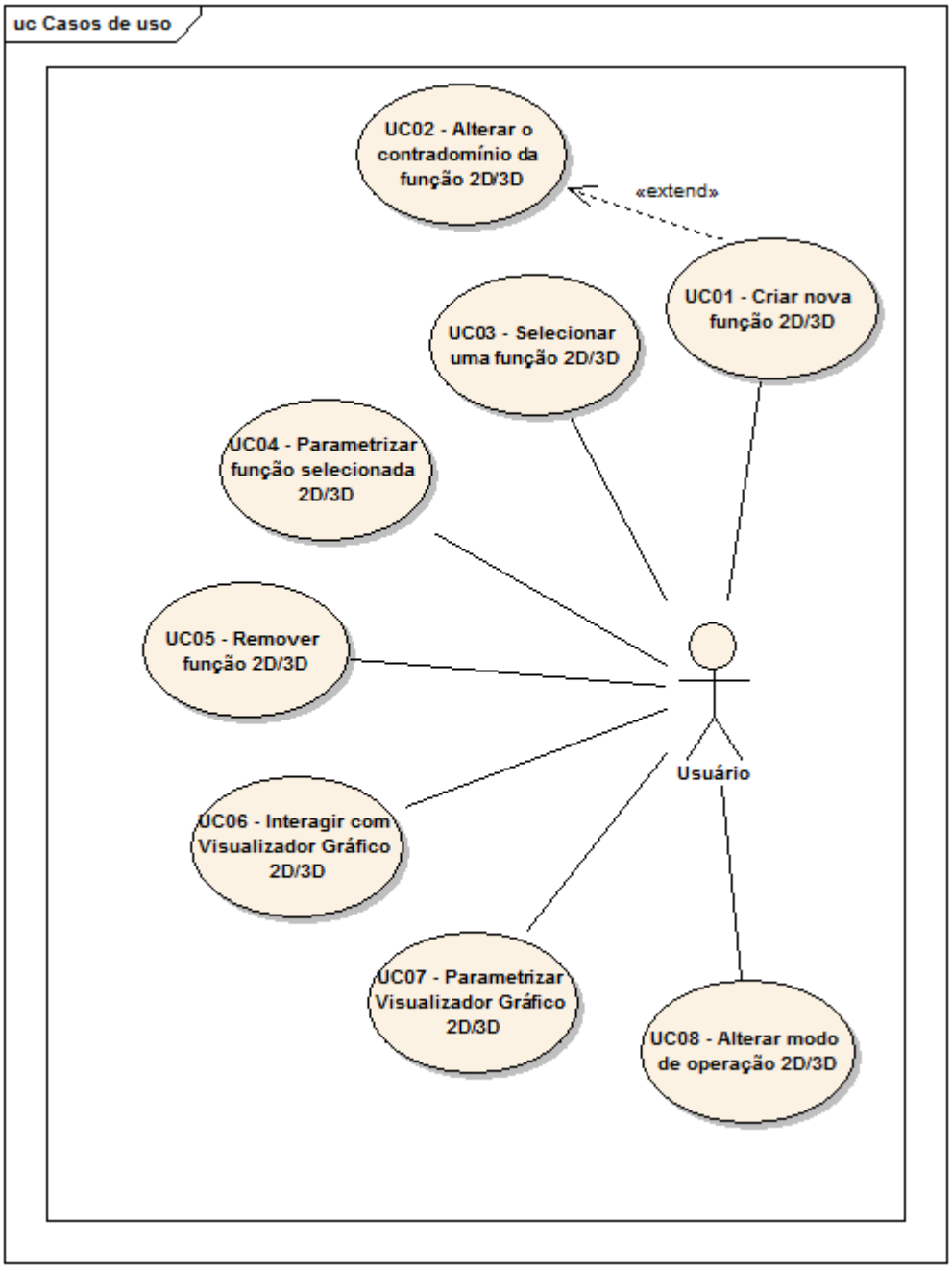
## 3.2 ESPECIFICAÇÃO

A especificação deste trabalho é baseada no paradigma de orientação a objetos usando os diagramas da *Unified Modeling Language* (UML), através da ferramenta Enterprise Architect. Na próxima seção são apresentados os diagramas de caso de uso, diagramas de classe e de sequência.

### 3.2.1 Diagrama de casos de uso

Nesta seção são descritos os casos de uso do visualizador de material educacional matemático. A partir dos requisitos, foram descritos 8 casos de uso, que tem como objetivo organizar os requisitos em funcionalidades que envolvem todas as ações que o ator pode executar na aplicação. Conforme a Figura 8, foi identificado somente um ator, descrito como `Usuário`, que utilizará todas as funcionalidade da aplicação.

Figura 8 - Diagrama de casos de uso




#### 3.2.1.1 Criar nova função 2D/3D

Este caso de uso descreve como o `Usuário` deve proceder para criar uma nova função que será exibida no visualizador gráfico. O Quadro 5 mostra os detalhes deste caso de uso.

Quadro 5 - Detalhes do caso de uso UC01

| UC01 - Criar nova função 2D/3D | |
|---|---|
| Requisitos atendidos | RF01, RF02, RF03, RF07 |
| Pré-condição | Nenhuma. |
| Cenário principal | 1. Em modo operação 2D ou 3D, o `Usuário` digita a função desejada seguindo as convenções do validador, conforme Apêndice A;<br>2. O `Usuário` pressiona o botão `OK`;<br>3. A aplicação valida a sintaxe da função;<br>4. A aplicação cria o objeto gráfico da função;<br>5. A aplicação salva a função localmente;<br>6. A aplicação adiciona a função na lista;<br>7. A aplicação exibe o objeto gráfico resultante no visualizador gráfico. |
| Exceção | Na etapa 3, caso a sintaxe da função esteja incorreta, é mostrado o erro logo abaixo da caixa de digitação. |
| Pós-condição | O `Usuário` é capaz de visualizar o resultado da função digitada no visualizador gráfico. |

### 3.2.1.2 Alterar o contradomínio da função 2D/3D

Este caso de uso mostra como o `Usuário` deve fazer para alterar o contradomínio da função que será gerada. Alterar o contradomínio implica em alterar o eixo de orientação no qual a função será gerada. O Quadro 6 apresenta o caso de uso em detalhes.

Quadro 6 - Detalhes do caso de uso UC02

| UC02 - Alterar o contradomínio da função 2D/3D | |
|---|---|
| Requisitos atendidos | RF01 |
| Pré-condição | Nenhuma. |
| Cenário principal | 1. Em modo de operação 3D, o `Usuário` clica no botão do contradomínio;<br>2. A aplicação altera o eixo no qual a função será exibida, de acordo com o contradomínio escolhido, que pode ser `x, y` ou `z`;<br>3. A aplicação altera os parâmetros de domínio de acordo com o contradomínio escolhido. |
| Cenário alternativo 01 | 1. Em modo de operação 2D, o `Usuário` clica no contradomínio;<br>2. A aplicação altera o eixo no qual a função será exibida, de acordo com o contradomínio escolhido, que pode ser `x` ou `y;`<br>3. A aplicação altera os parâmetros de domínio de acordo com o contradomínio escolhido. |
| Pós-condição | A função resultante será exibida no eixo correspondente ao contradomínio escolhido, que poder ser `x, y` ou `z` para modo de operação 3D, ou `x e y` para o modo de operação 2D. |

### 3.2.1.3 Selecionar uma função 2D/3D

Este é o caso de uso que descreve como o `Usuário` deve selecionar uma função gerada. O Quadro 7 apresenta detalhes do caso de uso.

Quadro 7 - Detalhes do caso de uso UC03

| UC03 - Selecionar uma função 2D/3D | |
|---|---|
| Requisitos atendidos | Nenhum. |
| Pré-condição | UC01 |
| Cenário principal | 1. Em modo de operação 2D ou 3D, o `Usuário` clica uma vez sobre a função na lista de funções;<br>2. A aplicação altera a cor da função selecionada na lista;<br>3. A aplicação exibe a `BoundingBox` do objeto no visualizador gráfico;<br>4. A aplicação habilita os componentes parametrizáveis da função no menu principal. |
| Cenário alternativo 01 | 1. Em modo de operação 3D, o `Usuário` clica duas vezes sobre o objeto gráfico da função no visualizador gráfico;<br>2. A aplicação altera a cor da função selecionada na lista;<br>3. A aplicação exibe a `BoundingBox` do objeto no visualizador gráfico;<br>4. A aplicação habilita os componentes parametrizáveis da função no menu principal. |
| Pós-condição | O `Usuário` estará apto a alterar o parâmetros da função. |

#### 3.2.1.4 Parametrizar função selecionada 2D/3D

Este caso de uso mostra como o `Usuário` deve proceder para alterar os parâmetros de uma função gerada. Detalhes do caso de uso são mostrados no Quadro 8.

Quadro 8 - Detalhes do caso de uso UC04

| UC04 - Parametrizar função selecionada 2D/3D | |
|---|---|
| Requisitos atendidos | RF06 |
| Pré-condição | UC03 |
| Cenário principal | 1. Em modo de operação 3D, o `Usuário` clica sobre o `checkBox Wireframe`;<br>2. A aplicação exibe somente os vértices que compõem a malha de pontos do objeto;<br>3. O `Usuário` seleciona uma cor no `comboBox Cor`;<br>4. A aplicação altera a cor do objeto selecionado de acordo com a escolha do usuário;<br>5. O `Usuário` seleciona um material no `comboBox Material`;<br>6. A aplicação altera o material do objeto selecionado de acordo com a escolha do usuário;<br>7. O `Usuário` aumenta ou diminui os `sliders` de domínios da função;<br>8. A aplicação aumenta ou diminui o intervalo de valores no qual a função é gerada;<br>9. A aplicação recria a função com os novos intervalos de valores de domínio. |
| Cenário alternativo 01 | 1. Em modo de operação 2D, o `Usuário` seleciona uma cor no `comboBox Cor`;<br>2. A aplicação altera a cor do objeto selecionado de acordo com a escolha do usuário;<br>3. O `Usuário` aumenta ou diminui o `slider` de domínio da função;<br>4. A aplicação aumenta ou diminui o intervalo de valores no qual a função é gerada;<br>5. A aplicação recria a função com os novos intervalos de valores de domínio;<br>6. O `Usuário` clica no `checkBox Ponteiro`;<br>7. A aplicação passa a mostrar o resultado a função, conforme o mouse é movimentado no visualizador gráfico. |
| Exceção 01 | Na etapa 6 do cenário alternativo 01, caso o `Usuário` não tenha habilitado o cursor, o `checkBox Ponteiro` ficará desabilitado até que o cursor tenha sido habilitado. |
| Pós-condição | O `Usuário` passa a visualizar a função com as alteração feitas |

#### 3.2.1.5 Remover função 2D/3D

Este caso de uso descreve como o `Usuário` deve proceder para remover uma função gerada. Detalhes do caso de uso são apresentados no Quadro 9.

Quadro 9 - Detalhes do caso de uso UC05

| UC05 - Remover função 2D/3D | |
|---|---|
| Requisitos atendidos | Nenhum. |
| Pré-condição | UC01 |
| Cenário principal | 1. O `Usuário` clica no ícone lixeira da função na lista que deseja remover;<br>2. A aplicação remove a função da lista;<br>3. A aplicação remove do visualizador gráfico, o objeto gráfico da função. |
| Pós-condição | O visualizador gráfico e a lista de funções passam a ter uma função a menos sendo exibida. |

#### 3.2.1.6 Interagir com visualizador gráfico 2D/3D

Este caso de uso mostra como o `Usuário` deve fazer para interagir com o visualizador gráfico. O Quadro 10 apresenta o caso de uso em detalhes.

Quadro 10 - Detalhes do caso de uso UC06

| UC06 - Interagir com visualizador gráfico 2D/3D | |
|---|---|
| Requisitos atendidos | RF05 |
| Pré-condição | Nenhuma. |
| Cenário principal | 1. Em modo de operação de 3D, o `Usuário` clica com o botão esquerdo do mouse e arrasta para cima, para baixo, para esquerda ou para direita;<br>2. A aplicação rotaciona a câmera para cima, para baixo, para esquerda ou para direita respectivamente;<br>3. O `Usuário` rola o `scroll` do mouse para frente ou para trás;<br>4. A aplicação adiciona ou remove zoom respectivamente;<br>5. O `Usuário` clica com o botão direito do mouse e arrasta para cima, para baixo, para esquerda ou para direita;<br>6. A aplicação translada a câmera para cima, para baixo, para esquerda ou para direita respectivamente. |
| Cenário alternativo 01 | 1. Em modo de operação 2D, o `Usuário` rola o `scroll` do mouse para frente ou para trás;<br>2. A aplicação adiciona ou remove zoom respectivamente. |
| Pós-condição | O `Usuário` pode visualizar a função gerada de vários ângulos diferentes, sendo mais próximo ou mais distante. |

#### 3.2.1.7 Parametrizar visualizador gráfico 2D/3D

Este caso de uso descreve como o `Usuário` pode alterar parâmetros de configurações do visualizado gráfico. Detalhes do caso de uso são apresentados no Quadro 11.

Quadro 11 - Detalhes do caso de uso UC07

| UC07 - Parametrizar visualizador gráfico 2D/3D | |
|---|---|
| Requisitos atendidos | Nenhum. |
| Pré-condição | Nenhuma. |
| Cenário principal | 1. Em modo de operação 2D ou 3D, o `Usuário` desmarca o `checkBox Grade` no menu principal;<br>2. A aplicação oculta a grade auxiliar;<br>3. O `Usuário` desmarca o `checkBox Eixos` no menu principal;<br>4. A aplicação oculta os eixos de orientação auxiliar. |
| Cenário alternativo 01 | 1. Em modo de operação 3D, o `Usuário` desmarca os `checkBoxs Luzes`;<br>2. A aplicação remove as luzes do visualizador gráfico;<br>3. A aplicação altera a maneira como as luzes afetam a exibição dos materiais do tipo `Sólido` e `Brilhante`;<br>4. O `Usuário` seleciona uma posição pré-definida do `comboBox Camera` que pode ser acima, abaixo, direita, esquerda, frente e atrás;<br>5. A aplicação altera a visualização da câmera conforme selecionado na etapa 4. |
| Cenário alternativo 02 | 1. Em modo de operação 2D, o `Usuário` marca o `checkBox Cursor`;<br>2. A aplicação exibe o cursor do visualizador gráfico;<br>3. A aplicação passa a exibir a posição `x` e `y` do mouse. |
| Pós-condição | O `Usuário` pode exibir e ocultar elementos auxiliares no visualizador gráficos. |

#### 3.2.1.8 Alterar o modo de operação 2D/3D

Este caso de uso descreve como o `Usuário` deve fazer para alterar o modo de operação da aplicação, alternando entre uma visão 2D ou 3D. O Quadro 12 apresenta mais detalhes do caso de uso.

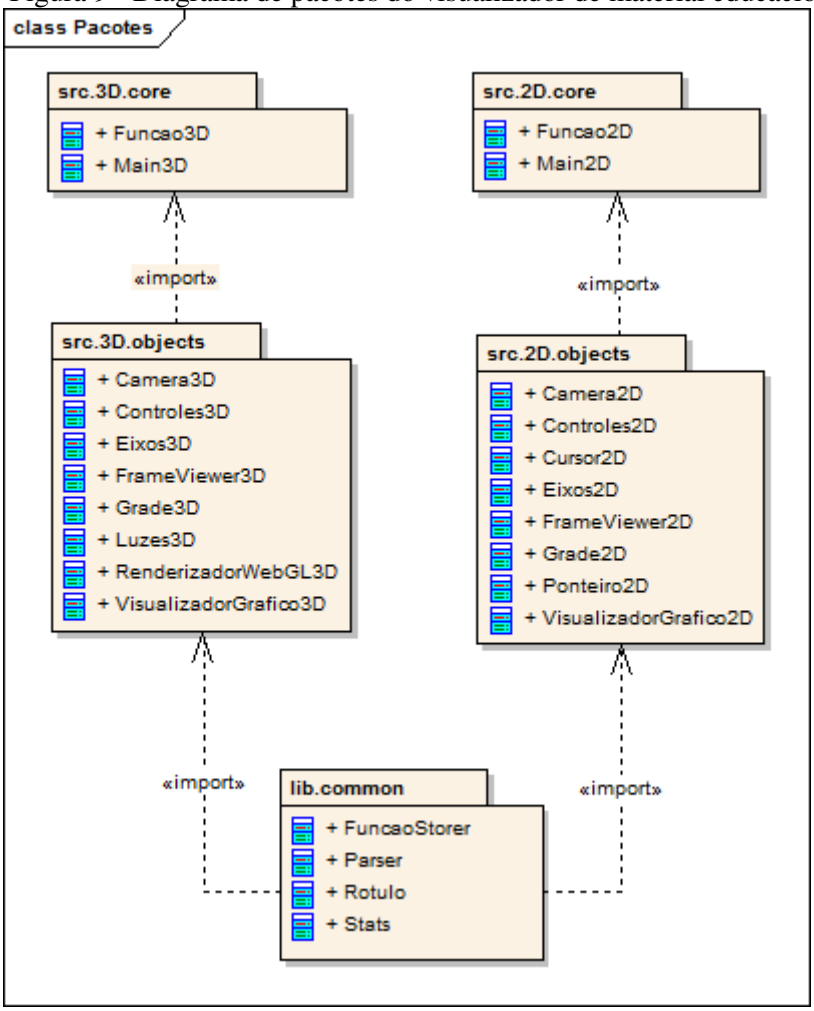
Quadro 12 - Detalhes do caso de uso UC08

| UC08 - Alterar o modo de operação 2D/3D | |
|---|---|
| Requisitos atendidos | RF04. |
| Pré-condição | Nenhuma. |
| Cenário principal | 1. Em modo de operação 3D, o `Usuário` clica no botão `2D`;<br>2. A aplicação altera o modo de operação para 2D. |
| Cenário alternativo 01 | 1. Em modo de operação 2D, o `Usuário` clica no botão `3D`;<br>2. A aplicação altera o modo de operação para 3D. |
| Pós-condição | O `Usuário` pode criar, selecionar e alterar funções de acordo com o modo de operação escolhido. |

### 3.2.2 Diagrama de classes

Nesta seção são descritas as classes que compõem a aplicação visualizador de material educacional matemático. Na Figura 9 são exibidos os pacotes principais da aplicação, como estão relacionados e quais as classes que os compõem.

Figura 9 - Diagrama de pacotes do visualizador de material educacional



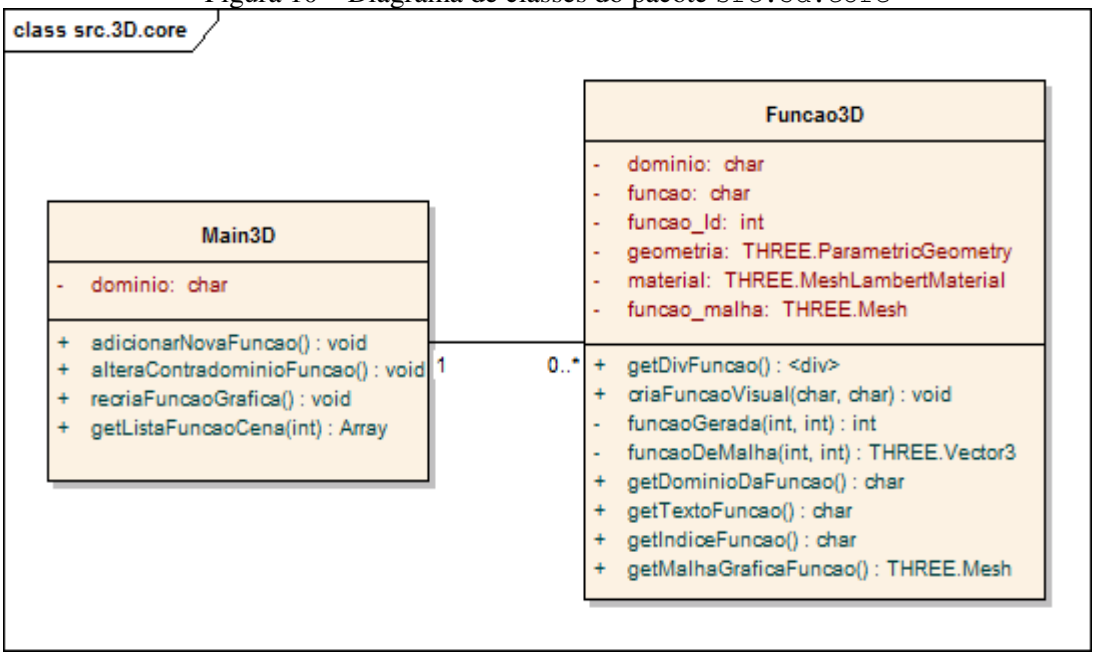

#### 3.2.2.1 Pacote `src.3D.core`

O pacote `src.3D.core` constitui o núcleo do visualizador de material educacional matemático 3D. É neste pacote que estão as classes reponsáveis por criar e gerenciar uma nova função 3D. A Figura 10 apresenta o diagrama de classes deste pacote, assim como seus principais métodos e atributos.

A classe `Main3D` é a entidade responsável por criar explicitamente uma nova função e gerenciá-la. Ela possui um método chamado `adicionaNovaFuncao` executado quando o usuário clica no botão `OK` localizado ao lado da caixa de digitação da função. Esse método instancia um novo objeto `Funcao3D` passando como parâmetros a função digitada na caixa de

texto, o índice da função que também será seu identificador único e o domínio da função. Essa classe possui apenas um atributo chamado `dominio`, usado para armazenar o domínio atual da função, que será passado como parâmetro para a criação do objeto gráfico da mesma.

Figura 10 – Diagrama de classes do pacote `src.3d.core`



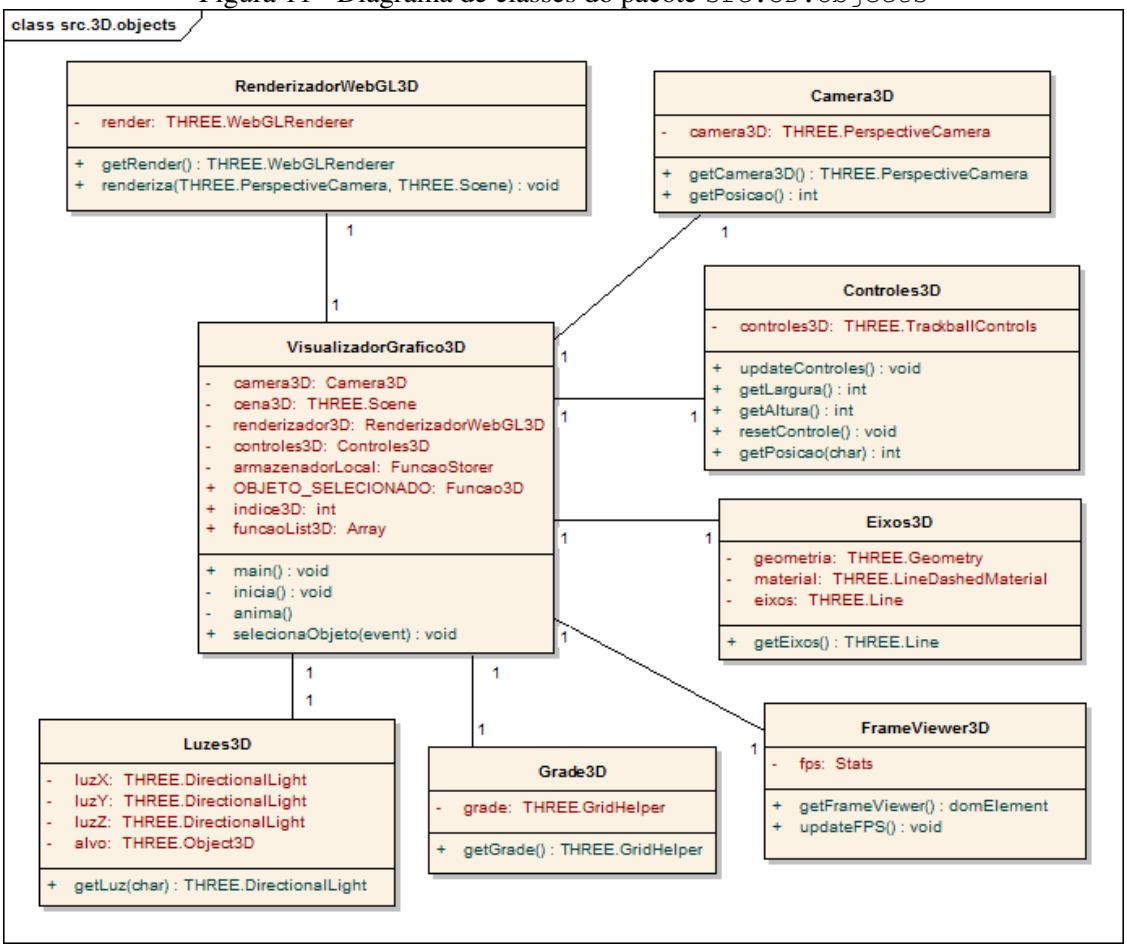

A classe `Funcao3D` é a entidade da aplicação que representa um objeto gráfico da cena do visualizador gráfico. Ao ser instanciada, a classe `Funcao3D` cria um elemento `<div>` que será inserido na lista de funções, abaixo do campo de digitação da função, e que pode ser obtido através da chamada do método `getDivFuncao`, e cria também o objeto gráfico `função_malha` resultante da função que pode ser obtido através do método `getMalhaGraficaFuncao`.

O método `criaFuncaoVisual` é quem atribui conteúdo para os atributos `geometria`, `material` e `funcao_malha`. O atributo `geometria` recebe um novo objeto do tipo `THREE.ParametricGeometry` criado a partir de dois parâmetros, sendo um dos parâmetros um método JavaScript gerado após a validação da função, e outro parâmetro é a quantidade de segmentos no qual será dividida a malha de pontos, considerado como nível de detalhe do objeto. O atributo `material` recebe somente uma cor como parâmetro e é quem possui um objeto do tipo `THREE.MeshLambertMaterial` que define o tipo da superficie do objeto e a sua cor. Já o atributo `funcao_malha` possui um objeto do tipo `THREE.Mesh` que recebe como parâmetros uma geometria e um material, ambos previamente criados. Esse objeto desenha a geometria dos vértices gerados com o material colorido formando a malha gráfica que será exibida na cena do visualizador gráfico.

### 3.2.2.2 Pacote `src.3D.objects`

O pacote `src.3D.objects` contém as classes da aplicação que compõem o visualizador gráfico. As classes deste pacote são responsáveis pela exibição dos objetos gráficos das funções. Todas as classes deste pacote fazem uso de algum objeto da biblioteca Three.js do pacote `lib.common`, conforme a Figura 11.

Figura 11 - Diagrama de classes do pacote `src.3D.objects`




A classe `VisualizadorGrafico3D` é a entidade principal deste pacote e é iniciada assim que a página é carregada. É nesta classe que são instanciados todos os objetos, derivados da biblioteca Three.js, que serão inseridos na cena. Para adicionar um novo objeto gráfico na cena, é usado o método `add()` da classe `THREE.Scene` da biblioteca Three.js, no qual é referenciada pela variável `cena3D`.

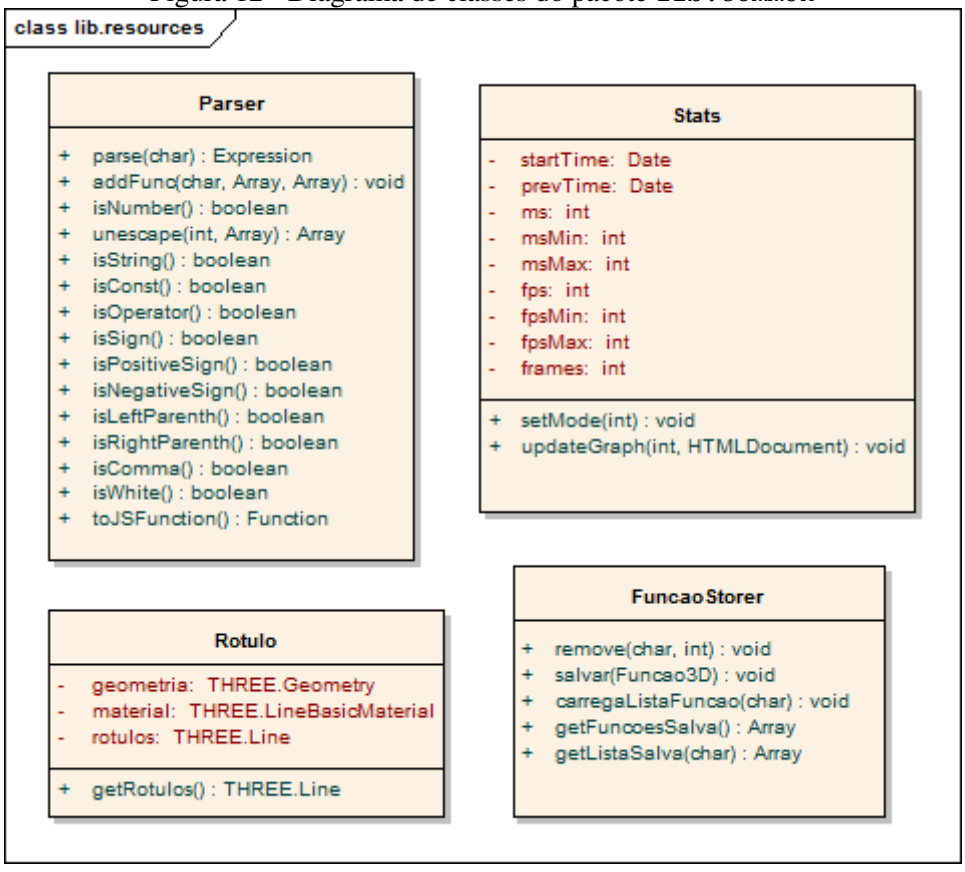
Ao ser iniciada, a classe `VisualizadorGrafico3D` executa o seu método principal chamado `main()` que possui dois outros métodos, o `inicia()` e o `anima()`. O método `inicia()` apenas cria os objetos, inclusive a cena, e adiciona-os na cena. O método `anima()` executa a chamada para a função `renderiza()` da classe `RenderizadorWebGL3D` que é o responsável por renderizar os frames da cena em intervalos de tempos. Esse método possui

uma chamada recursiva para ele mesmo. Essa classe possui três variaveis globais muito importantes que são `OBJETO_SELECIONADO`, `indice3D` e `funcaoList3D`. O atributo `OBJETO_SELECIONADO` mantém a referência para o objeto gráfico selecionado no visualizador gráfico. A variável `indice3D` funciona como contador global de funções criadas. A cada nova função criada, essa variável é incrementada com o objetivo de manter o controle das mesmas. E por último, a variável `funcaoList3D` mantém a lista de funções adicionadas no visualizador gráfico.

#### 3.2.2.3 O pacote `lib.common`

O pacote `lib.common` é formado pelas classes auxiliares que não fazem parte do núcleo principal do visualizador de material educacional matemático mas que são frequentemente utilizadas. Neste pacote estão as classes responsáveis pelo armazenamento local das funções adicionadas na lista, pela análise sintática da função digitada e pela exibição do visualizador de `frames` por segundo. A Figura 12 mostra os principais métodos e atributos destas classes.

Figura 12 - Diagrama de classes do pacote `lib.common`



A classe `Parser` é a entidade da aplicação responsável pela análise sintática da função digitada. Através do seu método `parse` é possível validar um texto passado como parâmetro em seguida transformá-lo em função JavaScript executando o método `toJSFunction`.

A classe `FuncaoStorer` é quem implementa o recurso de armazenamento local, ou `Local Storage`, do HTML5. Através dos seus métodos é possível salvar uma nova função localmente, remover uma função e carregar uma lista de funções usando os metódos `salvar`, `remove` e `carregaListaFuncao` respectivamente. Como esta classe é a mesma para a versão 3D e para a versão 2D do visualizador de material educacional matemático, os métodos implementados possuem uma lógica diferenciada para cada versão, uma vez que o recurso de armazenamento local mantém salvo o seu conteúdo mesmo depois que a sessão é encerrada e o navegador é fechado. Isso se mostrou importante pois evitou-se que funções salvas na versão 3D fossem carregadas na versão 2D.

#### 3.2.2.4 Pacotes da versão 2D

Os pacotes `src.2D.core` e `src.2D.objects` são equivalentes aos pacotes `src.3D.core` e `src.3D.objects` respectivamente, mas para o visualizador de material educacional matemático 2D. Eles são responsáveis pelas mesmas ações executadas na versão 3D com algumas diferenças no que diz respeito a geometria do objeto, que na versão 3D usa espaço gráfico tridimensional e na versão 2D, bidimensional. Algumas das diferenças entre as versões 3D e 2D incluem também a câmera usada na cena principal. Na versão 3D a câmera utilizada foi do tipo perspectiva, já na versão 2D foi utilizado uma câmera do tipo ortográfica com visualização somente para os eixos `x` e `y`. Outra diferença é que na classe `Funcao2D` os valores permitidos para domínio e contradomínio variam apenas entre `x` e `y` diferente da versão 3D que variam entre `x`, `y` e `z`.

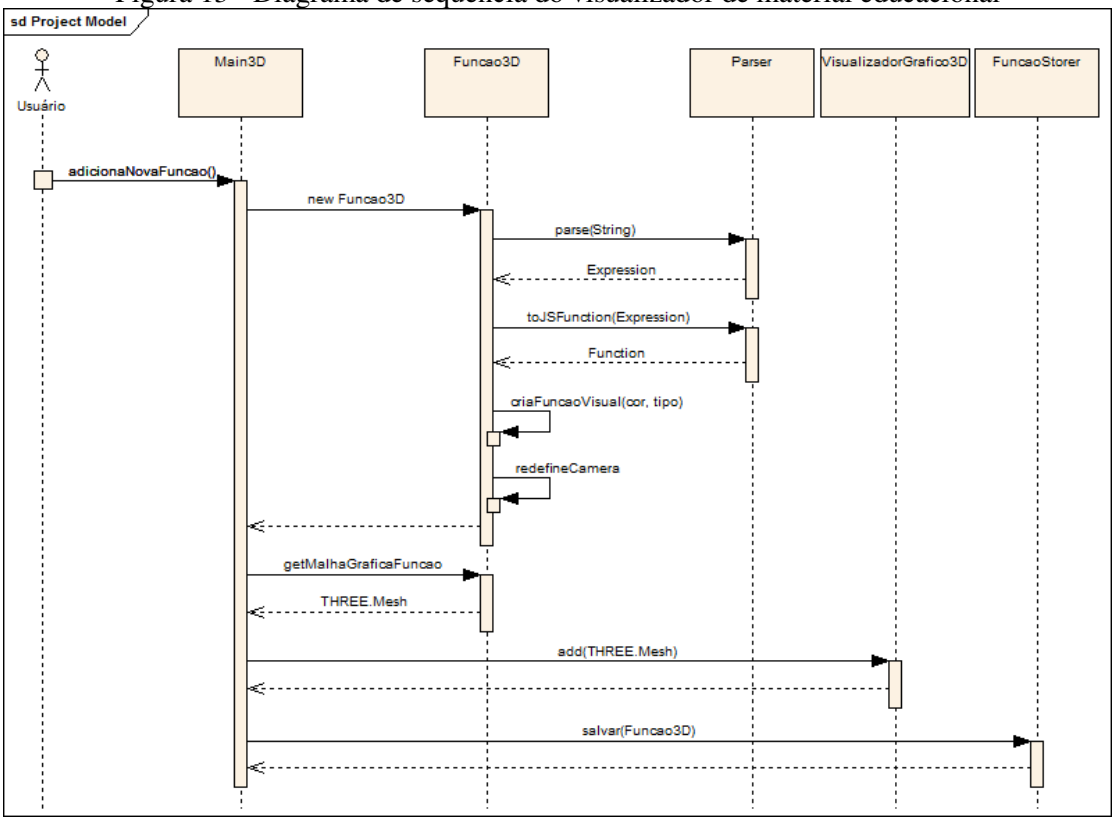
### 3.2.3 Diagrama de sequência

Esta seção apresenta um diagrama de sequência do visualizador de material educacional matemático que ilustra o processo de criação e adição de uma nova função 3D na cena principal do visualizador gráfico. A Figura 13 exibe o diagrama em detalhes.

O fluxo inicia após o `Usuário` digitar a função no campo de texto, que é executado o método `adicionaNovaFuncao`, este método cria uma nova instância de `Funcao3D` passando como parâmetros o texto digitado, o índice da função e o contradomínio. Na sequência, ocorre a chamada para o método `parse` que valida a sintaxe da função digitada e retorna uma nova expressão, que em seguida é transformada em um função JavaScript através do método

`toJSFunction`. Feito isso, a classe `Funcao3D` cria o objeto gráfico da função usando o método `criaFuncaoVisual` e redefine a posição da câmera através do método `redefineCamera`. Após a etapa de criação do objeto gráfico da função, a classe `Main3D` faz uma chamada do método `getMalhaGraficaFuncao` que retorna o objeto do tipo `THREE.Mesh`. Esse objeto representa a malha gráfica de pontos renderizada na etapa de criação que é adicionada na cena do visualizador gráfico e também salva na lista de funções mantidas pelo recurso de `Local Storage` implementado na classe `FuncaoStorer`.

Figura 13 - Diagrama de sequência do visualizador de material educacional




## 3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas para a implementação da aplicação e a operacionalidade desta implementação.

### 3.3.1 Técnicas e ferramentas utilizadas

Para a concepção desta aplicação foram utilizados a linguagem JavaScript para o desenvolvimento das classes e HTML5 para implementação da interface web. Também foram utilizadas a biblioteca gráfica `Three.js` release 60 e a biblioteca `ASCIIMathML` versão 2.0 para transformação da notação ASCII em MathML (atualmente só funciona no Firefox). Como ambiente de desenvolvimento foi o usado o Notepad++ versão 6.1.1.

Para controle de versões e armazenamento seguro dos fontes foi utilizada a ferramenta SourceTree versão 1.3.1.0 e criado um repositório on-line (KRAUSS, 2013) hospedado pelo serviço BitButcket.

Os navegadores utilizados nos testes foram Google Chrome versão 30.0.1599.101, Mozilla Firefox versão 25.0 e Opera versão 17.0.1241.53.

O computador usado em todo o desenvolvimento da aplicação foi um desktop Dell modelo Optiplex 980, com sistema operacional Windows 7 Professional Service Pack 2, com processador Intel Core i5, memória RAM de 8 GB e placa de vídeo ATI Radeon HD3450.

### 3.3.2 O visualizador de material educacional (VisEdu-MAT)

Esta seção descreve a implementação do Visualizador de material Educacional Matemático. As principais etapas que descritas ocorrem tanto na versão 2D como na versão 3D, portanto os conceitos utilizados e as estruturas que formam as duas interfaces web principais são as mesmas para ambas as versões.

#### 3.3.2.1 O visualizador gráfico

A exibição de um objeto gráfico resultante de uma função no visualizador gráfico é uma das partes principais do VisEdu-MAT, tanto na versão 3D quanto na versão 2D.

Para a criação de uma nova função é preciso que o elemento `<canvas>`, que representa o visualizador gráfico, já esteja carregado e a cena principal da aplicação precisa estar instanciada, ou seja, pronta para adicionar novos objetos gráficos.

Para que isso seja possível, as classes `VisualizadorGrafico3D` e `VisualizadorGrafico2D` possuem um método principal chamado `main` que é executado quando a página é carregada, através da chamada `window.onload`. Essa chamada para o método `main` executa a criação da cena, da câmera, das luzes, da grade, dos eixos de orientação e do renderizador WebGL. Como a câmera, as luzes e os outros objetos fazem parte da cena, estes são criados posteriormente para que possam ser adicionados na cena do visualizador gráfico, através do método `add`. A Figura 14 apresenta o método `main` e a criação dos objetos e adição na cena.

Durante a inicialização do visualizador gráfico no método `main`, ocorre também a chamada para o método `carregaListaFuncao` da classe `FuncaoStorer` responsável pelo gerenciamento do armazenamento local de funções previamente salvas. A chamada desse método recebe como parâmetro uma `String` com o tipo de função que se deseja carregar, que pode ser 3D ou 2D, visto que o recurso é usado para ambas as versões.

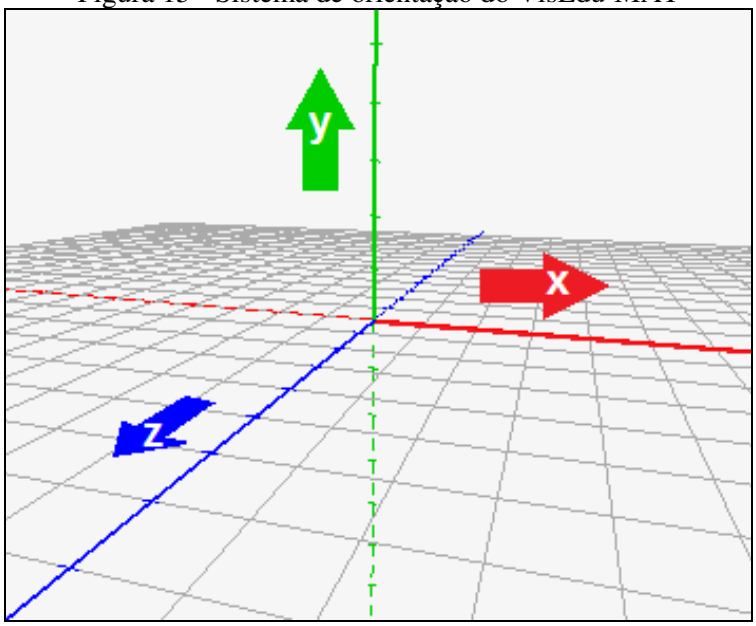
Após a chamada do método `main`, é executado o método `anima` que inicia uma chamada recursiva para a função `renderiza` da classe `RenderizadorWebGL3D`. Essa função é executada enquanto a aplicação estiver em execução para que os `frames` sejam renderizados várias vezes por segundo.

Figura 14 - Detalhes do método `main`

```
24
25  function main(){
26      inicia();
27      anima();
28  };
29
30  function inicia(){
31      //Detecta qual e o browser usado
32      BrowserDetect.init();
33
34      //Cria a cena3D
35      cena3D = new THREE.Scene();
36
37      //Adiciona a camera3D
38      camera3D = new Camera3D();
39      cena3D.add(camera3D.getCamera3D());
40
41      //Cria o renderizador3D "WebGLRenderer"
42      renderizador3D = new RenderizadorWebGL3D();
43      container = document.getElementById("div_drawArea");
44      container.appendChild(renderizador3D.getRender());
45
46      //Adiciona a grade auxiliar
47      var grade = new Grade3D();
48      cena3D.add(grade.getGrade());
49
50      //Adiciona os controle do mouse3D na camera3D
51      controles3D = new Controles3D( camera3D.getCamera3D(), renderizador3D.getRender());
```

O visualizador gráfico usa o mesmo sistema de orientação padrão do WebGL chamado de sistema de coordenadas tridimensional ortonormal mão direita. No entanto por motivos didáticos, a câmera foi posicionada de frente para o eixo z de modo que o eixo y continue apontando para cima e o eixo x para a direita da mesma forma que foi usado na versão 2D. A Figura 15 demonstra o sistema de orientação utilizado pelo VisEdu-MAT.

Figura 15 - Sistema de orientação do VisEdu-MAT




### 3.3.2.2 Criação de uma nova função

A etapa de criação de uma nova função inicia após o usuário pressionar o botão `OK`, disparando o método `adicionarNovaFuncao` associado ao evento `onclick`, que instancia um novo objeto `Funcao3D` ou `Funcao2D`, passando como parâmetros, a função digitada, o identificador da função e o contradomínio escolhido.

Essa etapa é dividida em três partes principais. A primeira parte é a validação da sintaxe da função digitada, a segunda parte é a transformação do texto da função digitada em uma função JavaScript válida e a terceira parte é a geração da malha de pontos a partir da função JavaScript previamente criada.

Para fazer a validação da sintaxe da função foi usado o método `parse` da classe `Parser`. Esse método recebe como parâmetro uma `String` no qual será feita a análise sintática, e retorna um objeto do tipo `Expression` que contem uma lista de `tokens` gerados pelo `parse`. Se não ocorrer nenhum erro de sintaxe na função digitada, é iniciada a segunda parte da criação, que é a etapa de transformação do texto em função JavaScript executável.

Para transformar o texto em função JavaScript, o objeto da classe `Expression` executa o método `toJSFunction` usando como parâmetros dois valores que são as variáveis de domínio escolhida pelo usuário. Portanto, se for escolhido o contradomínio em `z` na criação da função, as variáveis de domínio usadas implicitamente como parâmetro por esse método serão `x` e `y`. Se for escolhido o contradomínio em `y`, as variáveis usadas serão `x` e `z` e se caso for escolhido o contradomínio em `x`, as variáveis de domínio usadas serão `z` e `y`. A Figura 16

demonstra o trecho de código que gera a função Javascript baseado na escolha do contradomínio.

A ideia de fornecer ao usuário um meio de poder alterar o contradomínio da função, surgiu após uma entrevista feita com o professor de matemática Londero (2013)[2] onde o mesmo explica que isso facilitaria o entendimento dos usuários quanto aos eixos de orientação do objeto gráfico resultante da função.

Figura 16 - Execução dos métodos `parse` e `toJSFunction` baseado no contradomínio

```
232
233     if(contradominioDaFuncao == 'z'){
234         funcaoGerada = Parser.parse(funcao).toJSFunction(['x', 'y']);
235     } else if (contradominioDaFuncao == 'y'){
236         funcaoGerada = Parser.parse(funcao).toJSFunction(['x', 'z']);
237     } else {
238         funcaoGerada = Parser.parse(funcao).toJSFunction(['z', 'y']);
239     }
240
```

A função gerada a partir do método `toJSFunction` é salva na variável `funcaoGerada`.

Após criada a função Javascript a partir da função digitada, inicia-se a parte de geração da malha de pontos que serão renderizados e exibidos no visualizador gráfico.

Para criar a malha de pontos, é executado o método `criaFuncaoVisual` que recebe como parâmetros uma cor e um tipo de material. A Figura 17 apresenta a chamada para o método que é feita no corpo da classe `Funcao3D`.

Figura 17 - Detalhes da chamada para o método `criaFuncaoVisual`

```
241
242   if (BrowserDetect.browser != 'Mozilla'){
243       this.criaFuncaoVisual(cores[(funcao_Id+1)%cores.length], "Solido");
244   } else {
245       this.criaFuncaoVisual(cores[(funcao_Id+1)%cores.length], "Basico");
246   }
```

Esse método cria uma instância do objeto `THREE.ParamectricGeometry` da biblioteca `Three.js` no qual executa o algorítmo de criação de superfícies de malhas paramétricas. Esse objeto recebe como parâmetros uma função que será utilizada para geração dos pontos da malha paramétrica, o nível de detalhe da coordenada `U` e o nível de detalhe da coordenada `V`. O nível de detalhe é um parâmetro de valor constante, definido após vários testes de geração de malhas para que não ficassem muito quadradas. O valor definido para o nível de detalhe sempre será 40 para ambas as coordenadas, ou seja, a quantidade total de vértices gerada para qualquer função sempre será 1.600 vértices. A Figura 18 apresenta os detalhes do método que cria a função dentro da classe `Funcao3D`.

[2] O professor tem aproximadamente 30 anos de experiência em ensinar disciplinas relacionadas à matemática.

Figura 18 - Detalhes do método `criaFuncaoVisual`

```
142
143 //Metodo privado que cria a equacao grafica a partir da funcao passado como parametro
144 this.criaFuncaoVisual = function(cor, tipo mat){
145     nivelDetalhe = 40;
146
147     //Cria a parte geometrica do objeto grafico baseado no parametro 'funcaoDeMalha'
148     geometria = new THREE.ParametricGeometry(this.funcaoDeMalha, nivelDetalhe, nivelDetalhe);
149     //Cria o material e a cor baseado nos parametros 'cor' e 'tipo_mat'
150     if(tipo_mat == "Solido"){
151         material = new THREE.MeshLambertMaterial( { color: cor, side: THREE.DoubleSide } );
152     } else if(tipo_mat == "Basico"){
153         material = new THREE.MeshBasicMaterial( { color: cor, side: THREE.DoubleSide } );
154     } else if(tipo_mat == "Brilhante"){
155         material = new THREE.MeshPhongMaterial( { color: cor, specular: '#ffffff', shininess: 100, side: THREE.DoubleSide } );
156     }
157     funcao_malha = null;
158     //Une a geometria e o material em uma malha grafica
159     funcao_malha = new THREE.Mesh(geometria, material);
160     //O nome da equacao recebe o mesmo nome do seu respectivo checkbox na lista de equacao.
161     funcao_malha.name = funcao_Id;
162 }
163
```

Ao final da geração da malha de pontos, a variável `funcao_malha` recebe um objeto do tipo `THREE.Mesh` que é o objeto final que une a geometria dos vértices com o material e que será adicionado na cena do visualizador gráfico.

#### 3.3.2.3 Interagindo com os parâmetros

Para que fosse possível a exibição de um objeto gráfico a partir de uma função, foi necessário o uso de intervalos de valores para as variáveis de domínio presentes no texto da função. O intervalo padrão escolhido para a criação de uma nova função para a versão 3D foi de -2 a 2 para cada variável de domínio, já para a versão 2D foi definido um intervalo de -4 a 4. A razão dessa escolha foi pelo motivo de que certas operações matemáticas como a pontenciação, tornavam o objeto gráfico grande demais para exibir na cena do visualizador gráfico.

A ideia de interagir com os parâmetros de intervalos de valores de uma função foi para criar uma didática maior em cima do funcionamento da mesma. Ao alterar os valores de um desses parâmetros através do componente `slider`[3] no menu principal, o objeto gráfico da função é totalmente recriado usando os novos valores que podem ser menores ou maiores, dando a impressão de que ao aumentar o intervalo, o objeto gráfico estaria aumentando e ao diminuir o intervalo, o objeto gráfico estaria diminuindo. Para recriar o objeto gráfico, foi adicionado o método `recriaFuncaoGrafica` na propriedade `slide` do componente `slider`. A Figura 19 apresenta a chamada para a função nas propriedades do componente `slider`.

[3] Componente gráfico da biblioteca jQuery UI que possui uma barra e ponteiros deslizáveis.

Figura 19 - Chamada para o método `recriaFuncaoGrafica`

```
389 $(function() {
390     $( "#slider-range_1" ).slider({
391         range: true,
392         min: -10,
393         max: 10,
394         step: 0.2,
395         values: [ -2, 2 ],
396         slide: function( event, ui ) {
397             $( "#in_range_1" ).val(ui.values[ 0 ] + "    |    " + ui.values[ 1 ]);
398             if(OBJETO_SELECIONADO != null){
399                 OBJETO_SELECIONADO.setRange1(ui.values[ 0 ], ui.values[ 1 ]);
400                 recriaFuncaoGrafica();
401             }
402         }
403         });
```

Antes da chamada de recriação da função, ocorre a alteração dos intervalos de valores através do método `setRange`, o qual recebe como parâmetros os valores atuais dos ponteiros do `slider`. Os valores mínimos e máximos definidos para os intervalos de valores da versão 3D foi de -10 e 10 e para a versão 2D foi -15 e 15, respectivamente.

### 3.3.3 Operacionalidade da implementação

As seções a seguir descrevem os detalhes e funcionalidades da operacionalidade VisEdu-MAT. Inicialmente é apresentada a versão 3D do visualizador gráfico e na sequência é apresentada a versão 2D.

#### 3.3.3.1 Visualizador de material educacional matemático 3D

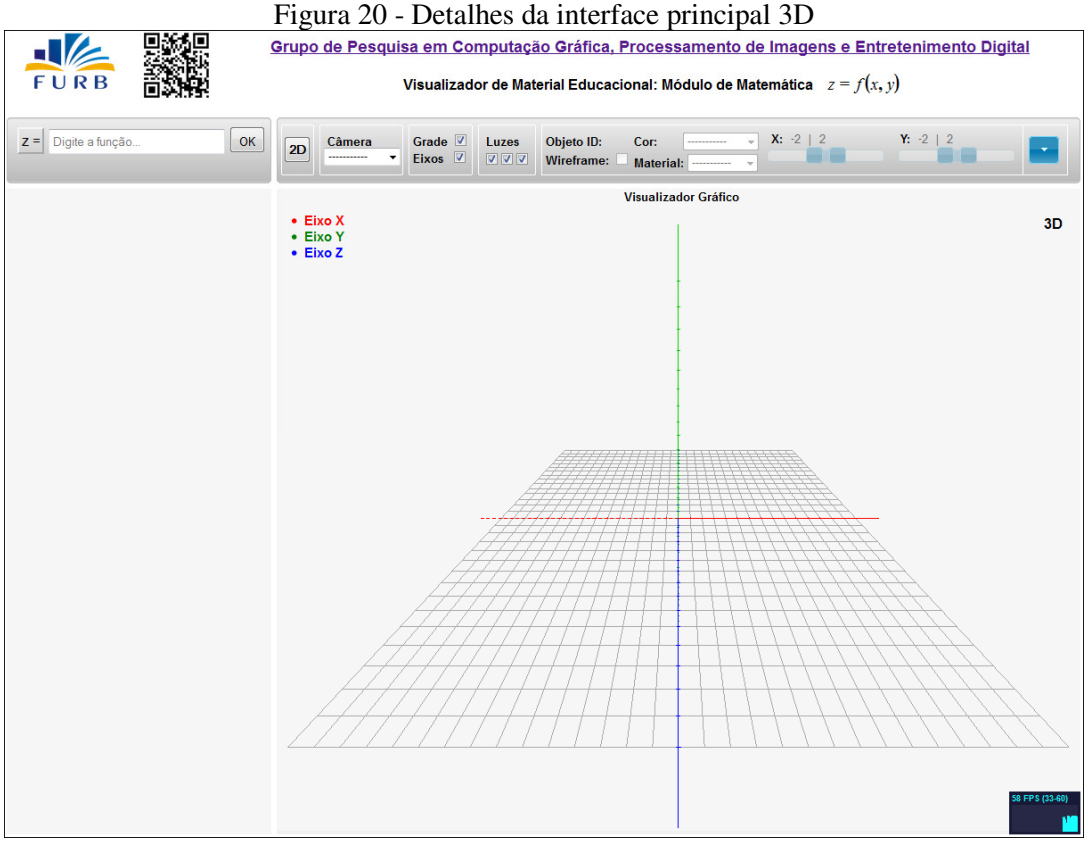
Ao iniciar o visualizador de material educacional, o usuário é remetido a interface principal da aplicação, que é a tela da versão 3D. Essa interface é formada basicamente por um menu principal, que contém a caixa de texto para digitação da função e também os parâmetros de configuração da função e da cena. É formada também por um visualizador gráfico localizado ao centro da tela e por uma lista onde serão adicionadas as funções digitadas. A Figura 20 ilustra a interface principal e os elementos de tela.

Antes de digitar uma função, o usuário pode escolher qual será o contradomínio da função, clicando no botão do lado esquerdo da caixa de texto. Essa escolha implicará em quais dos três eixos de orientação do visualizador gráfico a função será exibida e também quais variáveis de domínio serão parametrizáveis. Como o visualizador gráfico usa sistema de orientação mão direita, se o controdomínio escolhido for `z`, a função será exibida no eixo frontal, se o controdomínio escolhido for `y`, a função será exibida no eixo vertical e se o controdomínio escolhido for `x`, a função será exibida no eixo horizontal.

Após realizar a escolha do contradomínio, o usuário pode digitar a função, seguindo as convenções impostas, e clicar no botão `OK`. Neste momento, se não houver erros na sintaxe da função, o objeto gráfico resultante é exibido no devido eixo do visualizador gráfico com uma cor aleatória pré-definida e o texto da função é adicionado na lista de funções. Para remover a função da lista, o usuário deve clicar no ícone da lixeira na função desejada.

Por padrão, todas as funções digitadas são geradas num intervalo de valores de -2 a 2 para cada variável de domínio parametrizável. Isso que dizer que uma função criada com o contradomínio em `z` terá disponível as variáveis de domínio parametrizáveis `x` e `y`, e assim sucessivamente.

Figura 20 - Detalhes da interface principal 3D




Após a etapa de criação e exibição da função, o usuário pode interagir com a mesma, desde que tenha feito a seleção dessa função. Para fazer a seleção de uma função na versão 3D, existem duas formas, a primeira é usando um clique simples sobre o texto da função na lista de funções e a segunda forma é usando um duplo-clique sobre objeto gráfico da função no visualizador gráfico. Para desfazer a seleção, basta usar um duplo-clique em qualquer espaço do visualizador gráfico que não seja uma função.

Ao efetuar a seleção de uma função, no mesmo instante é exibido a sua BoundingBox e também são habilitados os componentes de tela dos seus parâmetros de configuração no menu principal, permitindo alterações de cor, material e os valores de suas variáveis de

domínio parametrizáveis. Qualquer alteração desses parâmetros é aplicado imediatamente na função selecionada, permitindo que o usuário veja em tempo real o resultado no visualizador gráfico.

A interação do usuário com o visualizador gráfico ocorre através do mouse. Como a aplicação não permite alteração de posicionamento e orientação do objeto gráfico, todas as operações de rotação, translação e zoom são aplicadas na câmera do visualizador gráfico, dando a impressão de que é o objeto que está sendo alterado. Para rotacionar a câmera em qualquer um dos três eixos, o usuário deve pressionar o botão esquerdo do mouse e arrastar em uma das direções `x`, `y` ou `z`. Para transladar o objeto em qualquer um dos três eixos, o usuário deve pressionar o botão direito do mouse e arrastá-lo. Para adicionar ou remover zoom na câmera, o usuário deve rolar o botão scroll do mouse para cima ou para baixo, respectivamente.

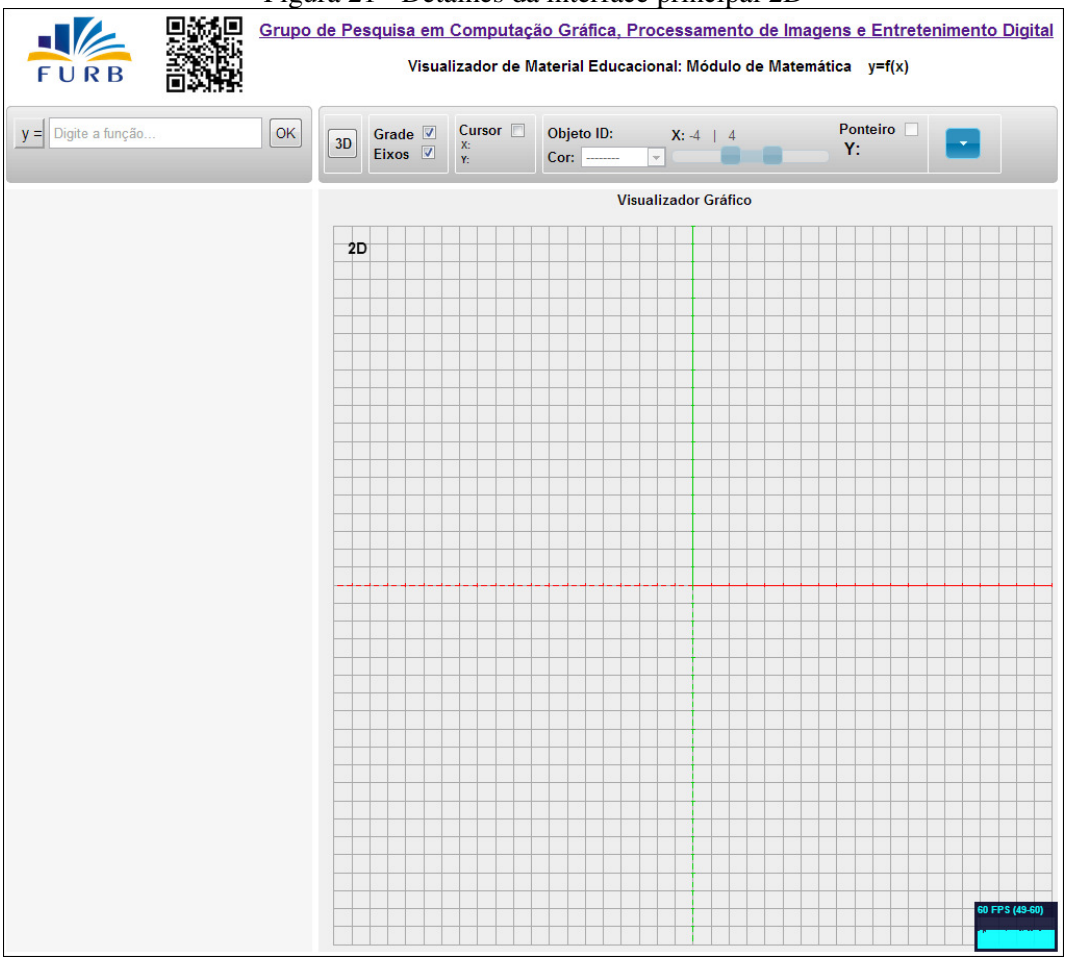
Para tornar a aplicação mais interativa e fácil de utilizar, foi desenvolvido um menu de ajuda e também exemplos didáticos que podem ser acessados através do menu principal. Para acessar os exemplos, o usuário deve acessar o menu principal em seguida clicar no menu `Exemplos didáticos`, onde é exibida uma janela com dois exemplos de paraboloides. Para adicioná-los na cena, basta clicar sobre o texto da função. Para acessar o menu de ajuda, o usuário deve acessar o menu principal e clicar em `Ajuda`. O Apêndice A mostra mais detalhes do manual de usuário.

Para mudar o modo de operação do visualizador de material de educacional matemático para a versão 2D, o usuário deve clicar no botão `2D` localizado no menu principal.

#### 3.3.3.2 Visualizador de material educacional matemático 2D

Ao alterar o visualizador de material educacional matemático para modo de operação 2D, a aplicação carrega a interface principal da versão, conforme apresentado na Figura 21. O funcionamento de criação e exibição de uma função 2D é exatamente o mesmo da versão 3D.

Figura 21 - Detalhes da interface principal 2D




Na versão 2D a câmera é ortográfica e tem uma visão superior do plano cartesiano, dando visibilidade apenas para os eixos `x` e `y`, que são os eixos disponíveis neste modo de operação. Além disso, nessa versão aparece mais uma opção no menu principal: um recurso chamado `Cursor`, que quando habilitado passa a mostrar duas linhas auxiliares no visualizador gráfico indicando a posição `x`, `y` exata no plano.

Outra diferença dessa versão é que existe apenas uma maneira de fazer a seleção de uma função, que é usando um clique simples sobre a função na lista. E para desfazer a seleção basta usar um clique simples em qualquer espaço do visualizador gráfico. Ao efetuar a seleção de uma função, além das opções de interação já existentes na versão 3D, esta versão possui uma opção chamada de ponteiro, que só é possível de ser usado quando o cursor também estiver habilitado. O ponteiro, quando habilitado, passa a mostrar e movimentar uma circunferência sobre o resultado da função conforme o mouse é movimentado na cena.

A implementação do `Cursor` e do `Ponteiro` na versão 2D foram baseadas em duas sugestões feitas pelo professor de matemática Londero (2013) que identificou a necessidade

de mostrar ao usuário a posição do mouse e o resultado da função de `x` ou `y` referente aquela posição do mouse.

A interação do usuário com o visualizador gráfico nesta versão também é feita através do mouse, porém nesta versão somente a operação de zoom está disponível e funciona da mesma forma que na versão 3D. O usuário deve rolar o botão *scroll* do mouse para cima ou para baixo para aumentar ou diminuir o zoom, respectivamente.

## 3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta uma aplicação web para visualização de material educacional, na qual fosse possível criar e exibir uma função matemática e interagir com a mesma em uma cena. O VisEdu-MAT encontra-se disponível para uso na web, ver VisEdu-MAT (2013).

O objetivo inicial deste trabalho era criar um visualizador usando somente WebGL, capaz de validar e exibir todos os tipos de equações matemáticas. Porém após um estudo matemático dos tipos de equações, identificou-se uma enorme abrangência nesta área que dificultaria a validação das expressões digitadas e que poderia gerar um possível atraso no cronograma previsto, sendo necessário restringir o objetivo do trabalho para apenas funções matemáticas explícitas.

Inicialmente foi desenvolvido um protótipo de tela usando HTML5 baseado no aplicativo Quick Graph (2013), disponível para a plataforma iOS. Esse protótipo permitia somente a exibição de funções pre-definidas para fins de testes no elemento canvas. Como o novo objetivo era que a aplicação validasse e exibisse qualquer função matemática explícita, buscou-se um validador de sintaxe para as funções, visto que desenvolvê-lo não fazia parte do objetivo principal deste trabalho. Optou-se então por utilizar um validador pronto chamado `Parser`. O validador, que vinha nas primeiras versões da biblioteca Three.js e foi descontinuado, tem suas limitações porém atendeu as principais necessidades impostas, validandos todas as operações aritméticas básicas e algumas funções trigonométricas.

Conforme prospoto nos objetivos, foi implementada uma versão 2D para a aplicação, que precisou de alguns poucos ajustes de interface e código fonte, como a câmera e os eixos. Foram removidas as luzes e a opção de alterar o material do objeto, uma vez que o efeito das luzes só pode ser percebido em objetos tridimensionais com materiais especiais, sendo que o tipo de material usado para criar um objeto gráfico 2D era do tipo básico.

O método de desenvolvimento seguido na implementação deste trabalho assemelha-se com o processo incremental da engenharia de software, onde foi criada uma lista de tarefas a serem feitas, uma lista de tarefas feitas e uma lista de tarefas sendo feitas. Conforme uma

tarefa fosse concluída, a mesma era removida de uma lista para outra e assim sucessivamente até terminarem todas.

### 3.4.1 Testes de desempenho da aplicação

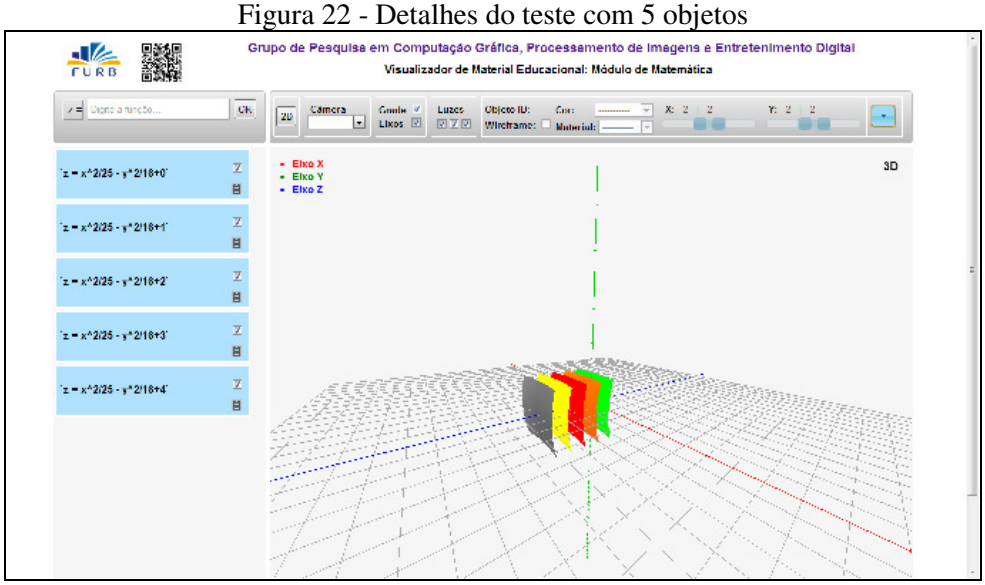
Para avaliar o desempenho da aplicação foi realizada somente a operação de adição de novas funções no visualizador gráfico. Os testes foram feitos em três navegadores diferentes que foram o Firefox, o Chrome e o Opera (descritos na seção 3.3.1). O Internet Explorer não foi utilizado nos testes devido a sua incompatibilidade com o WebGL.

As ferramentas utilizadas para fazer o levantamento das informações no Chrome e no Opera foi o Google Developer Tools que é uma ferramenta embutida no próprio navegador, e para o navegador Firefox foi usado o Firebug versão 1.12.5.

Os critérios utilizados para testar o desempenho da aplicação foram, o processamento de Frames Por Segundo (FPS) em relação a quantidade de objetos adicionados na cena, o consumo de memória durante a criação dos objetos e o consumo de espaço em disco usado pelo recurso de armazenamento local.

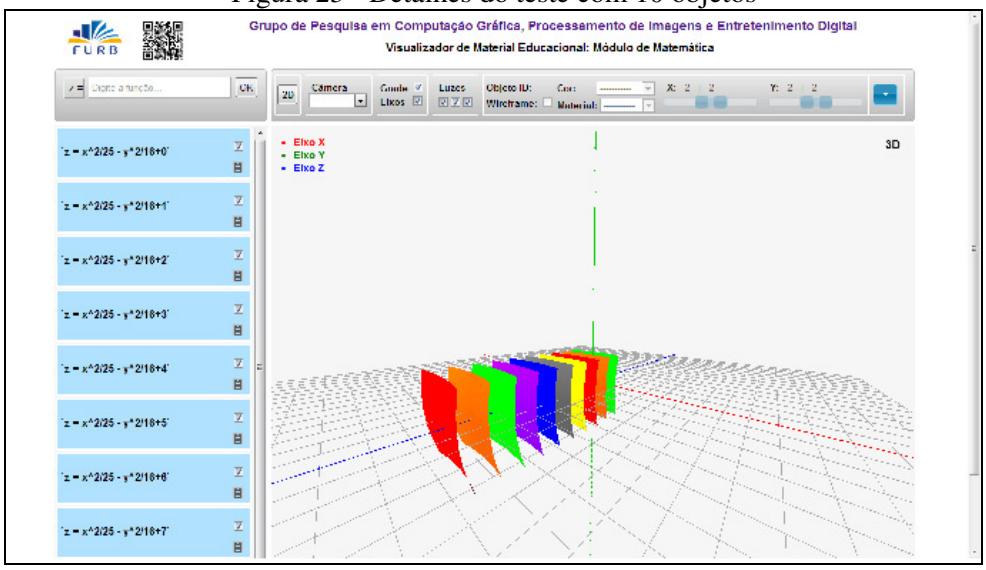
Foram criadas então 6 baterias de testes. O objeto escolhido para os testes foi uma paraboloide hiperbólica. O primeiro teste adicionava 5 objetos na cena posicionados no eixo z. A Figura 22 exibe os detalhes do teste.

Figura 22 - Detalhes do teste com 5 objetos



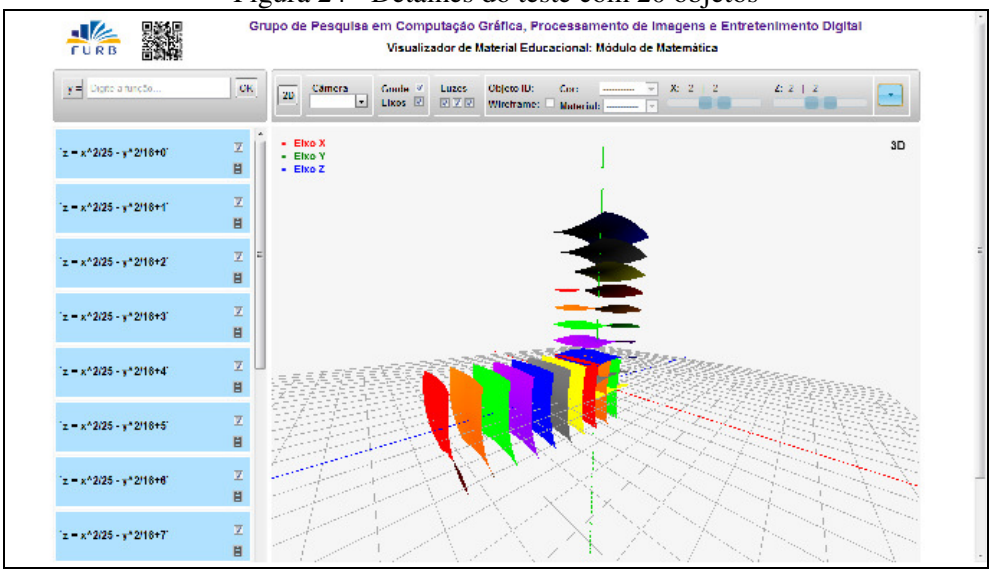

O segundo teste adicionava 10 objetos na cena, posicionados também no eixo z. A Figura 23 exibe os detalhes do teste.

Figura 23 - Detalhes do teste com 10 objetos



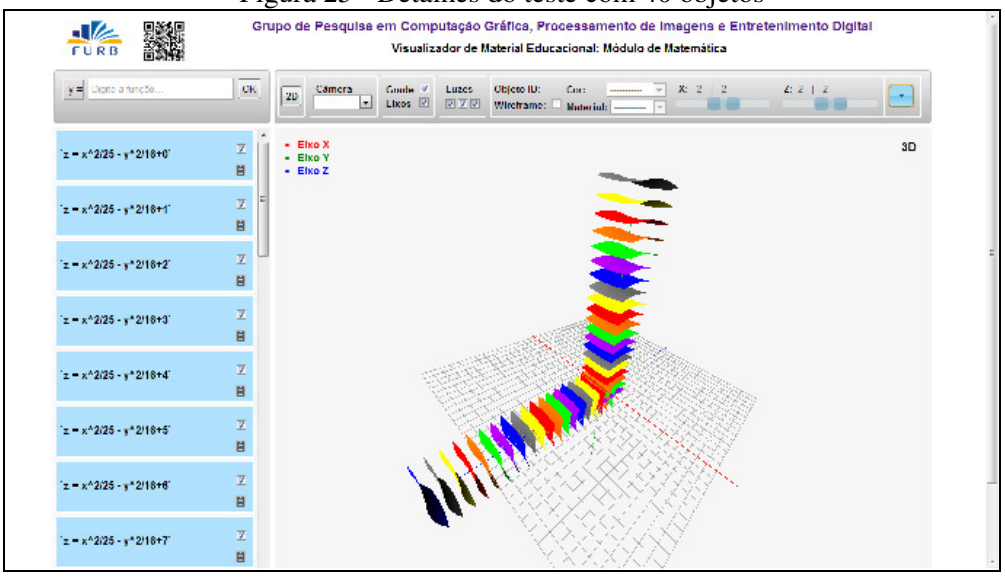

O terceiro teste adicionava 20 objetos na cena, divididos em 10 objetos no eixo z e os outros 10 no eixo y. A Figura 24 exibe os detalhes do teste.

Figura 24 - Detalhes do teste com 20 objetos



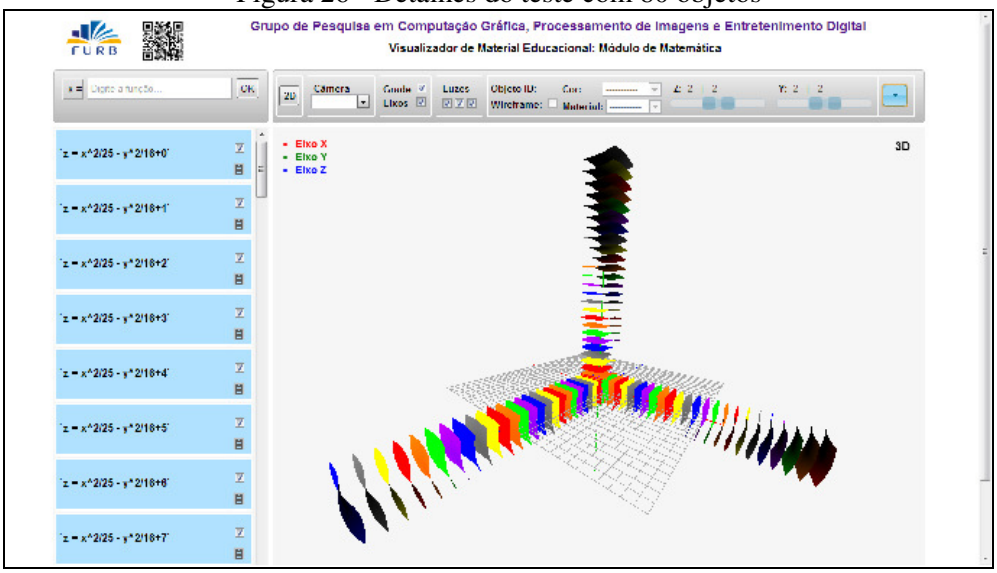

O quarto teste adicionava 40 objetos na cena, divididos em 20 objetos na eixo z e os outros 20 no eixo y. A Figura 25 mostra os detalhes do teste.

Figura 25 - Detalhes do teste com 40 objetos



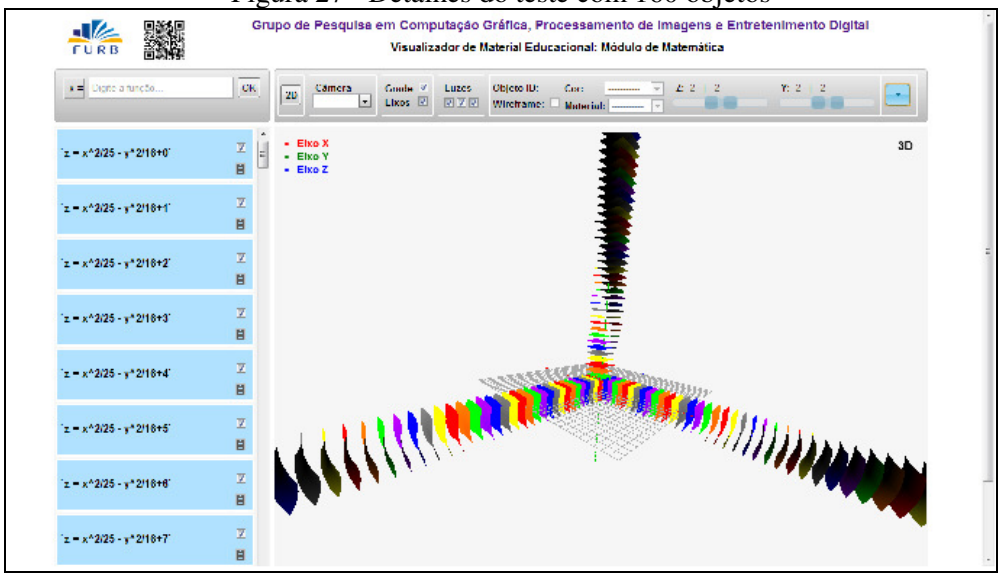

O quinto teste adicionava 80 objetos na cena, divididos em 24 objetos no eixo x, 28 objetos no eixo y e os outros 28 objetos no eixo z. A Figura 26 exibe os detalhes deste teste.

Figura 26 - Detalhes do teste com 80 objetos




O último teste adicionava 160 objetos na cena, divididos em 48 objetos no eixo x, 56 objetos no eixo y e 56 objetos no eixo z. A Figura 27 ilustra o teste.

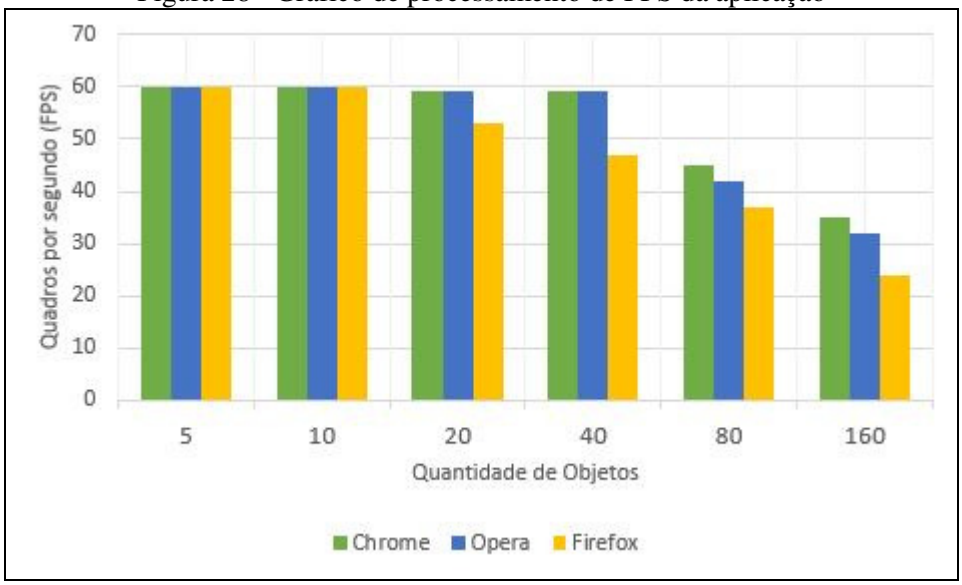
Figura 27 - Detalhes do teste com 160 objetos




Como todos os objetos criados são adicionados na posição zero do visualizador gráfico e o mesmo não permite a movimentação de um objeto selecionado, a criação dos objetos em várias posições na execução dos testes foi possível através do incremento do valor de `x`, `y` ou `z`, de acordo com a variável do contradomínio.

Analisando os resultados em relação ao processamento de quadros por segundo, todos os navegadores tiveram o mesmo desempenho com até 10 objetos na cena, que foram a primeira e a segunda bateria de testes. A partir da terceira, com 20 objetos na cena, o processamento de FPS da aplicação no novegador Firefox começou a diminuir. A Figura 28 apresenta o gráfico de processamento de FPS da aplicação.

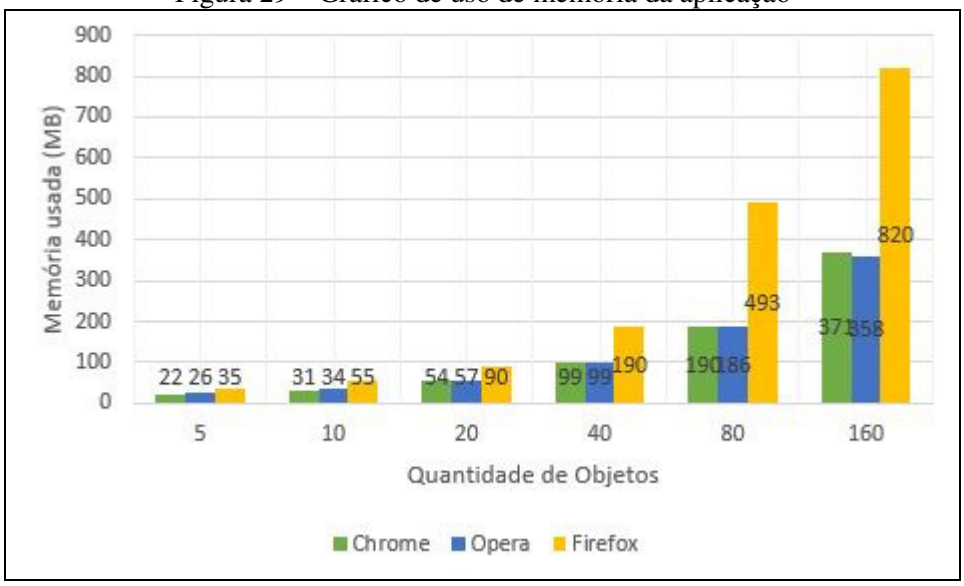
Figura 28 - Gráfico de processamento de FPS da aplicação




Em relação ao consumo de memória da aplicação durante as 6 baterias de testes, o Firefox demonstrou uma grande diferença dos outros dois navegadores. O aumento do uso de

memória no Firefox é percebida principalmente quando há interação do usuário com o visualizador gráfico, ocasionando eventuais travadas nas operações de zoom, rotação e translação. A Figura 29 exibe o gráfico de uso de memória da aplicação nos 3 navegadores testados.

Figura 29 – Gráfico de uso de memória da aplicação

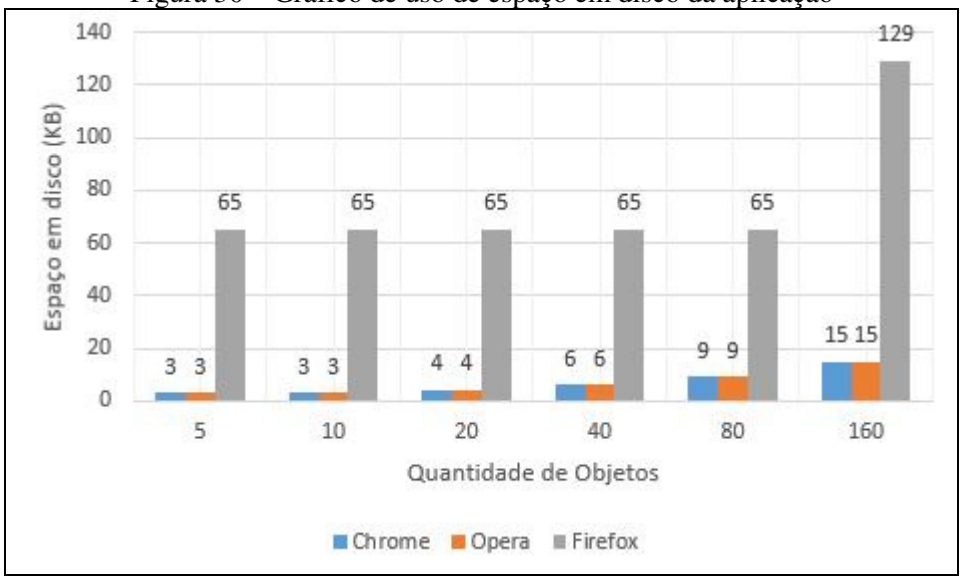



E por último, no que diz respeito ao consumo de espaço em disco da aplicação durante as baterias de testes, os navegadores Chrome e Opera tiveram exatamente o mesmo desempenho, ao contrário do Firefox, que criava arquivos temporários notavelmente maiores. A diferença percebida é que nos navegadores Chrome e Opera, os arquivos temporários criados pelo recurso de armazenamento local não liberavam o espaço alocado mesmo após remover as funções da cena. Já no Firefox, apesar de criar arquivos maiores, o gerenciamento deste se mostrou aprimorado, uma vez que o espaço em disco usado era liberado na medida que fossem removidas as funções da cena.

O diretório onde podem ser encontrados os arquivos criados pelo recurso de armazenamento local do HTML5 varia conforme o navegador. No Chrome o diretório usado é `%homepath%\appdata\Local\Google\Chrome\User Data\Default\Local Storage\`. No Opera ele se encontra no seguinte diretório: `%homepath%\appdata\Roaming\Opera Software\Opera Stable\Local Storage\`. E no Firefox o diretório usado é `%homepath%AppData\Roaming\Mozilla\Firefox\Profiles\2q3w3vjd.default\`.

A Figura 30 exibe um gráfico que detalha os resultados do desempenho em relação ao espaço em disco usado.

Figura 30 – Gráfico de uso de espaço em disco da aplicação




Analisando os resultados obtidos, percebe-se que o desempenho da aplicação variou em relação ao navegador utilizado. Quando executada nos navegadores Chrome ou Opera, os resultados em todos os testes foram superiores e similares, já o Firefox demonstrou inferioridade em todos os testes. A razão da similaridade do Chrome e do Opera é pelo fato de ambos usarem o WebKit como motor de renderização HTML e o Google V8 JavaScript Engine como motor JavaScript.

# 4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de uma aplicação web para visualização de funções matemáticas.

A biblioteca Three.js, usada para abstrair a implementação WebGL do visualizador gráfico, se mostrou simples de usar e muito eficiente, porém deixou muito a desejar no que diz respeito a documentação, sendo esse um fator que eventualmente dificultava a implementação de certas partes da aplicação.

Mesmo com as dificuldades encontradas, com exceção da implementação de textura, todos os outros objetivos propostos foram alcançados com êxito. A implementação de textura não foi realizada pois foi considerada desnecessária, uma vez que o visualizador gráfico não representa um mundo ou um cenário de jogo onde existe a necessidade de renderizar os mais diversos tipos de objetos e seus diferentes tipos de texturas. Além de já permitir usar recursos de materiais (brilhante) que proporcionam boas sensações visuais de profundidade para os objetos 3D.

O visualizador gráfico representa não só um espaço tridimensional ou bidimensional que possui uma grade e eixos de orientação, mas também permite a exibição do objeto resultante da função digitada.

A validação das expressões das funções digitadas através da classe `Parser` também mostrou-se útil validando todas as operações aritméticas básicas e algumas funções trigonométricas. Com alguns ajustes nas chamadas dos métodos de validação foi possível usá-los tanto para a implementação 3D como para a 2D.

Como principal limitação desta aplicação, destaca-se a incapacidade de validar funções implícitas.

## 4.1 EXTENSÕES

Durante as etapas finais do desenvolvimento deste trabalho, foram identificados alguns pontos que podem ser melhorados e algumas sugestões de extensões, que são:

a) validar funções implícitas tanto na versão 3D quanto na versão 2D;
b) adicionar um editor de funções no formato MathML;
c) permitir o uso da biblioteca ASCIIMathML nos outros navegadores;
d) adicionar um parâmetro de configuração para que o usuário possa escolher o nível de detalhe do objeto;
e) permitir que o usuário salve uma função na lista de funções didáticas do menu principal;

f) adicionar um botão para alterar o sistema de coordenadas do plano cartesiano (orientação dos eixos);
g) permitir que o usuário inspecione os valores de uma função 3D como já é feito nas funções 2D através do `Ponteiro`;
h) agrupar os botões `2D` e o `Menu principal` na primeira parte da barra de menus;
i) adicionar valores numéricos nos eixos de orientações;
j) adicionar setas de orientação nos eixos de orientação;
k) permitir que o usuário recolha o cabeçalho para aumentar o espaço do visualizador gráfico.

# REFERÊNCIAS BIBLIOGRÁFICAS

ANYURU, Andreas. **Professional WebGL programming:** developing 3D graphics for the web. Chichester: John Wiley & Sons, 2012.

ARAÚJO, Luciana P. **ADUBOGL** - aplicação didática usando a biblioteca OpenGL. 2012. 77 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

CARRIJO, Gilberto A. et al. **Associando realidade virtual ao ensino de matemática fundamental**. Uberlândia, 2007. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/svr/2007/0050.pdf>. Acesso em: 27 jan. 2014.

FLANAGAN, D. **Javascript**: the definitive guide. 5ª Edição. ed. Sebastopol, CA, EUA: O'Reilly Media Inc., 2008.

GARBI, Gilberto G. **O romance das equações algébricas**. 3. ed. São Paulo: Livraria da Física, 2009.

GROSSKURTH, Alan; GODFREY, Michael W. **Architecture and evolution of the modern web browser.** Waterloo, 2006. Disponível em: <http://grosskurth.ca/papers/browser-archevol-20060619.pdf>. Acesso em: 31 maio 2013.

KHRONOS. **WebGL specification**. [S.l.], 2011. Disponível em: <https://www.khronos.org/registry/webgl/specs/1.0/>. Acesso em: 27 mar. 2013.

KRAUSS, José R. **Repositório TCC_JoseKrauss**. Blumenau, 2013. Disponível em: <https://bitbucket.org/daltonreis/tcc_josekrauss>. Acesso em: 8 ago. 2013.

LONDERO, Evandro F. **Entrevista sobre elicitação de requisitos.** Entrevistadores: José Ricardo Krauss e Dalton Solano dos Reis. Blumenau. 2013. Entrevista feita através de conversação – não publicada.

MALAQUIAS, Fernanda F. O. et al. VirtualMat: um ambiente virtual de apoio ao ensino de matemática para alunos com deficiência intelectual. **Revista Brasileira da Informática na Educação**, Uberlândia, v. 20, n. 2, p. 17-30, Ago. 2012.

MATEMÁTICA DIDÁTICA. **Equação exponencial.** [S.l.], [2013?]. Disponível em: <http://www.matematicadidatica.com.br/EquacaoExponencial.aspx>. Acesso em: 29 maio 2013.

MICROSOFT. **Get started using HTML5**. [S.l.], 2013. Disponível em: <http://www.microsoft.com/web/post/get-started-using-html5?sf1284466=1>. Acesso em: 28 maio 2013.

PEREIRA, Daniel. **Desenvolvimento de um motor de jogos 3D utilizando WebGL.** 2012. 65 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

QUICK GRAF. **Quick Graph:** scientific graphing calculator. [S.l.], 2013. Disponível em: <https://itunes.apple.com/br/app/quick-graph-your-scientific/id292412367?l=en&mt=8>. Acesso em: 29 out. 2013.

VISEDU-MAT. **Visualizador de material educacional, módulo de matemática**. Blumenau, 2013. Disponível em: <http://gcg.inf.furb.br/visedu/TCC_Jose/Main3D.html>. Acesso em: 22 nov. 2013.

W3C BRASIL. **HTML5 -** curso W3C escritório Brasil. [S.l.], [2013?]. Disponível em: <http://w3c.br/pub/Cursos/CursoHTML5/html5-web.pdf>. Acesso em: 26 maio 2013.

W3SCHOOLS. **HTML introduction.** [S.l.], 2013a. Disponível em: <http://www.w3schools.com/html/html_intro.asp>. Acesso em: 21 mar. 2013.

_____. **HTML5 introduction.** [S.l.], 2013b. Disponível em: <http://www.w3schools.com/html/html5_intro.asp>. Acesso em: 21 mar. 2013.

WIKIPEDIA. **Função.** [S.l.], 2013a. Disponível em: <http://pt.wikipedia.org/wiki/Função_(matemática)>. Acesso em: 12 set. 2013.

_____. **Three.js.** [S.l.], 2013b. Disponível em: <http://en.wikipedia.org/wiki/Threejs>. Acesso em: 12 out. 2013.

ZILL, Dennis G. **Equações diferenciais:** com aplicações em modelagem. 2. ed. [S.l.]: Cengage Learning, 2011.

# APÊNDICE A – Manual de ajuda do usuário

Neste apêndice é apresentado o manual de ajuda do usuário, que pode ser acessado através do menu principal da aplicação. O manual foi dividido em várias partes para facilitar a visualização e o entendimento.

Figura 31 - Manual de usuário parte 1

O **VisEdu-MAT** é um aplicativo que visa auxiliar o ensino de funções matemáticas para os alunos do curso de Bacharelado em Ciência da Computação.

**1 - Padrão de notação do VisEdu-MAT**

O VisEdu-Mat usa o padrão de notação do ASCIIMathML para exibir uma função. Portanto deve-se seguir algumas convenções impostas pela biblioteca, que são:
Para operadores aritméticos:

adição = **+** (Ex: 3 + x)
subtração = **-** (Ex: 3 - x)
divisão = **/** (Ex: 3 / x)
multiplicação = ***** (Ex: 3 * x)
potenciação = **^** (Ex: 3 ^ x)
radiciação = **sqrt(x)** (Ex: sqrt(3))
logaritmo = **log(x)** (Ex: log(3))

Para funções trigonométricas:

seno = **sin(x)**
cosseno = **cos(x)**
tangente = **tan(x)**

Para funções trigonométricas inversas:

arco seno = **asin(x)**
arco cosseno = **acos(x)**
arco tangente = **atan(x)**

Exemplo de função no VisEdu-MAT:

**x^2 + sin(y)/2** = $x^2 + \frac{\sin(y)}{2}$

Figura 32 - Manual de usuário parte 2

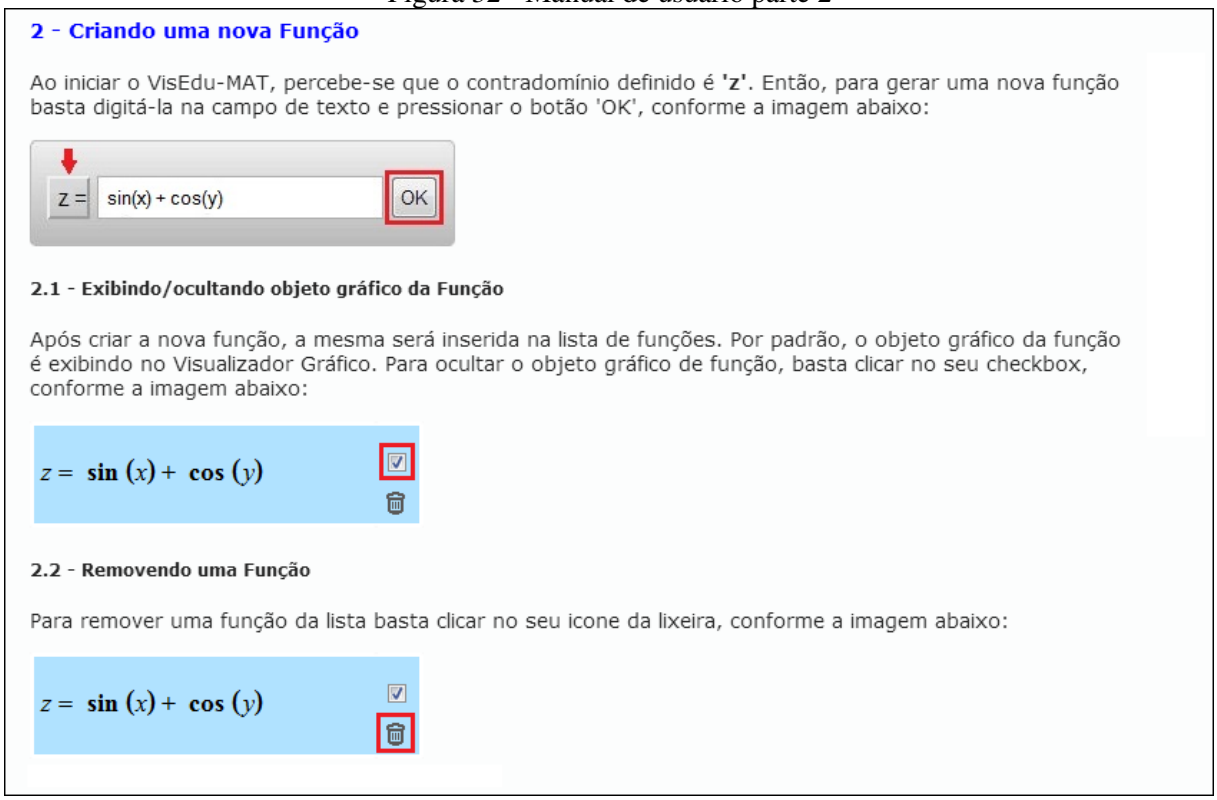



Figura 33 - Manual de usuário parte 3

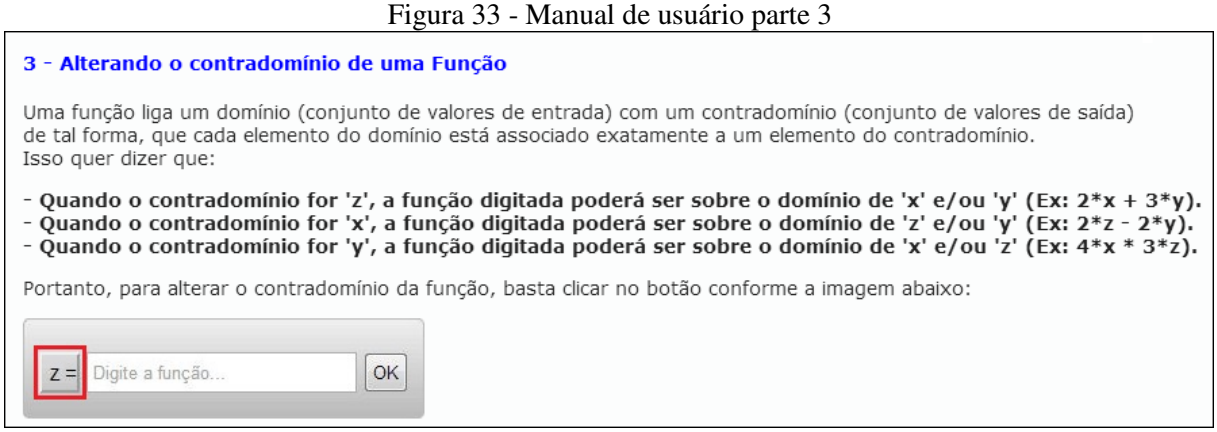


Figura 34 - Manual de usuário parte 4

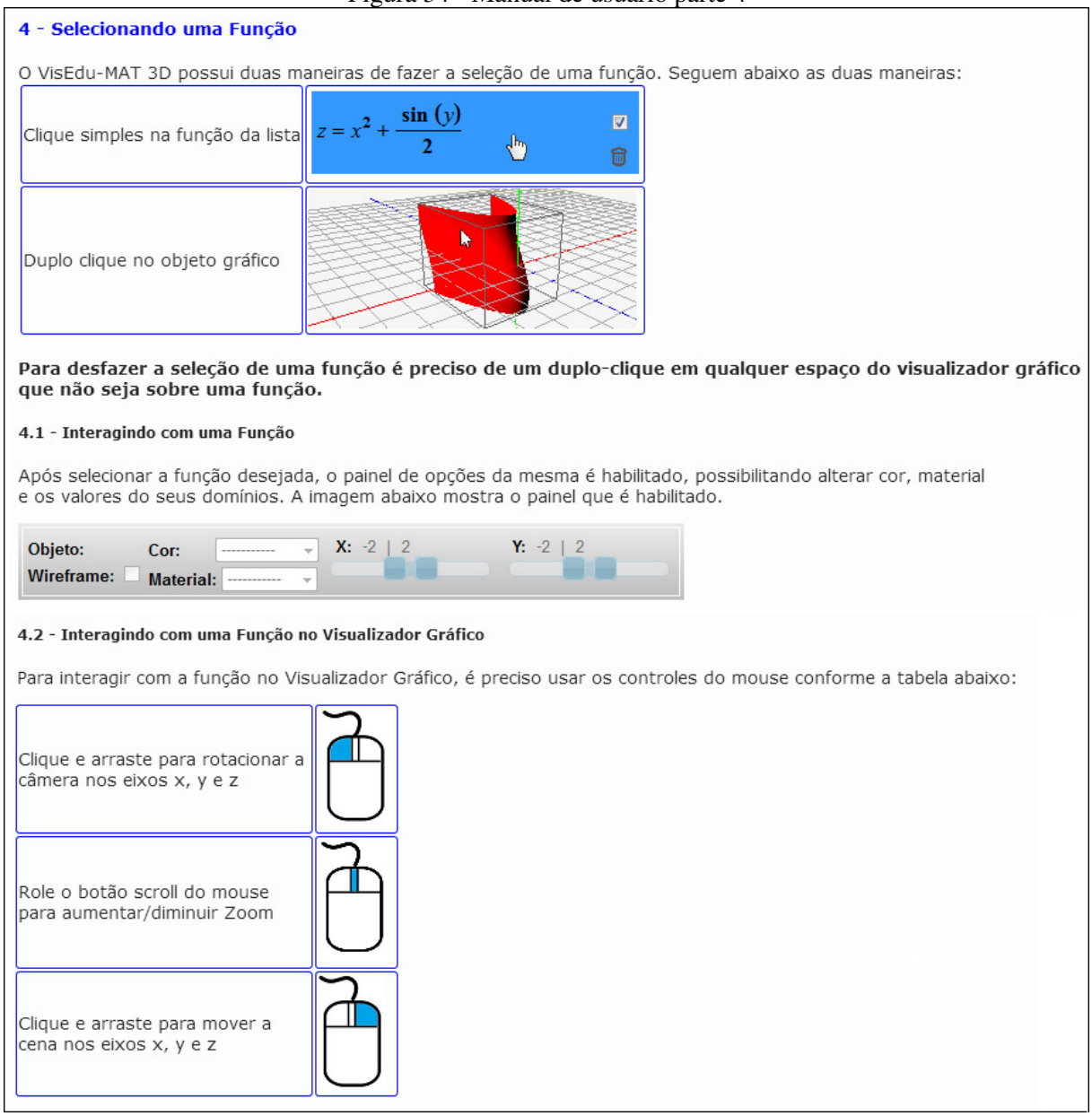



Figura 35 - Manual de usuário parte 5 e 6