

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

**VISEDU-MAT 2.0: VISUALIZADOR DE MATERIAL
EDUCACIONAL - MÓDULO DE MATEMÁTICA**

OSVALDO BAY MACHADO

**BLUMENAU
2014**

2014/2-16

OSVALDO BAY MACHADO

VISEDU-MAT 2.0: VISUALIZADOR DE MATERIAL

EDUCACIONAL - MÓDULO DE MATEMÁTICA

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

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

**BLUMENAU
2014**

2014/2-16

VISEDU-MAT 2.0: VISUALIZADOR DE MATERIAL EDUCACIONAL - MÓDULO DE MATEMÁTICA

Por

OSVALDO BAY MACHADO

Trabalho de Conclusão de Curso 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. Aurélio Faustino Hoppe, Mestre – FURB

Membro: Prof. Miguel Alexandre Wisintainer, Mestre – FURB

Blumenau, 10 de dezembro de 2014

Dedico este trabalho aos meus familiares e amigos, de forma especial a aqueles que me deram forças e motivação para a conclusão deste.

AGRADECIMENTOS

Agradeço a minha família e especialmente a minha esposa, que desde o primeiro momento me incentivou a ingressar e concluir o curso.

Ao meu orientador, Dalton Solano dos Reis, por ter dado o apoio necessário e fazer acreditar em mim para a conclusão deste trabalho.

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

Aos professores da universidade, que possibilitaram a construção dos meus conhecimentos que desenvolvo no meu ambiente profissional.

Aos amigos, por me dar forças para seguir em frente e entender minha ausência nas confraternizações.

A força não provém da capacidade física.
Provém de uma vontade indomável.

Mahatma Gandhi

RESUMO

Este trabalho apresenta o desenvolvimento de um aplicativo para iPad que permite a visualização de funções matemáticas explícitas usando o OpenGL ES. O aplicativo tem como principal objetivo a criação, validação e exibição de uma função em uma área de desenho. Esta área de desenho compreende a uma área tridimensional (3D) ou bidimensional (2D), permitindo que o usuário use nas funções as variáveis de domínio x, y e z para 3D ou x, y para 2D. Para a obtenção das coordenadas dos pontos que formam o desenho é usada a biblioteca DDMathParser que realiza a validação e cálculo a partir da função matemática. Ao inserir uma nova função matemática o aplicativo permite escolher o contradomínio. O aplicativo ainda permite que o usuário interaja o objeto gráfico previamente selecionado. Os valores do domínio da função matemática são parametrizáveis. Há possibilidade de mudar a cor e a visibilidade da função matemática. O tipo de primitiva pode ser alterado no modo 3D. O aplicativo ainda possui uma biblioteca didática de funções matemáticas. Na avaliação do aplicativo, foram realizados testes de memória e desempenho. Como resultado o aplicativo foi desenvolvido atendendo as principais características encontradas nos trabalhos correlatos pesquisados.

Palavras-chave: Computação gráfica. Objective-C. iOS. OpenGL ES. DDMathParser.

ABSTRACT

This work presents the development of an iPad app that allows you to view explicit mathematical functions using OpenGL ES. The application's main objective is the creation, validation and display a function in a drawing area. This drawing area comprises a three-dimensional area (3D) or two dimensional (2D), allowing the user to use the functions of the domain variables x, y and z for 3D or x, y to 2D. To obtain the coordinates of the points that form the design is used to DDMathParser library that performs validation and calculation from the mathematical function. When inserting a new math function allows the application to choose the codomain. The application also allows the user to interact the previously selected graphic object. The values from the domain of mathematical function are parameterized. There is possibility to change the color and visibility of the mathematical function. The primitive type can be changed in 3D mode. The application also has an educational library of mathematical functions. In the application evaluation, memory and performance tests were performed. As a result the application was developed in view of the key features found in the related work surveyed.

Key-words: Computer graphics. Objective-C. iOS. OpenGL ES. DDMathParser.

LISTA DE FIGURAS

| | |
|---|----|
| Figura 1 – Função seno de x | 17 |
| Figura 2 – Função cosseno de x | 17 |
| Figura 3 – Função tangente de x | 17 |
| Figura 4 – Regra da mão esquerda e regra da mão direita | 19 |
| Figura 5 – Valida e calcula expressão matemática..... | 20 |
| Figura 6 – Cria e adiciona nova função..... | 21 |
| Figura 7 – Cria e adiciona novo operador | 22 |
| Figura 8 – Tela principal do VisEdu-MAT 1.0 | 23 |
| Figura 9 – Quick Graph em modo 2D e 3D, respectivamente | 24 |
| Figura 10 – Grafikal De Ageio em modo 2D e 3D, respectivamente | 25 |
| Figura 11 – Diagrama de casos de uso | 27 |
| Figura 12 – Diagrama de pacotes | 33 |
| Figura 13 – Diagrama de classes do pacote Utils..... | 33 |
| Figura 14 – Diagrama de classes do pacote View..... | 35 |
| Figura 15 – Diagrama de classes do pacote Controller..... | 37 |
| Figura 16 – Diagrama de sequência | 39 |
| Figura 17 – Detalhes do método geraPontos3DWithObjetoCena | 41 |
| Figura 18 – Continuação do método geraPontos3DWithObjetoCena | 42 |
| Figura 19 – Final do método geraPontos3DWithObjetoCena | 43 |
| Figura 20 – Detalhes do método carregaDesenhoFuncao3D | 45 |
| Figura 21 – Detalhes do programa Shader .vsh | 46 |
| Figura 22 – Detalhes do método mostraDesenhoFuncao3DWithPrimitiva | 48 |
| Figura 23 – VisEdu-MAT 2.0 modo 3D | 50 |
| Figura 24 – VisEdu-MAT 2.0 modo 2D | 52 |
| Figura 25 – Teste com 10 objetos..... | 54 |
| Figura 26 – Teste com 20 objetos..... | 54 |
| Figura 27 – Teste com 50 objetos..... | 55 |
| Figura 28 – Teste com 100 objetos..... | 55 |
| Figura 29 – Teste com 150 objetos..... | 56 |
| Figura 30 – Testes com 300 objetos | 56 |

| | |
|--|----|
| Figura 31 – Teste de memória utilizada | 57 |
| Figura 32 – Manual do usuário página 1 | 63 |
| Figura 33 – Manual do usuário página 2 | 64 |
| Figura 34 – Manual do usuário página 3 | 65 |
| Figura 35 – Manual do usuário página 4 | 66 |
| Figura 36 – Manual do usuário página 5 | 67 |
| Figura 37 – Manual do usuário página 6 | 68 |
| Figura 38 – Manual do usuário página 7 | 69 |
| Figura 39 – Manual do usuário página 8 | 70 |
| Figura 40 – Manual do usuário página 9 | 71 |
| Figura 41 – Manual do usuário página 10 | 72 |
| Figura 42 – Manual do usuário página 11 | 73 |
| Figura 43 – Manual do usuário página 12 | 74 |
| Figura 44 – Manual do usuário página 13 | 75 |
| Figura 45 – Manual do usuário página 14 | 76 |
| Figura 46 – Manual do usuário página 15 | 77 |
| Figura 47 – Função aritmética vetorial de adição | 78 |
| Figura 48 – Função aritmética escalar de multiplicação | 79 |
| Figura 49 – Função tangente | 79 |
| Figura 50 – Planos Coordenados | 81 |

LISTA DE QUADROS

| | |
|--|----|
| Quadro 1 – Descrição do caso de uso UC01 | 28 |
| Quadro 2 – Descrição do caso de uso UC02 | 28 |
| Quadro 3 – Descrição do caso de uso UC03 | 29 |
| Quadro 4 – Descrição do caso de uso UC04 | 30 |
| Quadro 5 – Descrição do caso de uso UC05 | 30 |
| Quadro 6 – Descrição do caso de uso UC06 | 31 |
| Quadro 7 – Descrição do caso de uso UC07 | 32 |
| Quadro 8 – Descrição do caso de uso UC08 | 32 |
| Quadro 9 – Testes de desempenho realizados no aplicativo | 57 |
| Quadro 10 – Comparaçao dos trabalhos correlatos e o VisEdu-MAT 2.0 | 58 |

LISTA DE SIGLAS

2D – 2 Dimensões

3D – 3 Dimensões

API – *Application Programming Interface*

CPU – *Central Processing Unit*

CSV – *Comma-Separated Values*

FPS – *Frames Per Second*

GLSL – *OpenGL Shading Language*

GPU – *Graphics Processing Unit*

IDE – *Integrated Development Environment*

RGB – *Red Green Blue*

RGBA – *Red Green Blue Alpha*

SDK – *Software Development Kit*

SRU – Sistema de Referência Universal

TSV – *Tab-Separated Values*

UML – *Unified Modeling Language*

VBO – *Vertex Buffer Object*

vDSP – *vector Digital Signal Processing*

WebGL – *Web Graphics Library*

SUMÁRIO

| | |
|---|-----------|
| 1 INTRODUÇÃO..... | 14 |
| 1.1 OBJETIVOS..... | 15 |
| 1.2 ESTRUTURA..... | 15 |
| 2 FUNDAMENTAÇÃO TEÓRICA..... | 16 |
| 2.1 FUNÇÕES MATEMÁTICAS | 16 |
| 2.1.1 Funções trigonométricas | 16 |
| 2.2 OPENGL ES..... | 18 |
| 2.3 DDMATHPARSER | 20 |
| 2.3.1 Validando e calculando expressões matemáticas..... | 20 |
| 2.3.2 Adicionando novas funções | 21 |
| 2.3.3 Adicionando novos operadores | 21 |
| 2.4 TRABALHOS CORRELATOS | 22 |
| 2.4.1 VisEdu-MAT 1.0 | 22 |
| 2.4.2 Quick Graph | 23 |
| 2.4.3 Grafikal De Ageio | 24 |
| 3 DESENVOLVIMENTO..... | 26 |
| 3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO..... | 26 |
| 3.2 ESPECIFICAÇÃO | 27 |
| 3.2.1 Diagrama de casos de uso | 27 |
| 3.2.1.1 Gerar função 2D ou 3D..... | 28 |
| 3.2.1.2 Mudar o contradomínio da função 2D ou 3D | 28 |
| 3.2.1.3 Selecionar função 2D ou 3D..... | 28 |
| 3.2.1.4 Alterar propriedades da função selecionada 2D ou 3D | 29 |
| 3.2.1.5 Excluir função 2D ou 3D | 30 |
| 3.2.1.6 Interagir com o aplicativo no modo 2D ou 3D | 30 |
| 3.2.1.7 Alternar entre o modo 2D e 3D | 31 |
| 3.2.1.8 Mudar primitiva 3D | 32 |
| 3.2.2 Diagrama de classes | 32 |
| 3.2.2.1 Pacote Utils..... | 33 |
| 3.2.2.2 Pacote View | 35 |
| 3.2.2.3 Pacote Controller | 36 |

| | |
|---|-----------|
| 3.2.3 Diagrama de sequência | 38 |
| 3.3 IMPLEMENTAÇÃO | 39 |
| 3.3.1 Técnicas e ferramentas utilizadas..... | 39 |
| 3.3.2 O visualizador de material educacional (VisEdu-MAT 2.0) | 40 |
| 3.3.2.1 Gerar nuvem de pontos da função matemática..... | 40 |
| 3.3.2.2 Carregar nuvem de pontos na memória da GPU | 44 |
| 3.3.2.3 Renderizar nuvem pontos na área de desenho | 45 |
| 3.3.3 Operacionalidade da implementação | 49 |
| 3.3.3.1 VisEdu-MAT 2.0 modo 3D | 49 |
| 3.3.3.2 VisEdu-MAT 2.0 modo 2D | 51 |
| 3.4 RESULTADOS E DISCUSSÃO | 52 |
| 3.4.1 Testes de desempenho do aplicativo..... | 53 |
| 3.4.2 Comparativo entre os trabalhos correlatos | 58 |
| 4 CONCLUSÕES..... | 59 |
| 4.1 EXTENSÕES | 59 |
| REFERÊNCIAS | 61 |
| APÊNDICE A – Manual do Usuário | 63 |
| APÊNDICE B – Framework Accelerate | 78 |
| ANEXO A – Planos Coordenados..... | 81 |

1 INTRODUÇÃO

De acordo com as últimas avaliações do Exame Nacional do Ensino Médio (ENEM), o Sistema de Avaliação da Educação Básica – Ministério da Educação (SAEB-MEC) e o *Program for International Student Assessment – Organization for Economic Co-operation and Development* (PISA-OECD), são baixos os índices de aprendizagem dos alunos no Brasil, principalmente nas áreas de ciências e matemática. A preocupação com o modelo de ensino praticado em salas de aula desperta interesse de um crescente número de professores por um fazer educativo voltado ao desenvolvimento cognitivo, criando programas de estudo e experiências que buscam transformar o fazer pedagógico, desenvolvendo pesquisas para auxiliar nesta transformação (MARTINS; MIOTTO; VILLAS-BOAS, 2011, p. 10-11).

Os avanços tecnológicos têm possibilitado aos alunos e professores buscar informações em diferentes meios e culturas, facilitando o processo de aprendizado. As ferramentas tecnológicas, assim como sistemas e aplicativos produzidos para a educação, podem ser utilizadas por educadores, a fim de despertar maior interesse em seus alunos.

A emergência de uma ferramenta tecnológica, que pode ser definida como uma tecnologia intelectual, possibilita, do ponto de vista instrumental, construir relações e correspondências novas. São propriamente essas relações que, ao transformarem os objetos e os sujeitos do conhecimento, reconfiguram as bases da ecologia cognitiva. (OLIVEIRA; VIGNERON, 2005, p. 50).

Segundo Barbosa (2005, p. 74), o novo currículo de matemática deve oferecer condições para que os alunos sejam capazes de alcançar abstração por entendimento de fórmulas ao verificar resultados numéricos, tabulares e gráficos, além de proporcionar maneiras de ensinar e aprender.

Diante do exposto, foi desenvolvido um aplicativo para ser utilizado como auxílio a professores e alunos na visualização dos resultados de funções matemáticas explícitas. Considerando a crescente evolução das tecnologias digitais disponíveis em dispositivos móveis, tais como *smartphones* e *tablets*, foi desenvolvido um aplicativo para iPad, buscando uma usabilidade simplificada, como um validador sintático para as funções inseridas e recursos de exibição gráfica 2 e 3D, que auxiliem a facilitar o aprendizado nas áreas de cálculo.

Este trabalho compõe o projeto VisEdu-MAT (2013) do Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital promovido pelo Departamento de Sistemas e Computação (DSC) da FURB. O VisEdu-MAT 2.0: Visualizador de Material Educacional - Módulo de Matemática, é uma portabilidade em arquitetura iOS da

versão anterior (VisEdu-MAT 1.0) desenvolvido em arquitetura web por Krauss (2013) (ver seção 2.4.1).

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver um aplicativo para iPad que permita inserir funções matemáticas e obter resultado gráfico.

Os objetivos específicos do trabalho são:

- a) visualizar uma função matemática previamente digitada nos modos 2D e 3D;
- b) validar a função ao inserir;
- c) interagir com a câmera na área de desenho;
- d) alterar parâmetros de domínio, contradomínio, cor e visibilidade;
- e) exibir *bounding box* no objeto selecionado no modo 3D;
- f) destacar e exibir *tracing* no objeto selecionado no modo 2D.

1.2 ESTRUTURA

A estrutura deste trabalho é formada por quatro capítulos, sendo que no primeiro é apresentada a introdução ao tema e em seguida são exibidos os objetivos e a estrutura do trabalho.

O segundo capítulo apresenta a fundamentação teórica necessária para o entendimento dos principais assuntos que compõe este trabalho.

O terceiro capítulo possui o desenvolvimento do aplicativo, onde são apresentados os requisitos da especificação, os casos de uso, os diagramas de pacotes, de classes e de sequência.

Por fim, o quarto capítulo expõe os resultados obtidos e as conclusões, além das sugestões de melhorias e extensões para a continuação deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo está organizado em quatro seções. A seção 2.1 elucida o conceito de formas de funções matemáticas. Na seção 2.2 são discutidos conceitos e recursos do *framework* OpenGL ES. Na seção 2.3 são demonstradas as principais funções da biblioteca DDMathParser. Por fim, na seção 2.4 traz trabalhos correlatos ao aplicativo proposto.

2.1 FUNÇÕES MATEMÁTICAS

Segundo Bonjorno, Giovanni e Giovanni Júnior (1994, p. 31-36), a definição da função é a relação entre dois ou mais conjuntos, aplicando uma regra, ou seja, uma lei de formação que relaciona os elementos de um grupo com o outro. Desta forma, considerando dois conjuntos numéricos não vazios, sendo o conjunto x conhecido como domínio e y como contradomínio e aplicando uma lei de formação $f(x)$, obtém-se uma função ao relacionar todos os elementos do domínio com elementos do contradomínio, neste caso conhecidos por conjunto imagem.

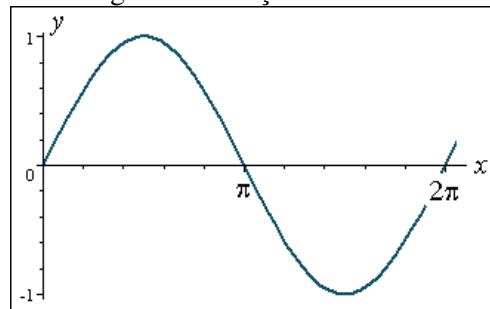
As funções podem ser explícitas ou implícitas. No caso de função explícita, os valores das variáveis x e y são obtidos de forma independente, enquanto que no caso de função implícita os valores das variáveis x e y são dependentes uma da outra na resolução da função (BALBO, 2014).

De acordo com Paiva (1995, p. 98) existem diversas formas de funções como: função linear, função exponencial, função modular, função quadrática, função exponencial, função logarítmica, função trigonométrica, entre outras.

2.1.1 Funções trigonométricas

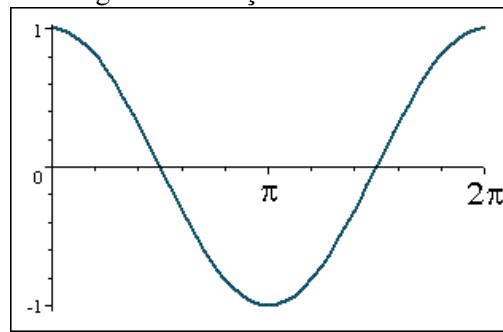
As funções trigonométricas são conhecidas como funções angulares, importantes no estudo dos triângulos e na modelação de fenômenos periódicos. Dentre as funções trigonométricas básicas em uso destacam-se a função seno de x dada pela notação $f(x) = \sin(x)$, a função cosseno de x dada pela notação $f(x) = \cos(x)$ e a função tangente de x dada pela notação $f(x) = \tan(x)$ (PAIVA, 1995, p. 479).

A Figura 1 mostra uma função trigonométrica na forma de onda senoidal dada pela função $f(x) = \sin(x)$.

Figura 1 – Função seno de x 

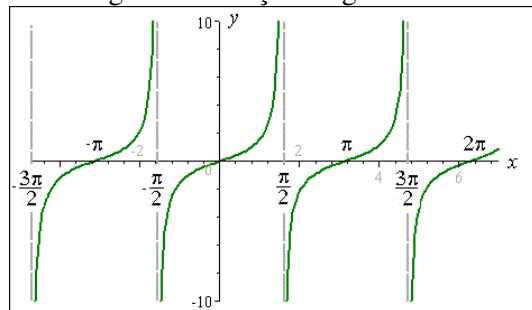
Fonte: Intmath (2014).

Na Figura 1 o eixo x representa o conjunto do domínio e o eixo y o conjunto do contradomínio. Os pontos que formam a onda senoidal são conhecidos como o conjunto imagem do contradomínio. Desta forma observando o eixo x os valores que variam entre 0 até o valor de 2π radianos corresponde ao período completo da circunferência. Aplicado os valores do eixo x a função $f(x) = \text{sen}(x)$ obteve-se os valores do conjunto imagem no intervalo de -1 e 1 conforme a função de seno (PAIVA, 1995, p. 523-526).

Figura 2 – Função cosseno de x 

Fonte: Intmath (2014).

A Figura 2 demonstra a função cosseno de x denotada por $f(x) = \cos(x)$ possui um comportamento semelhante à função seno de x . Os valores do conjunto imagem também estão no intervalo de -1 e 1 , porém sua onda se inicia defasada em 90 graus, sendo assim quando x for igual a 0 o valor de y será 1 (PAIVA, 1995, p. 530).

Figura 3 – Função tangente de x 

Fonte: Intmath (2014).

Por fim, a Figura 3 mostra a função tangente de x denotada por $f(x) = \tan(x)$ que possui um período de π e conjunto imagem de $-\infty$ até ∞ (PAIVA, 1995, p. 535-536).

2.2 OPENGL ES

Fundado em janeiro de 2000, o Grupo Khronos criou o OpenGL, uma *Application Programming Interface* (API) 3D permitindo o desenvolvimento de aplicações gráficas multiplataforma em sistemas operacionais *desktop* como Linux, Mac OS e Microsoft Windows (GINSBURG; MUNSHI; SHREINER, 2009, p. 1). Segundo Ginsburg, Munshi e Shreiner (2009, p. 2-3), devido ao amplo uso do OpenGL fez sentido iniciar o desenvolvimento de uma API 3D em padrão aberto, para dispositivos embarcados e computadores de mão, que tivesse o comprometimento sobre as restrições dos dispositivos como capacidade limitada de processamento, disponibilidade de memória e consumo de energia, surge então a especificação OpenGL ES.

Os recursos encapsulados nos objetos do *framework* OpenGL ES, utilizados por uma aplicação iOS, são contêineres reutilizáveis que guardam informações ou configurações necessárias para a renderização. Segundo Apple (2013a), os principais tipos são:

- a) *texture*: podendo ser amostrada por um *pipeline* gráfico, é utilizada como mapa de cores da imagem dentro das primitivas. Este mapa pode conter informações como um mapa normal ou iluminação pré-calculada;
- b) *vertex array*: mantém a configuração para atributos de vértices para serem lidos pelo *pipeline* gráfico;
- c) *buffer*: conhecido como *Vertex Buffer Object* (VBO) é o bloco de memória utilizado para controlar o processo de cópia de informações entre a aplicação e a API, iniciando o processo de desenho. Em contraste, utilizando um VBO o processo de desenho somente ocorrerá com um comando para modificar os conteúdos;
- d) *renderbuffer*: é uma simples imagem gráfica 2D em um formato específico, definido por cor, profundidade ou *stencil data*, que de acordo com MSDN (2013) é uma informação usada para ter mais controle sobre os *pixels* a serem renderizados, podendo ser utilizado em combinação com *buffer* de profundidade para uma renderização complexa, como sombreamentos e contornos;
- e) *framebuffer*: são objetos no destino final do *pipeline* gráfico em forma de contêiner de texturas e *renderbuffers* anexados, criando as configurações necessárias para renderização das imagens.

Além dos recursos citados, de acordo com Rideout (2010, p. 34-35), os *shaders* são pequenos pedaços de código que executam simultaneamente na *Graphics Processing Unit*

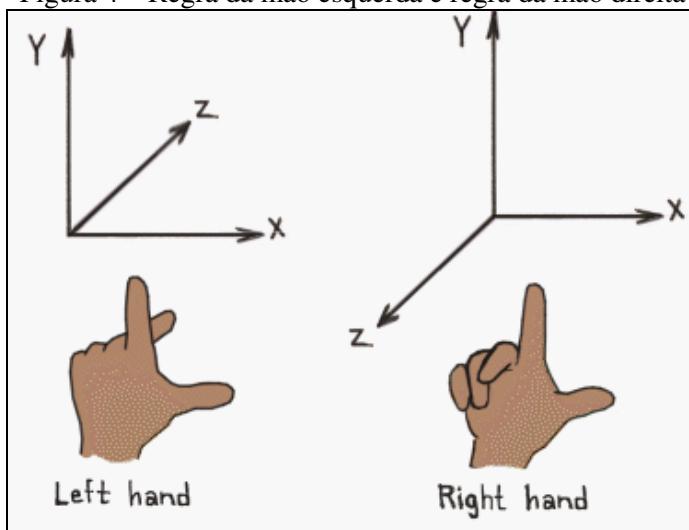
(GPU), e foram introduzidos como uma nova funcionalidade na versão OpenGL ES 2.0. Os *shaders* são escritos em uma linguagem própria chamada *OpenGL Shading Language* (GLSL) e são compilados em tempo de execução diretamente no iOS, subdividindo-se nas categorias: *vertex shader* e *fragment shader*.

O *vertex shader* implementa o propósito geral de programação para operar com vértices, podendo transformar matrizes de pontos, cálculos de luz para gerar uma cor por vértice e gerar ou transformar coordenadas de textura. Ele possui entradas como: *attribute*, utilizado para fornecer os dados por vértice; *uniforms*, informação usada para todos os vértices; *samplers*, um tipo específico de *uniforms* que representa textura, de uso opcional e *shader program*, um código fonte que descreve operações a serem executadas no vértice (GINSBURG; MUNSHI; SHREINER, 2009, p. 4-5).

O *fragment shader* recebe a saída do *vertex shader* e faz as transformações dos vértices e preparação das primitivas para renderização, que conforme Ginsburg, Munshi e Shreiner (2009, p. 181), produzem efeitos que incluem textura, luz por *pixel* e sombras.

Por fim, o processo de desenho realizado pelo *vertex shader* e *fragment shader* seguem um padrão no sistema de coordenadas. A Figura 4 mostra dois tipos de sistemas de coordenadas 3D utilizadas em APIs gráficas, conhecidas como regra da mão esquerda (*left hand*) e regra da mão direita (*right hand*) (LEARNOOPENGLES, 2014).

Figura 4 – Regra da mão esquerda e regra da mão direita



Fonte: LearnOpenGLES (2014).

O OpenGL ES utiliza o sistema de coordenadas na regra da mão direita *Right hand*, conforme a exibido na Figura 4. Neste caso quanto mais positivo for o valor do eixo de *z*, mais próximo o desenho estará do observador, enquanto na regra da mão esquerda isso será ao contrário.

2.3 DDMATHPARSER

A DDMathParser é uma biblioteca gratuita que tem por objetivo validar e realizar cálculos de expressões matemáticas. De acordo com sua licença de uso, qualquer pessoa pode copiar, modificar, publicar e distribuir partes ou a versão completa do software. É desenvolvida na linguagem de programação Objective-C, possuindo um conjunto de operadores e funções pré-definidas para realizar os cálculos matemáticos, além de permitir a extensão dos mesmos através de chamadas às classes e métodos em sua linguagem de programação (GITHUB, 2014c).

Uma das principais tarefas realizada por esta biblioteca é a “tokenização”, que visa identificar números, funções, variáveis e operadores na *string* da expressão matemática informada. Os números são tratados como valores positivos e permitem apenas o sinal ponto como separador decimal. As funções podem conter letras maiúsculas e minúsculas, dígitos decimais, e *underscores*. As variáveis devem ser prefixadas com o caractere \$. Por fim, os operadores podem ser do tipo comparação, *bit a bit*, lógicos ou padrão conhecido como operadores de soma, subtração, multiplicação, divisão e outros.

2.3.1 Validando e calculando expressões matemáticas

Há muitas maneiras de validar e realizar cálculos de expressões matemáticas dependendo da customização que se deseja fazer. A Figura 5 demonstra o processo de validação e cálculo de uma expressão matemática, que resulta em um desenho geométrico conhecido na matemática como parabolóide hiperbólico.

Figura 5 – Valida e calcula expressão matemática

```

1 NSString *expression = @"pow(&x, 2)/25-pow(&y, 2)/16";
2 DDMathStringTokenizer *tokenizer = nil;
3 NSDictionary *dictionary = nil;
4 NSError *error = nil;
5 float x = 1.5f;
6 float y = 2.0f;
7 float z = 0.0f;
8
9 tokenizer = [[DDMathStringTokenizer alloc] initWithString:expression
10                                     operatorSet:nil error:&error];
11 if (error != nil) {
12     return [NSString stringWithFormat:@"%@", [[error localizedDescription] UTF8String]];
13 }
14 dictionary = [NSDictionary dictionaryWithObjects:
15                 [NSArray arrayWithObjects:[NSNumber numberWithFloat: x],
16                              [NSNumber numberWithFloat: y], nil]
17                 forKey:[NSArray arrayWithObjects:@"x", @"y", nil]];
18
19 DDMathEvaluator *evaluator = [DDMathEvaluator defaultMathEvaluator];
20 z = [[evaluator evaluateString:expression withSubstitutions:dictionary error:&error] floatValue];

```

Neste exemplo o objetivo é validar e calcular apenas uma coordenada espacial x, y e z que compõe esse desenho. A classe DDMathStringTokenizer é responsável pela validação da

expressão matemática através do processo de tokenização. Se o processo de validação falhar um erro será retornado, senão as variáveis `$x` e `$y` da expressão matemática serão substituídas pelos valores `1.5f` e `2.0f` respectivamente. Por fim, a classe `DDMathEvaluator` é utilizada para calcular a expressão matemática e retornar um valor, que nesse caso é o valor de `z`.

2.3.2 Adicionando novas funções

A biblioteca `DDMathParser` permite adicionar novas funções para realizar a validação e cálculo das expressões matemáticas. Detalhes desse processo são mostrados na Figura 6.

Figura 6 – Cria e adiciona nova função

```

1 DDMathFunction function = ^ DDExpression* (NSArray *args, NSDictionary *variables,
2                                         DDMathEvaluator *evaluator, NSError **error) {
3     NSNumber * n = [[args objectAtIndex:0] evaluateWithSubstitutions:variables
4                         evaluator:evaluator error:error];
5     NSNumber * result = [NSNumber numberWithDouble:[n doubleValue] * 42.0f];
6     return [DDExpression numberExpressionWithNumber:result];
7 }
8 [[DDMathEvaluator defaultMathEvaluator] registerFunction:function forName:@"multiplyBy42"];

```

Fonte: Github (2014a).

A Figura 6 demonstra um exemplo de adição de uma nova função chamada `multiplyBy42`. Esta função de exemplo tem por objetivo retornar um valor resultante da multiplicação do valor 42 pelo parâmetro numérico informado na função. Uma função será do tipo `DDMathFunction`, dessa forma é necessário passar um bloco com `DDExpression` contendo as operações necessárias e retornando um resultado numérico como esperado. Por fim, a classe `DDMathEvaluator` é utilizada para registrar essa função e receber um nome, que neste caso é `multiplyBy42` (GITHUB, 2014a).

2.3.3 Adicionando novos operadores

A biblioteca `DDMathParser` permite adicionar novos operadores para realizar a validação e cálculo das expressões matemáticas. Detalhes desse processo são mostrados na Figura 7.

Figura 7 – Cria e adiciona novo operador

```

1 [[DDMathEvaluator defaultMathEvaluator] registerFunction:^DDExpression *(NSArray *args,
2                                         NSDictionary *vars, DDMathEvaluator *eval, NSError *_autoreleasing *error) {
3     if (args.count != 2) {
4         *error = [NSError errorWithDomain:DDMathParserErrorDomain
5                     code:DDErrorCodeInvalidNumberOfArguments
6                     userInfo:@{NSLocalizedStringKey: @"muladd requires 2 arguments"}];
7         return nil;
8     }
9
10    DDExpression *first = [args objectAtIndex:0];
11    DDExpression *second = [args objectAtIndex:1];
12
13    DDExpression *multiply = [DDExpression functionExpressionWithFunction:DDOperatorMultiply
14                                arguments:@[first, second] error:error];
15    DDExpression *sum = [DDExpression functionExpressionWithFunction:@"sum"
16                                arguments:@[first, second, multiply] error:error];
17    return sum;
18
19 } forName:@"muladd"];

```

Fonte: Github (2014b).

A Figura 7 demonstra um exemplo de adição de um novo operador chamado `muladd`. Esta função tem por objetivo retornar um valor resultante da multiplicação de dois valores numéricos com a soma de cada um desses, utilizando os parâmetros passados na função. Neste caso é passado o bloco `DDExpression` com as operações necessárias e retorno numérico, registrando direto na classe `DDMathEvaluator` com um nome desejado para o operador. Dentro do bloco desse exemplo, se mais de dois parâmetros forem informados será gerado um erro, senão serão multiplicados os valores dos parâmetros e ao final é retornada a soma dessa multiplicação com a soma de cada um dos parâmetros (GITHUB, 2014b).

2.4 TRABALHOS CORRELATOS

Para a concepção deste trabalho, foi pesquisado o trabalho acadêmico de Krauss (2013) que iniciou o projeto VisEdu-MAT (2013), e outros aplicativos para iPad que permitem a visualização de funções matemáticas em 2D e 3D. A seguir serão apresentados o VisEdu-MAT 1.0 do Krauss (2013), Quick Graph (APPSTORE, 2013b) e o Grafikal De Ageio (APPSTORE, 2013a).

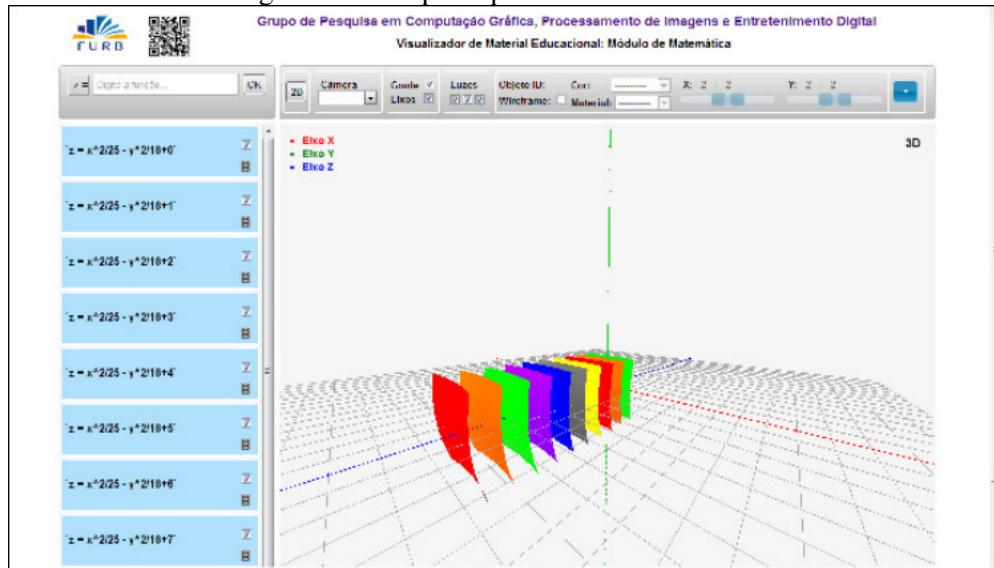
2.4.1 VisEdu-MAT 1.0

O trabalho desenvolvido por Krauss (2013) consiste no desenvolvimento de uma aplicação web para visualização de funções matemáticas explícitas usando *Web Graphics Library* (WebGL) e a biblioteca em JavaScript Three.js, que abstrai a complexidade da implementação WebGL.

Em resumo a aplicação tem como principal objetivo a criação, validação e exibição de uma função em um espaço 2D e 3D, permitindo que o usuário use nas funções as variáveis de domínio x , y , z para 3D ou x e y para 2D.

A Figura 8 mostra o aplicativo VisEdu-MAT 1.0, criado por Krauss (2013) com 10 objetos de cena posicionados no eixo z .

Figura 8 – Tela principal do VisEdu-MAT 1.0



Fonte: Krauss (2013, p. 52).

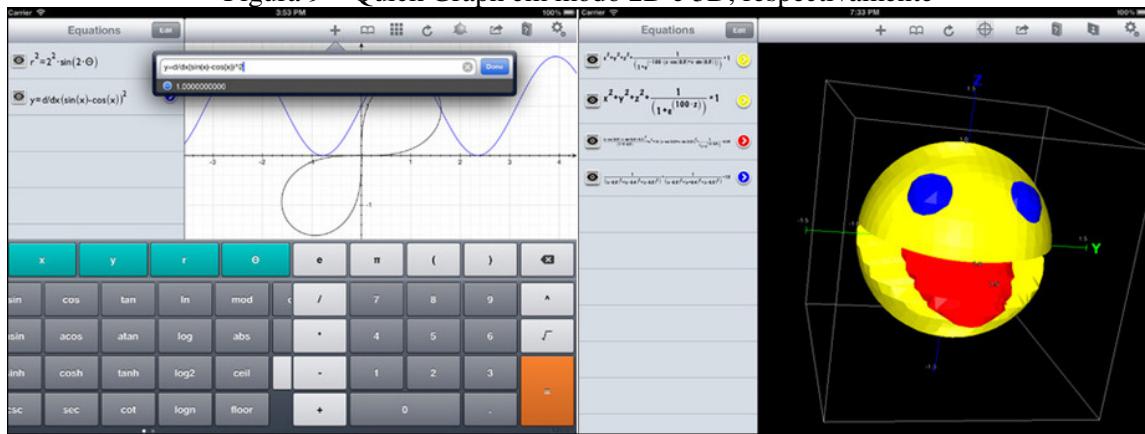
Como observado na Figura 8 é possível habilitar recursos de câmera, grade, eixos, luzes e *wireframe*. Também definir parâmetros isolados para cada objeto de cena como cor e material, além de mudar valores do domínio como ilustrado na Figura 8, previamente configurados em -2 e 2 pelos botões deslizantes X e Y.

O aplicativo faz validação sintática das funções digitadas. Permite a seleção do objeto da cena para alterar seus atributos como cor, material e valores do domínio. Interage com o usuário através das operações de ampliar, mover e girar aplicadas na câmera do visualizador gráfico. Possui uma biblioteca didática de exemplos de funções e salva (localmente) as funções que o usuário inseriu. Por fim, um recurso chamado *Cursor* pode ser habilitado, mostrando duas linhas auxiliares no visualizador gráfico indicando a posição x , y exata no plano, disponível apenas em modo 2D.

2.4.2 Quick Graph

O aplicativo Quick Graph é uma calculadora gráfica desenvolvida para iPad e iPhone, que permite a visualização de funções matemáticas simultaneamente em 2D e 3D, inseridas ou editadas em notação matemática. Na Figura 9 é ilustrada a visualização em modo 2D e 3D sucessivamente.

Figura 9 – Quick Graph em modo 2D e 3D, respectivamente



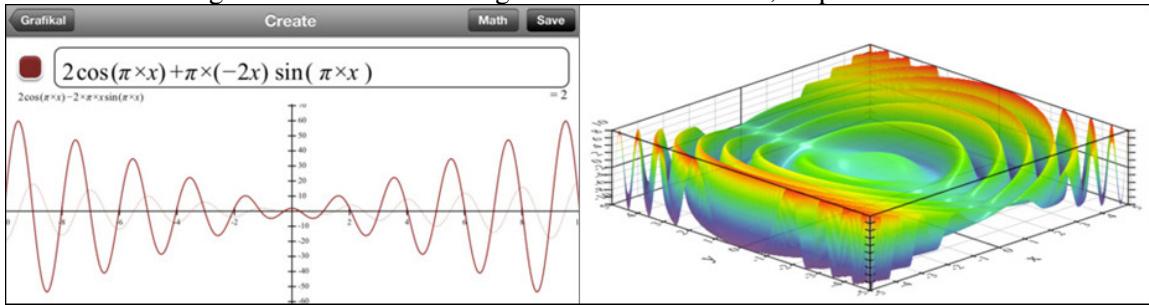
Fonte: AppStore (2013b).

Os gráficos podem ter sombreamento de cores personalizáveis, realçado pela iluminação, se estiver no modo 3D e habilitado, para melhorar a realidade em profundidade da superfície. Tirando proveito das telas multitoque é possível ampliar, mover, ou girar as imagens geradas com simples gestos com os dedos na tela, permitindo salvar os gráficos na biblioteca de imagens, copiar para a área de transferência ou compartilhar via email. Possui capacidade em exibir funções explícitas e implícitas, bem como inequações em 2D e 3D, em todos os sistemas de coordenadas padrão: cartesiano, polar, esférico e cilíndrico. A função *tracing* em gráficos 2D permite a visualização dos valores nas coordenadas x e y em um período da onda. Também disponibiliza um recurso de avaliação de expressões matemáticas que têm variáveis como parte da expressão e uma biblioteca para armazenar funções usadas (APPSTORE, 2013b).

2.4.3 Grafikal De Ageio

O aplicativo Grafikal De Ageio, desenvolvido para iPad e iPhone, é uma calculadora gráfica, capaz de executar cálculos em matemática simbólica avançada, expansão Taylor, função de passo Heaviside, retangulares e triangulares. Na Figura 10 é mostrado o resultado de uma expressão no formato 2D e 3D sucessivamente.

Figura 10 – Grafikal De Ageio em modo 2D e 3D, respectivamente



Fonte: AppStore (2013a).

Possui características como visualização de funções em 2D e 3D, recursos de constantes físicas, químicas e matemáticas, e função *tracing* em gráficos 2D. Com sua interface intuitiva, possui teclado customizável para expressões matemáticas, que traz maior aproximação ao teclado iOS padrão, contando com todas as constantes predefinidas do Sistema Internacional (SI) desde Pi (π), constante de Rydberg, até graus Celsius para Kelvin. Ao digitar uma parte da expressão, em tempo real, o gráfico atualiza-se para que o usuário possa verificar se a informação inserida é correta. O sistema permite traçar gráficos a partir dos tipos de arquivos *Comma-Separated Values* (CSV) e *Tab-Separated Values* (TSV), possuindo integração ao iCloud em iPad ou iPhone (APPSTORE, 2013a).

3 DESENVOLVIMENTO

Neste capítulo serão abordadas as principais etapas para o desenvolvimento do aplicativo VisEdu-MAT 2.0. Serão apresentados os requisitos funcionais e não funcionais, a especificação, a implementação e por fim os resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O aplicativo de visualização gráfica de funções matemáticas deverá:

- a) disponibilizar uma interface gráfica em 2D e 3D (Requisito Funcional – RF);
- b) permitir entrada da função matemática via digitação em campo de texto editável (RF);
- c) validar a função digitada (RF);
- d) manter salvas as funções matemáticas inseridas em uma lista visível e selecionável (RF);
- e) exibir o resultado gráfico em uma área de desenho (RF);
- f) alterar o contradomínio da função matemática (RF);
- g) alternar o modo de visualização de 2D para 3D, ou vice e versa (RF);
- h) permitir alterar propriedades das funções matemáticas (RF);
- i) permitir interação com a câmera (RF);
- j) permitir excluir uma função matemática da lista de funções (RF);
- k) disponibilizar um aplicativo para rodar em dispositivos iPad (Requisito Não Funcional – RNF);
- l) ser implementado utilizando o ambiente de desenvolvimento Xcode em linguagem de programação Objective-C (RNF);
- m) utilizar o *Software Development Kit* (SDK) iOS 8.1 no *Integrated Development Environment* (IDE) Xcode 6.1 para o desenvolvimento de aplicativos em sistemas iOS (RNF);
- n) utilizar a biblioteca gráfica OpenGL ES 2.0 para realizar a renderização das imagens em 2D e 3D (RNF);
- o) utilizar a biblioteca DDMathParser para realizar a validação e os cálculos da função matemática (RNF).

3.2 ESPECIFICAÇÃO

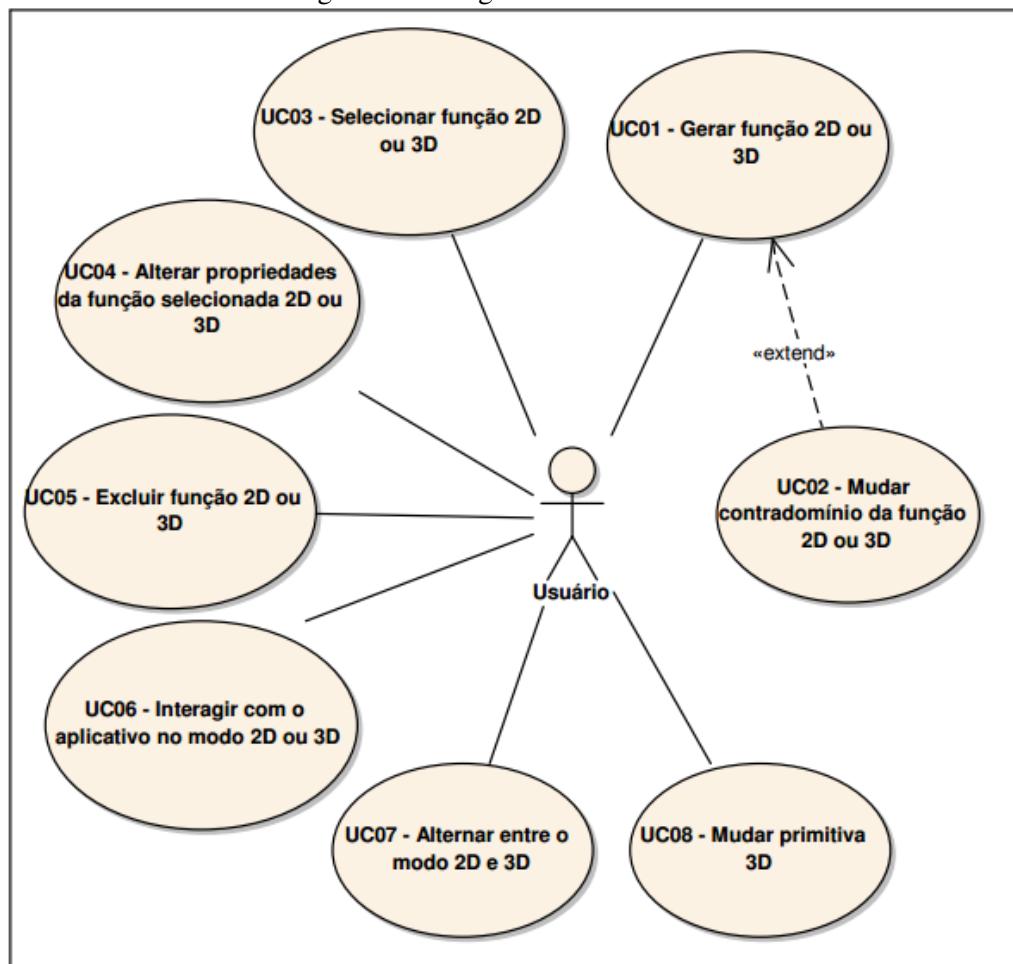
A especificação deste trabalho foi realizada utilizando a ferramenta Enterprise Architect, seguindo o modelo de orientação a objetos e diagramas *Unified Modeling Language* (UML).

A seguir são apresentados os diagramas de caso de uso, diagramas de classe e diagrama de sequência.

3.2.1 Diagrama de casos de uso

Nesta seção são apresentados os casos de uso que descrevem as funcionalidades do VisEdu-MAT 2.0. De acordo com os requisitos funcionais foram definidos oito casos de uso, que permitem o ator executar operações no aplicativo. Como o aplicativo necessita de apenas um ator este foi definido como **Usuário**, conforme ilustrado na Figura 11.

Figura 11 – Diagrama de casos de uso



3.2.1.1 Gerar função 2D ou 3D

Esse caso de uso descreve como o Usuário deve proceder para gerar uma nova função que irá ser mostrada no aplicativo. Detalhes deste caso de uso são mostrados no Quadro 1.

Quadro 1 – Descrição do caso de uso UC01

| UC01 - Gerar função 2D ou 3D | |
|------------------------------|---|
| Requisitos atendidos | RF01, RF02, RF03, RF04, RF05 |
| Pré-condição | Nenhuma. |
| Cenário principal | <ol style="list-style-type: none"> 1. No modo 2D ou 3D, o Usuário digita a função matemática; 2. O Usuário clica no botão +; 3. O aplicativo valida a função matemática; 4. O aplicativo cria o objeto de cena correspondente a função matemática; 5. O aplicativo grava o objeto de cena no banco de dados local; 6. O aplicativo adiciona a função matemática na lista; 7. O aplicativo mostra o desenho resultante da função matemática na área de desenho. |
| Exceção | Na etapa 3, caso a função matemática não for suportada ou estiver incorreta, será mostrado uma caixa de diálogo com o erro. |
| Pós-condição | O Usuário visualiza o objeto de cena gerado pela função matemática renderizado na área de desenho. |

3.2.1.2 Mudar o contradomínio da função 2D ou 3D

Esse caso de uso descreve como o Usuário deve proceder para mudar o contradomínio da função 2D ou 3D. Detalhes deste caso de uso são mostrados no Quadro 2.

Quadro 2 – Descrição do caso de uso UC02

| UC02 - Mudar o contradomínio da função 2D ou 3D | |
|---|---|
| Requisitos atendidos | RF06 |
| Pré-condição | Nenhuma. |
| Cenário principal | <ol style="list-style-type: none"> 1. No modo 3D, o Usuário clica no botão contradominio; 2. O aplicativo muda a imagem do botão, correspondente aos possíveis contradomínios: x, y e z; 3. O aplicativo muda os sliders do domínio para y e z, z e x ou x e y se o contradomínio for x, y ou z respectivamente. |
| Cenário alternativo | <ol style="list-style-type: none"> 1. No modo 2D, o Usuário clica no botão contradominio; 2. O aplicativo muda a imagem do botão, correspondente aos possíveis contradomínios x e y; 3. O aplicativo muda os parâmetros de domínio para y ou x se o contradomínio for x ou y respectivamente. |
| Pós-condição | O objeto de cena resultante será desenhado no eixo do plano cartesiano correspondente ao contradomínio. |

3.2.1.3 Selecionar função 2D ou 3D

Esse caso de uso descreve como o Usuário deve proceder para selecionar a função no modo 2D ou 3D. Detalhes deste caso de uso são mostrados no Quadro 3.

Quadro 3 – Descrição do caso de uso UC03

| UC03 – Selecionar função 2D ou 3D | |
|-----------------------------------|---|
| Requisitos atendidos | RF04 |
| Pré-condição | UC1 |
| Cenário principal | <ol style="list-style-type: none"> 1. No modo 3D, o Usuário clica sobre a função presente na lista de funções; 2. O aplicativo muda a cor no item da lista de funções, correspondente a função selecionada; 3. O aplicativo renderiza na área de desenho com a <i>Bounding Box</i> do objeto de cena selecionado; 4. O aplicativo atualiza os sliders de domínio de acordo com as propriedades do objeto de cena selecionado. |
| Cenário alternativo | <ol style="list-style-type: none"> 1. No modo 2D, o Usuário clica sobre a função presente na lista de funções; 2. O aplicativo muda a cor no item da lista de funções correspondente a função selecionada; 3. O aplicativo intensifica a cor do objeto de cena selecionado; 4. O aplicativo atualiza os sliders do domínio e slider do tracing de acordo com as propriedades do objeto de cena selecionado; 5. O aplicativo exibe componente slider do tracing para realizar o tracing da função 2D. |
| Pós-condição | O Usuário poderá mudar as propriedades do objeto de cena selecionado. |

3.2.1.4 Alterar propriedades da função selecionada 2D ou 3D

Esse caso de uso descreve como o Usuário deve proceder para alterar propriedades da função selecionada 2D ou 3D. Detalhes deste caso de uso são mostrados no Quadro 4.

Quadro 4 – Descrição do caso de uso UC04

| UC04 – Alterar propriedades da função selecionada 2D ou 3D | |
|--|--|
| Requisitos atendidos | RF08 |
| Pré-condição | UC3 |
| Cenário principal | <ol style="list-style-type: none"> No modo 2D ou 3D, o Usuário clica sobre o checkbox visibilidade localizado no lado esquerdo da lista de funções; O aplicativo alterna entre visível e invisível o objeto de cena renderizado na área de desenho; O Usuário clica sobre o botão cor posicionado no lado direito da lista de funções; O aplicativo muda cor desse botão e o objeto de cena é renderizado com a cor correspondente na área de desenho; O Usuário interage com componentes sliders do domínio, aumentando e diminuindo a faixa de seus valores; O aplicativo recalcula e renderiza o objeto de cena com as novas faixas de valores dos sliders. |
| Cenário alternativo | <ol style="list-style-type: none"> No modo 2D, o Usuário clica sobre o botão Ponto; O aplicativo alterna a visibilidade do desenho do ponto renderizado na área de desenho; O Usuário interage com o slider de tracing, posicionado ao lado direito do botão Ponto, aumentando e diminuindo a faixa de seus valores; O aplicativo atualiza os valores do campo x e campo y, posicionado ao lado direito do slider de tracing e renderiza o ponto exibido na área de desenho. |
| Pós-condição | O Usuário visualiza as alterações na área de desenho. |

3.2.1.5 Excluir função 2D ou 3D

Esse caso de uso descreve como o Usuário deve proceder para excluir uma função no modo 2D ou 3D. Detalhes deste caso de uso são mostrados no Quadro 5.

Quadro 5 – Descrição do caso de uso UC05

| UC05 – Excluir função 2D ou 3D | |
|--------------------------------|--|
| Requisitos atendidos | RF10 |
| Pré-condição | UC1 |
| Cenário principal | <ol style="list-style-type: none"> No modo 2D ou 3D, o Usuário pressiona um item na lista de funções e arrasta para a esquerda, isso mostrará o botão Delete; O Usuário deverá clicar sobre o botão Delete exibido; O aplicativo irá excluir a função da lista de funções e da área de desenho. |
| Pós-condição | O objeto de cena deixa de existir na lista de funções e na área de desenho. |

3.2.1.6 Interagir com o aplicativo no modo 2D ou 3D

Esse caso de uso descreve como o Usuário deve proceder para interagir com o aplicativo no modo 2D ou 3D. Detalhes deste caso de uso são mostrados no Quadro 6.

Quadro 6 – Descrição do caso de uso UC06

| UC06 – Interagir com o aplicativo no modo 2D ou 3D | |
|--|---|
| Requisitos atendidos | RF09 |
| Pré-condição | Nenhuma. |
| Cenário principal | <ol style="list-style-type: none"> 1. No modo 3D, o Usuário toca com um dedo e arrasta na área de desenho da tela; 2. O aplicativo translada a câmera na direção a qual o dedo arrasta na tela; 3. O Usuário toca com dois dedos na área de desenho da tela; 4. Com os dedos na tela o Usuário distancia ou aproxima os dedos; 5. O aplicativo aumenta ou diminui o <i>zoom</i> da câmera respectivamente; 6. O Usuário toca com dois dedos na área de desenho da tela; 7. Com os dedos na tela o Usuário arrasta para uma direção; 8. O aplicativo rotaciona a câmera no mesmo sentido em que os dedos arrastaram na tela; 9. O Usuário clica sobre o botão centralizar; 10. O aplicativo retorna os valores iniciais de <i>zoom</i>, rotação e translação da câmera, deixando-a no centro da área de desenho. |
| Cenário alternativo | <ol style="list-style-type: none"> 1. No modo 2D, o Usuário toca com dois dedos na a área de desenho da tela; 2. Com os dedos na tela o Usuário distancia ou aproxima os dedos; 3. O aplicativo aumenta ou diminui o <i>zoom</i> respectivamente; 4. O Usuário clica sobre o botão centralizar; 5. O aplicativo retorna os valores iniciais de <i>zoom</i>, rotação e translação da câmera, deixando-a no centro da área de desenho. |
| Pós-condição | O Usuário visualiza na área de desenho o resultado da interação dos gestos e cliques no aplicativo. |

3.2.1.7 Alternar entre o modo 2D e 3D

Esse caso de uso descreve como o Usuário deve proceder para alternar entre o modo 2D e 3D. Detalhes deste caso de uso são mostrados no Quadro 7.

Quadro 7 – Descrição do caso de uso UC07

| UC07 – Alternar entre o modo 2D e 3D | |
|--------------------------------------|---|
| Requisitos atendidos | RF07 |
| Pré-condição | Nenhuma. |
| Cenário principal | <ol style="list-style-type: none"> 1. No modo 3D, o Usuário clica no botão modo; 2. O aplicativo carrega os objetos de cena 2D do banco de dados local; 3. O aplicativo atualiza a lista de funções; 4. O aplicativo recalcula e renderiza os objetos de cena na área de desenho; 5. O aplicativo exibe a tela no modo 2D. |
| Cenário alternativo | <ol style="list-style-type: none"> 1. No modo 2D, o Usuário clica no botão modo; 2. O aplicativo carrega os objetos de cena 3D do banco de dados local; 3. O aplicativo atualiza a lista de funções; 4. O aplicativo recalcula e renderiza os objetos de cena na área de desenho; 5. O aplicativo exibe a tela no modo 3D. |
| Pós-condição | O Usuário interage com as funções matemáticas e interface de acordo com o modo selecionado. |

3.2.1.8 Mudar primitiva 3D

Esse caso de uso descreve como o Usuário deve proceder para mudar a primitiva no modo 3D. Detalhes deste caso de uso são mostrados no Quadro 8.

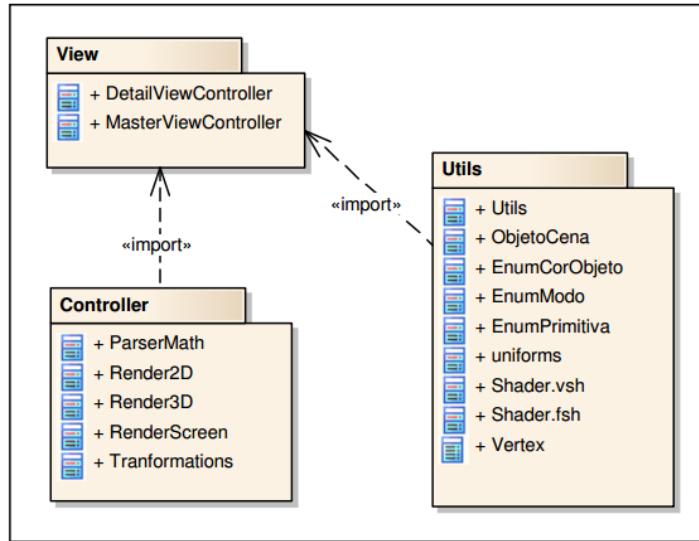
Quadro 8 – Descrição do caso de uso UC08

| UC08 – Mudar primitiva 3D | |
|---------------------------|--|
| Requisitos atendidos | Nenhum. |
| Pré-condição | UC01 |
| Cenário principal | <ol style="list-style-type: none"> 1. No modo 3D, o Usuário clica no botão primitiva; 2. O aplicativo alterna entre as primitivas <i>mesh</i>, <i>wireframe</i> ou <i>points</i>, que correspondem aos tipos de desenho malha de pontos, aramado ou pontos respectivamente. 3. O aplicativo atualiza todos os objetos de cena na área de desenho de acordo com a primitiva escolhida. |
| Pós-condição | O Usuário visualiza os tipos de primitivas aplicados na área de desenho. |

3.2.2 Diagrama de classes

Nesta seção são detalhadas as classes que fazem parte do VisEdu-MAT 2.0. Na Figura 12 são mostrados os principais pacotes do aplicativo, como estão relacionados e quais as classes que os compõem.

Figura 12 – Diagrama de pacotes

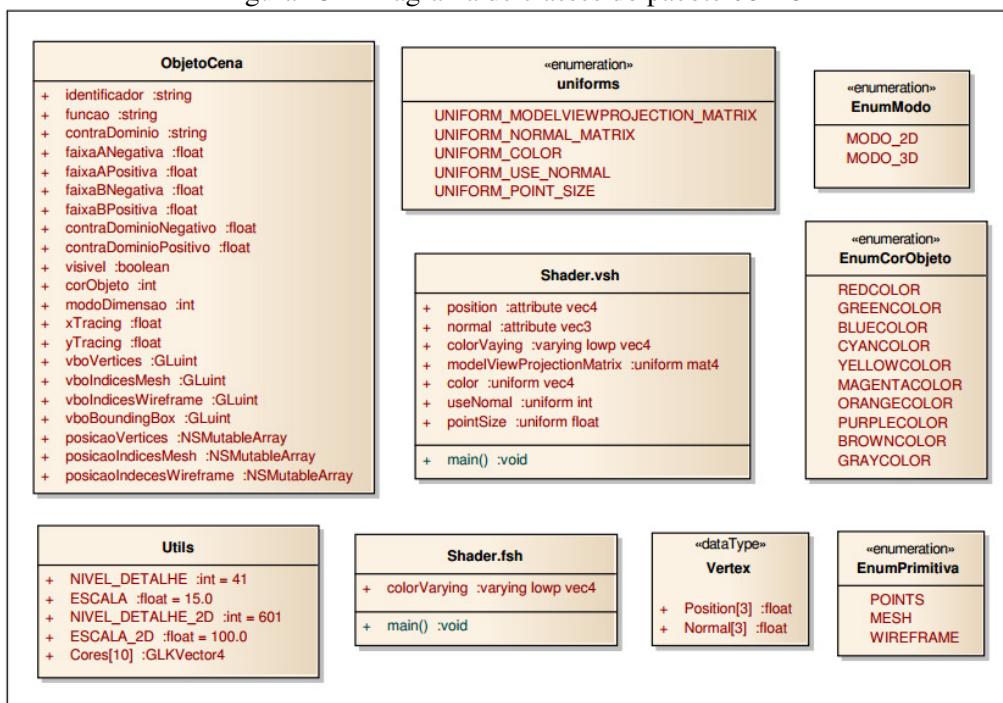


Na Figura 12 o pacote `Utils` possui as classes auxiliares ao aplicativo, o pacote `View` contém as classes responsáveis pelo controle da tela e persistência dos dados, e por último, o pacote `Controller` possui as classes especialistas na criação dos objetos gráficos gerados a partir da função matemática.

3.2.2.1 Pacote `Utils`

O pacote `Utils` possui as classes que são frequentemente utilizadas nos pacotes `View` e `Controller`. A Figura 13 apresenta o diagrama de classes desse pacote, assim como seus principais métodos e atributos.

Figura 13 – Diagrama de classes do pacote `Utils`



A classe `ObjetoCena` é a classe de modelo para representar um objeto de cena gerado a partir de uma função matemática. Ao instanciar um objeto da classe `ObjetoCena`, a função matemática é atribuída a variável `funcao` e seu contradomínio é atribuído a variável `contraDominio`. Para variar as faixas de valores do domínio as variáveis `faixaANegativa`, `faixaAPositiva`, `faixaBNegativa` e `faixaBPositiva` são utilizadas. Com essas faixas e as variáveis `contraDominioNegativo` e `contraDominioPositivo` é possível realizar o desenho do *Bounding Box* para mostrar que um objeto de cena foi selecionado no modo 3D. A visibilidade do objeto de cena é definido pela variável `visivel` e a cor do objeto pela variável `corObjeto`. A variável `modoDimensao` recebe o modo 2D ou 3D correspondente ao modo do objeto de cena. As variáveis `xTracing` e `yTracing` recebem as coordenadas de um ponto a ser renderizado no modo 2D se o *tracing* estiver ativado. Os *Vertex Buffer Object* (VBO) `vboVertices`, `vboIndicesMesh`, `vboIndicesWireframe` e `vboBoundingBox` são utilizados para guardar a referência do *buffer* presente na *Graphics Processing Unit* (GPU). Os arrays `posicaoVertices`, `posicaoIndicesMesh` e `posicaoIndicesWireframe` guardam valores correspondentes a nuvem de pontos gerados a partir da função matemática. Por fim, a variável `identificador` possui um identificador único em cada instância de `ObjetoCena` para controles do banco de dados do iPad.

As classes `Utils` contêm valores constantes referentes aos valores do universo. Para o modo 3D a constante `NIVEL_DETALHE` possui o valor inteiro de 41. Esse valor interfere no nível de detalhe de cada objeto de cena gerado, que nesse caso é 41 vezes 41 igual a 1.681 pontos que serão gerados. Essa é a nuvem de pontos que será atribuída ao array `posicaoVertices` da classe `ObjetoCena`. A constante `ESCALA` do modo 3D define a quantidade de linhas e colunas exibidas na grade visível ao usuário. No modo 2D a constante `NIVEL_DETALHE_2D` irá gerar 601 pontos que serão atribuídos ao array `posicaoVertices` da classe `ObjetoCena`. Esses valores foram definidos por questão de desempenho e usabilidade do aplicativo.

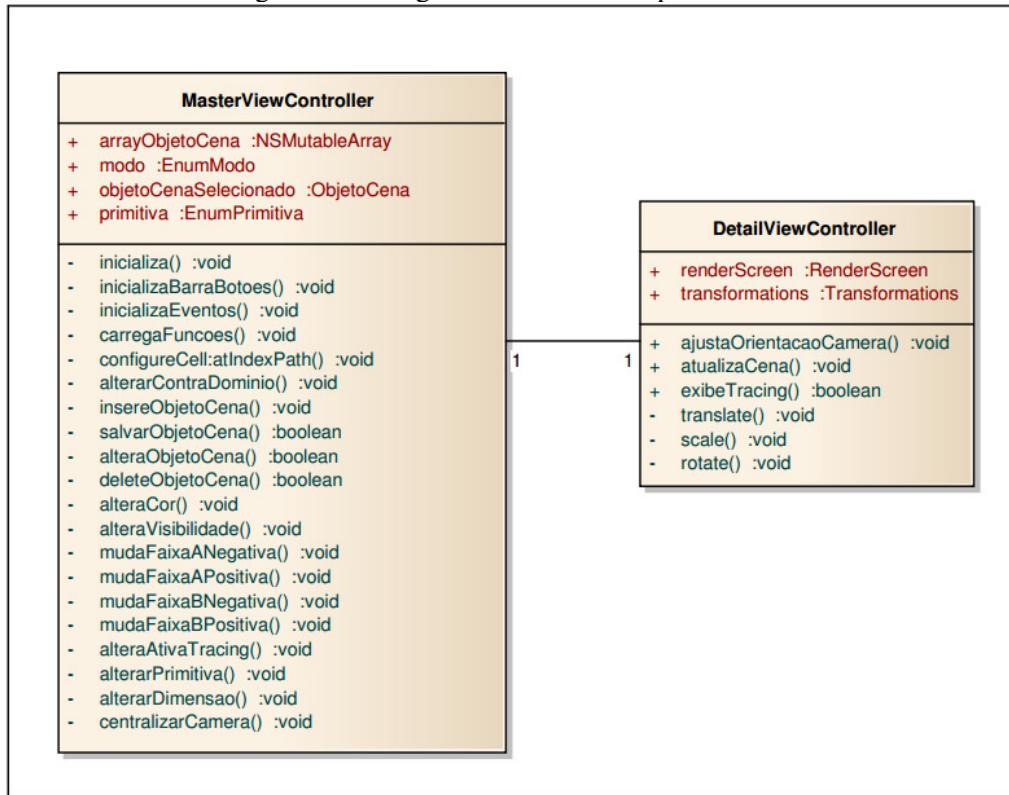
Os programas que rodam na GPU estão presentes neste pacote. O programa do *vertex shader* está escrito no `Shader.vsh`, enquanto o programa do *fragment shader* está escrito no `Shader.fsh`.

Os enumeradores `uniforms`, `EnumModo`, `EnumCorObjeto`, `EnumPrimitiva` e a estrutura `Vertex` são entidades auxiliares e que desta forma não necessitam de uma abordagem descritiva.

3.2.2.2 Pacote View

O pacote `View` constitui a camada de interação do usuário com o aplicativo VisEdu-MAT 2.0. Neste pacote estão as classes que realizam o controle dos componentes de tela e delegam funções para classes de outros pacotes. Os principais métodos e atributos das classes que compõe esse pacote são mostrados na Figura 14.

Figura 14 – Diagrama de classes do pacote `View`



A classe `MasterViewController` é responsável por gerenciar os objetos de cena gerados a partir de uma função matemática inserida pelo usuário. Além disso, é de sua responsabilidade apresentar e controlar esses objetos de cena na lista de funções matemáticas visível ao usuário.

O método `insereObjetoCena` é chamado toda vez que uma função matemática é inserida pelo usuário através do campo editável e confirmada no botão `+`. Neste momento é instanciado um objeto da classe `ObjetoCena` presente no pacote `Utils`. Em seguida é utilizado a classe `ParserMath` do pacote `Controller`, para validar a função matemática e gerar a nuvem de pontos que compõe o desenho. Na sequência, se não houver erro na validação da função matemática, a instância da classe `ObjetoCena` é persistido no banco de dados interno do iPad através do método `salvarObjetoCena` e adicionado ao `arrayObjetoCena`. Por fim é chamado o método `atualizaCena` da classe `DetailViewController`.

A classe `DetailViewController` é responsável por exibir a área de desenho visível ao usuário. Utilizando a instância `transformations` da classe `Transformations` presente no pacote `Controller`, é tratado os eventos de toques e gestos na tela. Estes são traduzidos em eventos de translação, *zoom* e rotação da câmera do OpenGL ES via métodos `translate`, `scale` e `rotate` respectivamente.

O método `atualizaCena` da classe `DetailViewController` delega a função de renderização dos objetos de cena através da classe `RenderScreen` do pacote `Controller`. Desta forma a instância `renderScreen` da classe `RenderScreen` será responsável por gerenciar e criar os desenhos no OpenGL ES.

Na sequência, o método `ajustaOrientacaoCamera` da classe `DetailViewController` é chamado pelo método `centralizarCamera` da classe `MasterViewController`, assim que o usuário clica no botão `centralizar`. Este método inicializa a variável `transformations` de forma que a câmera do OpenGL ES fique centralizada tanto no modo 2D quanto 3D.

Por fim, quando o aplicativo está no modo 2D e o usuário clica no botão `Ponto` ou interage com o slider do tracing, o método `desenhaPontoTracing` da classe `RenderScreen` é chamado via método `exibeTracing` da classe `DetailViewController`. O objetivo do método `exibeTracing` é exibir o desenho de um ponto em uma coordenada `x` e `y` em cima de um desenho de uma função matemática em 2D.

3.2.2.3 Pacote Controller

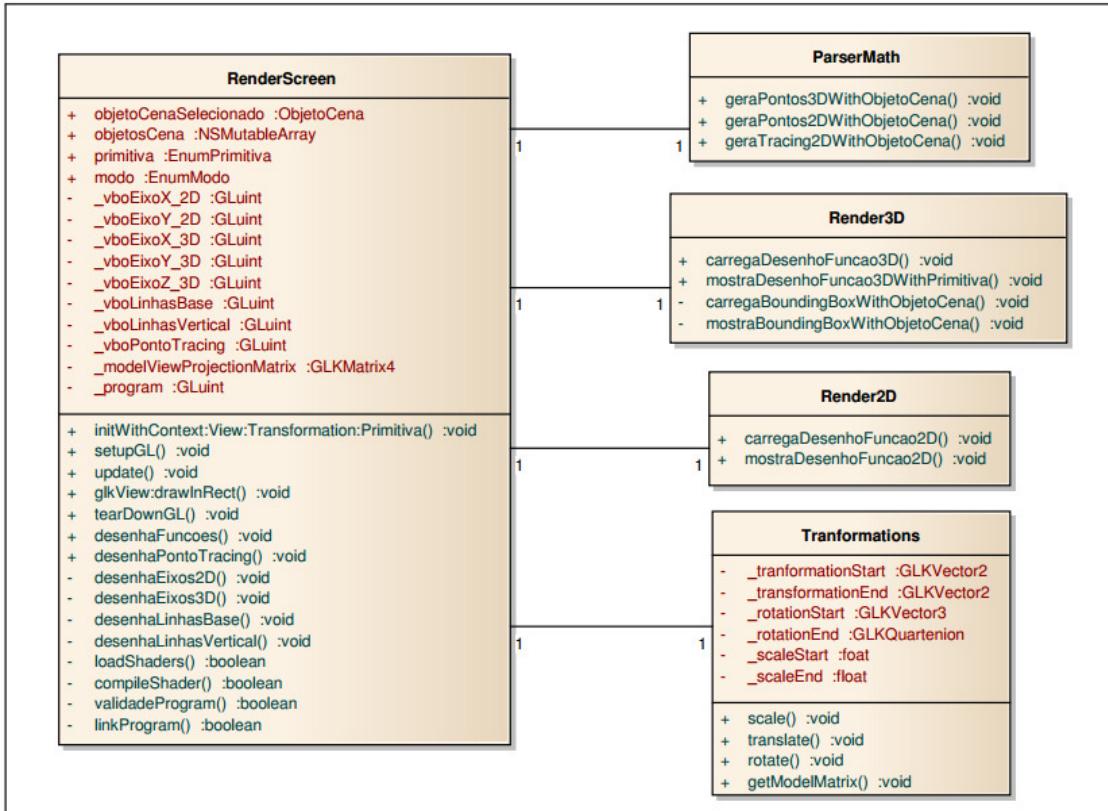
O pacote `Controller` constitui as classes especialistas que são responsáveis por gerar e manipular os objetos de cena e exibir via OpenGL ES. A Figura 15 apresenta o diagrama de classes desse pacote, assim como seus principais métodos e atributos.

A classe `Transformations` possui uma matriz de transformação que sofre mudanças através dos métodos `scale`, `translate` e `rotate`. Desta forma, essa classe é responsável por definir o ponto em que a câmera do OpenGL ES está posicionada e consequentemente a visão do usuário sobre a cena localizada na área de desenho.

A classe `ParserMath` tem por objetivo gerar a nuvem de pontos ao realizar a validação e o cálculo da função matemática utilizando a biblioteca `DDMathParser`. Se a função matemática não for válida, os métodos desta classe retornam uma *string* com a mensagem de erro, senão é iniciado o processo para gerar a nuvem de pontos a qual é associado ao objeto de cena. É desta forma que os métodos `geraPontos3DWithObjetoCena` e `geraPontos2DWithObjetoCena` desta classe funcionam. O método

`geraTracing2DWithObjetoCena` funciona de maneira semelhante porém ao invés de gerar uma nuvem de pontos é gerado apenas um ponto.

Figura 15 – Diagrama de classes do pacote Controller



As classes `Render3D` e `Render2D` são classes especialistas no preparo do desenho dos objetos de cena que são renderizados através da classe `RenderScreen`. Os métodos `carregaDesenhoFuncao3D` da classe `Render3D` e `carregaDesenhoFuncao2D` da classe `Render2D`, utilizam a nuvem de pontos gerados na classe `ParserMath` para gerar o VBO gerenciado pela GPU. Os métodos `mostraDesenhoFuncao3DWithPrimitiva` da classe `Render3D` e `mostraDesenhoFuncao2D` da classe `Render2D`, utilizam o VBO associado ao objeto de cena para mostrar o desenho na área de desenho.

A classe `RenderScreen` é responsável por criar os VBOs e desenhar na área de desenho, além de receber os eventos do OpenGL ES que são realizados através da classe `DetailViewController`.

De início o método `setupGL` é chamado para criar a grade de linhas visível ao usuário a partir dos métodos `desenhaLinhasBase` e `desenhaLinhasVertical`, inicializando o `_vboLinhasBase` e `_vboLinhasVertical` respectivamente. Na sequência, os VBOs `_vboEixoX2D`, `_vboEixoY2D`, `_vboEixoX3D`, `_vboEixoY3D` e `_vboEixoZ3D` que são responsáveis pela exibição do Sistema de Referência Universal (SRU), são inicializados

através dos métodos `desenhaEixos2D` e `desenhaEixos3D` para o modo 2D e 3D respectivamente. Ao final deste método, é realizado uma chamada ao método `desenhaFuncoes`, que utiliza a classe `Render2D` e `Render3D` para criar os VBOs dos objetos de cena presentes na variável `objetosCena` desta classe.

O método `update` é chamado pela classe `DetailViewController` do pacote `View`, a qual provém de um evento constantemente gerado pelo OpenGL ES. Neste método, a matriz de transformação da câmera do OpenGL ES `_modelViewProjectionMatrix` é atualizada com os valores obtidos a partir do método `getModelViewMatrix`, da classe `Transformations`.

Além do método `update`, o método `glkView:drawInRect` também é chamado pela classe `DetailViewController`, a partir de um evento constantemente gerado pelo OpenGL ES. Sua função é realizar o controle e a renderização dos desenhos exibidos na área de desenho, utilizando os VBOs inicializados anteriormente.

Por fim, os métodos `loadShader`, `compileShader`, `validadeProgram` e `linkProgram` são utilizados para carregar, compilar, validar e ligar os programas do *vertex shader* e *fragment shader* antes de serem utilizados na classe `RenderScreen`.

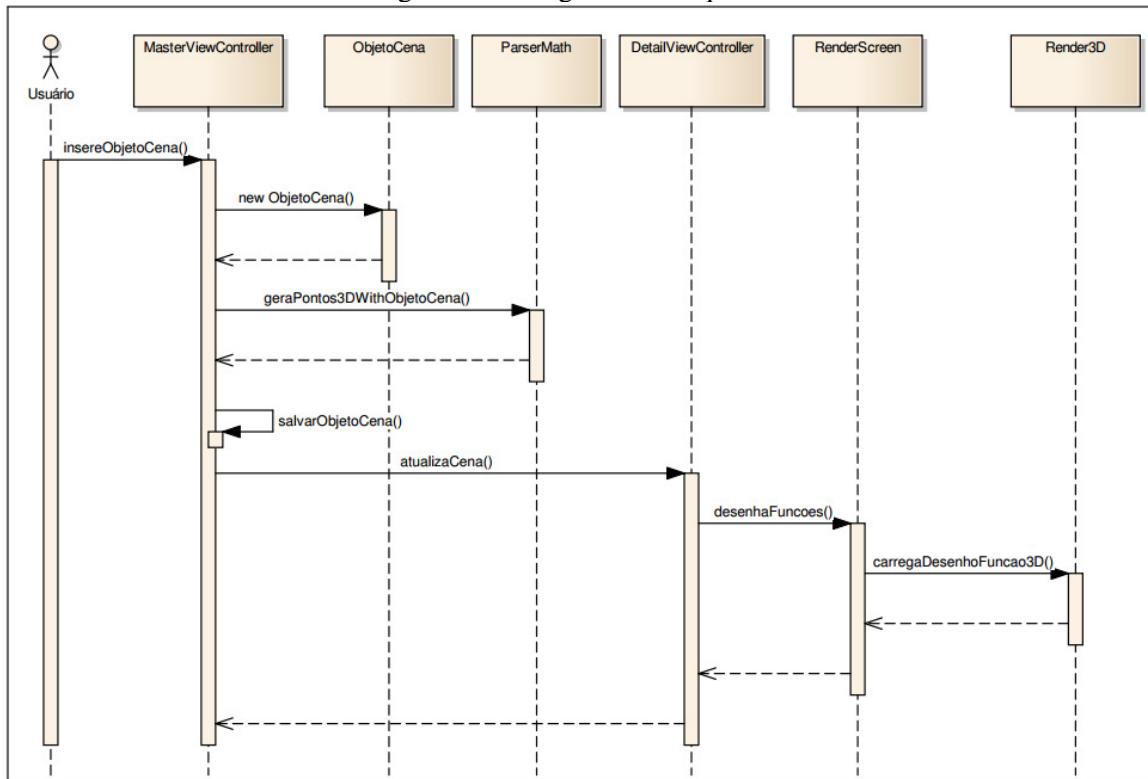
3.2.3 Diagrama de sequência

Esta seção apresenta o diagrama de sequência do VisEdu-MAT 2.0, que mostra o processo de criação de um objeto de cena e seu desenho a partir de uma função matemática. A Figura 16 ilustra esse diagrama em detalhes.

Ao iniciar o `Usuário` insere uma função matemática no campo editável e clica no botão `+`. Após esse clique uma chamada ao método `insereObjetoCena` da classe `MaterViewController` irá instanciar um objeto de cena a partir da classe `ObjetoCena`, passando os valores da função matemática, contradomínio selecionado e modo de operação 2D ou 3D. As faixas de domínio recebem o valor 7.0 em `float`, para exibir o objeto de cena com um tamanho apropriado para visualização do resultado gráfico. Por fim, a visibilidade do objeto de cena é definida como visível e sua cor inicialmente será vermelha.

Na sequência, retornando ao método `insereObjetoCena` da classe `MasterViewController`, o método `geraPontos3DWithObjetoCena` da classe `ParserMath`, será chamado com o objeto de cena criado anteriormente. Esse método será responsável por validar a função matemática e calcular cada ponto que compõe o desenho final do objeto de cena.

Figura 16 – Diagrama de sequência



Se não houver problemas na validação da função realizada na classe `ParserMath`, o método `salvarObjetoCena` desta classe será chamado para gravar o objeto de cena no banco de dados interno do iPad.

Ainda no método `insereObjetoCena`, será realizado uma chamada ao método `atualizaCena` da classe `DetailViewController`, a qual imediatamente será chamado o método `desenhaFuncoes` da classe `RenderScreen`. Essa classe é responsável por todo o processo de desenho exibido na área de desenho que utiliza o OpenGL ES. Desta forma a parte especialista de criação do desenho é realizada ao chamar o método `carregaDesenhoFuncao3D` da classe `Render3D`.

3.3 IMPLEMENTAÇÃO

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

3.3.1 Técnicas e ferramentas utilizadas

Para a concepção deste aplicativo foi utilizada a linguagem de programação Objective-C no ambiente de programação Xcode 6.1 (6A1052d) com o *Software Development Kit* (SDK) 8.1 do iOS. Também foram utilizadas a biblioteca DDMathParser na versão de *commit*

224 (hospedado no repositório *on-line* GitHub) e a biblioteca gráfica OpenGL ES na versão 2.0.

Para o controle de versões e armazenamento seguro dos códigos fontes, foi utilizada a ferramenta Git disponível no Xcode, a qual realiza *commit* local e remoto. Para o *commit* remoto, foi criado um repositório on-line hospedado pelo BitBucket.

O dispositivo utilizado para o desenvolvimento e testes foi o iPad 2 Mini (A1489), versão de software iOS 7.1.2 (11D257) e processador quad core 1.2 GHz. O computador usado em todo o desenvolvimento do aplicativo foi um MacBook Pro (A1278), com sistema operacional Mavericks versão 10.9.4 (13E28), processador Intel Core i5 de 2.5 GHz, memória RAM de 4GB e placa de vídeo Intel HD Graphics 4000.

3.3.2 O visualizador de material educacional (VisEdu-MAT 2.0)

Esta seção descreve a implementação do VisEdu-MAT 2.0, demonstrando as principais etapas na geração da nuvem de pontos, preparo do desenho do objeto gráfico e sua renderização na área de desenho.

3.3.2.1 Gerar nuvem de pontos da função matemática

Nesta seção são mostrados como é realizado o processo de geração da nuvem de pontos e primitivas de desenho, a partir de uma função matemática.

O método `geraPontos3DWithObjetoCena` da classe `ParserMath`, recebe como parâmetro um objeto de cena e tem por objetivo gerar a nuvem de pontos que compõe o desenho, a partir da função previamente digitada pelo usuário.

Nesse método a biblioteca `DDMathParser`, responsável por realizar a validação e os cálculos das funções matemáticas, é configurada para receber uma função matemática do objeto de cena com as possíveis variáveis de domínio x y e z , conforme mostrado na Figura 17.

Figura 17 – Detalhes do método geraPontos3DWithObjetoCena

```

193 - (NSString *)geraPontos3DWithObjetoCena:(ObjetoCena *)objeto
194 {
195     NSString *expression = objeto.funcao;
196     NSError *error = nil;
197
198     DDMathStringTokenizer *tokenizer = [[DDMathStringTokenizer alloc] initWithString:expression
199                                         operatorSet:nil error:&error];
200
201     if (error != nil) {
202         return [NSString stringWithFormat:@"%@", [[error localizedDescription] UTF8String]];
203     }
204
205     NSArray *tokens = [tokenizer allObjects];
206     tokens = [tokens valueForKey:@"token"];
207
208     NSMutableString *analize = [[NSMutableString alloc] initWithString:@""];
209
210     for (NSInteger i = 0; i < [tokens count]; i++) {
211         if ([tokens[i] isEqual:@"x"] || [tokens[i] isEqual:@"y"] || [tokens[i] isEqual:@"z"]) {
212             [analize appendString:@"$"];
213             [analize appendString:tokens[i]];
214             i = i + 2;
215         } else{
216             [analize appendString:tokens[i]];
217         }
218     }

```

Na Figura 17 a variável `tokenizer` (198) do tipo `DDMathStringTokenizer` possui a lista de `tokens` gerada a partir da função matemática do objeto de cena, a qual é atribuída a variável `tokens` (204). Para um `token` ser reconhecido como uma variável na biblioteca `DDMathParser`, o mesmo deve preceder o carácter \$ (211). Desta forma a variável `tokens` é iterada para este propósito buscando pelos `tokens` x, y e z (210). Ao final de cada iteração os `tokens` são concatenados na variável `string analise` que será utilizada mais a frente.

Por convenção, os eixos do plano cartesiano formados pelo domínio da função matemática foram definidos como sendo eixo A e eixo B. Os valores das faixas negativa `faixaANegativa` e `faixaBNegativa` do objeto de cena, seguem essa convenção. Assim como os valores das faixas positiva `faixaAPositiva` e `faixaBPositiva` do objeto de cena, que também seguem essa mesma convenção. Essa definição permitiu abstrair a idéia de que o domínio pode ser formado por y e z, z e x ou x e y se o contradomínio for x, y ou z respectivamente.

A Figura 18 mostra o uso dessa convenção para iterar entre os possíveis valores das variáveis de domínio representados por `eixoA` e `eixoB`, que iniciam pelo valor negativo da constante do tipo inteiro `NIVEL_DETALHE` dividido por 2, passa pelo valor 0 e finaliza no valor positivo da constante dividido por 2. Dentro desse escopo de iteração, o objetivo é encontrar os valores para as variáveis do tipo `float` `posicaoA` e `posicaoB`, que seguem a mesma convenção de `eixoA` e `eixoB`. A variável `posicaoA` assim como `posicaoB` varia de zero até o valor limite da faixa a qual pertence, interferido pelos valores atuais do `eixoA` ou `eixoB` da iteração. De acordo com o contradomínio, as variáveis `posicaoA` (258) e `posicaoB` (259) ganham o sentido real, pois são atribuídas ao seus devidos eixos de domínios.

Figura 18 – Continuação do método geraPontos3DWithObjetoCena

```

240 for (int eixoA = -NIVEL_DETALHE/2; eixoA <= NIVEL_DETALHE/2; eixoA++ )
241 {
242     for (int eixoB = -NIVEL_DETALHE/2; eixoB <= NIVEL_DETALHE/2; eixoB++ )
243     {
244
245         if(eixoA < 0)
246             posicaoA = ((float) eixoA / (NIVEL_DETALHE/2)) * ([objeto.faixaANegativa
247                                         floatValue] * -1.0f);
248         else
249             posicaoA = ((float) eixoA / (NIVEL_DETALHE/2)) * [objeto.faixaAPositiva
250                                         floatValue];
251
252         if(eixoB < 0)
253             posicaoB = ((float) eixoB / (NIVEL_DETALHE/2)) * ([objeto.faixaBNegativa
254                                         floatValue] * -1.0f);
255         else
256             posicaoB = ((float) eixoB / (NIVEL_DETALHE/2)) * [objeto.faixaBPositiva
257                                         floatValue];
258
259         if ([objeto.contraDominio isEqualToString:@"x"]) {
260
261             y = posicaoA;
262             z = posicaoB;
263
264             dictionary = [NSDictionary dictionaryWithObjects:
265                         [NSArray arrayWithObjects:[NSNumber numberWithFloat: y], [NSNumber
266                                         numberWithInt: z], nil]
267                                         forKeys:[NSArray arrayWithObjects:@"y",
268                                         @"z", nil]];
269
270             DDMathEvaluator *eval = [DDMathEvaluator defaultMathEvaluator];
271             x = [[eval evaluateString:analize withSubstitutions:dictionary error:&error]
272                                         floatValue];
273             valorContraDominio = x;
274         }
275         else if ([objeto.contraDominio isEqualToString:@"y"]) {

```

Conforme observado na Figura 18, se o contradomínio se refere ao eixo *x* do plano cartesiano, nota-se que a *posicaoA* é atribuída a variável *y* correspondente ao eixo de *y* e a *posicaoB* é atribuída a variável *z* correspondente ao eixo *z*. Nesse mesmo escopo é inicializado a variável *dictionary* (261) que recebe os valores do tipo *float* de *y* e *z* para as chaves do tipo *string* de *y* e *z*. A variável *dictionary* é utilizada para substituir as variáveis *y* e *z* por valores reais, assim que a chamada ao método *evaluateString* (266) da *DDMathEvaluator* for realizado com a variável *analise* citada anteriormente. O resultado desse método é o valor do contradomínio, que neste caso é atribuído a variável *x* (266), correspondente ao eixo *x*.

Seguindo essa convenção, se o contradomínio for *y*, então a *posicaoA* será *x* e a *posicaoB* será *z*. Entretanto se o contradomínio for *z*, então *posicaoA* será *x* e a *posicaoB* será *y*. Por fim, os valores de contradomínio *y* e *z* são obtidos de forma semelhante ao contradomínio *x* citado anteriormente.

Com os valores de *x*, *y* e *z*, obtém-se a posição de um vértice no espaço 3D. Desta forma esses valores são adicionados ao *array posicaoVertices* do objeto de cena, passado por parâmetro neste método. Também são adicionados os mesmos valores de *x*, *y* e *z* na potência de dois no *array posicaoVertices*, a fim de ser utilizado para realçar os cálculos de luz realizado dentro do *vertex shader* do OpenGL ES.

Ainda no momento dessa iteração, é buscado os valores limites do contradomínio que são atribuídos ao `contraDominioNegativo` e `contraDominioPositivo` no objeto de cena, passado por parâmetro nesse método. Com os valores de `contraDominioNegativo`, `contraDominioPositivo` e os outros limites `faixaANegativa`, `faixaAPositiva`, `faixaBNegativa` e `faixaBPositiva`, é possível realizar o *Bounding Box* do objeto selecionado.

Como a constante `NIVEL_DETALHE` foi definida com valor inteiro de 41, ao final da iteração, foram adicionados no *array* `posicaoVertices` do objeto de cena um total de 1.681 vértices, decorrentes da iteração dos dois laços `for`.

Ao final do método `geraPontos3DWithObjetoCena` são gerados os índices a serem utilizados caso o usuário selecione a primitiva `mesh` ou `wireframe`. A Figura 19 demonstra esse processo.

Figura 19 – Final do método `geraPontos3DWithObjetoCena`

```

331 // GL_TRIANGLES
332 for (int a = 0; a < NIVEL_DETALHE-1; a++ )
333 {
334     for (int b = 0; b < NIVEL_DETALHE-1; b++ )
335     {
336         [objeto.posicaoIndicesMesh addObject: [NSNumber numberWithInt:( a + 1 ) *
337             NIVEL_DETALHE) + b]];
338         [objeto.posicaoIndicesMesh addObject: [NSNumber numberWithInt:(a * NIVEL_DETALHE) + b
339             + 1]];
340         [objeto.posicaoIndicesMesh addObject: [NSNumber numberWithInt:(a * NIVEL_DETALHE) + b
341             + 1]];
342         [objeto.posicaoIndicesMesh addObject: [NSNumber numberWithInt:( a + 1 ) *
343             NIVEL_DETALHE) + b]];
344         [objeto.posicaoIndicesMesh addObject: [NSNumber numberWithInt:( a + 1 ) *
345             NIVEL_DETALHE) + b + 1]];
346         [objeto.posicaoIndicesMesh addObject: [NSNumber numberWithInt:(a * NIVEL_DETALHE) + b
347             + 1]];
348     }
349 }
350
351 // GL_LINES
352 for (int a = 0; a < NIVEL_DETALHE; a++ )
353 {
354     for (int b = 0; b < NIVEL_DETALHE-1; b++ )
355     {
356         [objeto.posicaoIndicesWireframe addObject: [NSNumber numberWithInt:( (a *
357             NIVEL_DETALHE) + b)]];
358         [objeto.posicaoIndicesWireframe addObject: [NSNumber numberWithInt:( (a *
359             NIVEL_DETALHE) + b + 1)]];
360         [objeto.posicaoIndicesWireframe addObject: [NSNumber numberWithInt:( (b *
361             NIVEL_DETALHE) + a)]];
362         [objeto.posicaoIndicesWireframe addObject: [NSNumber numberWithInt:( (b *
363             NIVEL_DETALHE) + a + NIVEL_DETALHE)]]];
364     }
365 }
366 }
```

Na Figura 19 inicialmente é gerado o *array* `posicaoIndicesMesh` (332), responsável por criar os índices necessários da primitiva `mesh`. O algoritmo se baseia na idéia de uma matriz com dimensão definida pela constante `NIVEL_DETALHE`, correspondente a localização dos vértices gerados anteriormente no *array* `posicaoVertices` do objeto de cena. Para cada quatro vértices vizinhos, são gerados os índices que compõe os dois triângulos. É necessário a criação desses triângulos, para posteriormente o OpenGL ES realizar o desenho da superfície aplicada ao objeto de cena. Ao final de toda a iteração, são geradas 3.200 primitivas de

triângulos (no caso, 40 linhas x 40 colunas x 2), formados por três índices que são adicionados no *array* posicaoIndicesMesh.

Por fim, como mostrado na Figura 19, é gerado o *array* posicacaoIndicesWireframe (346), responsável por criar os índices necessários da primitiva wireframe. O algoritmo também se baseia na idéia de uma matriz com dimensão definida pela constante NIVEL_DETALHE, correspondente a localização dos vértices gerados anteriormente no *array* posicaoVertices do objeto de cena. Para cada dois pontos vizinhos, tanto na linha como na coluna dessa matriz, são gerados os índices que compõe uma reta. É necessária a criação dessas retas, para posteriormente o OpenGL ES realizar o desenho das linhas aplicadas ao objeto de cena. Ao final de toda a iteração, são geradas 3.280 primitivas de retas (no caso, 41 linhas x 40 colunas x 2), formados por dois índices que são adicionados no *array* posicaoIndicesWireframe.

3.3.2.2 Carregar nuvem de pontos na memória da GPU

Nesta seção são mostrados os passos no modo 3D, para carregar a nuvem de pontos gerada na seção anterior, para dentro da memória administrada pela GPU.

O método `carregaDesenhoFuncao3D` da classe `Render3D`, tem por objetivo criar os VBOs utilizado dentro do *vertex shader* do OpenGL ES, a partir dos *arrays* posicaoVertices, posicaoIndicesMesh e posicaoIndicesWireframe de cada objeto de cena.

A Figura 20 mostra o momento a qual o *array* posicaoVertices do objeto cena `objetoCena` é convertido para a o *array* `fPosicaoVertices` (32) do tipo `GLfloat` e a criação do VBO `vboVertices` (36), que é atribuído ao objeto de cena `objetoCena` (37). O método `glBindBuffer` (40) do OpenGL ES, é chamado passando como parâmetro o tipo do *buffer* `GL_ARRAY_BUFFER` e o `vboVertices` do `objetoCena`. Dessa forma, as próximas chamadas aos métodos do OpenGL ES fazem referência diretamente a esse *buffer*. Na sequência o método `glBufferData` (41) do OpenGL ES, recebe como parâmetros o tipo do *buffer* `GL_ARRAY_BUFFER`, o tamanho em *bytes* do *buffer*, o *array* `fPosicaoVertices` e a forma de uso `GL_STATIC_DRAW`. Ao final da execução do método `glBufferData`, os vértices do objeto de cena são copiados através o OpenGL ES para a memória administrada pela GPU e podem ser acessados a partir da referência `vboVertices` do tipo inteiro, presente no objeto de cena.

Figura 20 – Detalhes do método carregaDesenhoFuncao3D

```

13 - (void)carregaDesenhoFuncao3D
14 {
15     if ([self.objetosCena != nil && [self.objetosCena count] > 0]) {
16
17         ObjetoCena *objetoCena;
18
19         for (int i = 0; i < [self.objetosCena count]; i++) {
20
21             objetoCena = [[self.objetosCena objectAtIndex:i];
22
23             if ([objetoCena.posicaoVertices != nil && [objetoCena.posicaoVertices count] > 0]) {
24
25                 NSUInteger tamanhPosicaoVertices = [objetoCena.posicaoVertices count];
26
27                 if (tamanhPosicaoVertices > 0) {
28
29                     GLfloat fPosicaoVertices[tamanhPosicaoVertices];
30
31                     for (int posVtx = 0; posVtx < tamanhPosicaoVertices; posVtx++)
32                         fPosicaoVertices[posVtx] = [[objetoCena.posicaoVertices objectAtIndex:posVtx]
33                                         floatValue];
34
35                     if (objetoCena.vboVertices == 0) {
36                         GLuint vboVertices;
37                         glGenBuffers(1, &vboVertices);
38                         objetoCena.vboVertices = vboVertices;
39
40                         glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, objetoCena.vboVertices);
41                         glBufferData(GL_ELEMENT_ARRAY_BUFFER, tamanhPosicaoVertices * sizeof(GLfloat),
42                                     fPosicaoVertices, GL_STATIC_DRAW);
43
44                     if (objetoCena.posicaoIndicesMesh != nil && [objetoCena.posicaoIndicesMesh
45                                         count] > 0) {

```

Na sequência tanto o *array* de índices *posicaoIndicesMesh*, quanto *posicaoIndicesWireframe* do objeto de cena *objetoCena*, são armazenados na memória administrada pela GPU, com o tipo de *buffer* *GL_ELEMENT_ARRAY_BUFFER*, referenciados como *vboIndicesMesh* e *vboIndicesWireframe* presente no *objetoCena*.

Para cada objeto de cena encontrado é chamado o método presente nessa classe, o *carregaBoundingBoxWithObjetoCena* que recebe como parâmetro um objeto de cena. O objetivo desse método é criar um *array* do tipo *GLfloat* a partir de valores do objeto de cena *objeto* *faixaANegativa*, *faixaAPositiva*, *faixaBNegativa*, *faixaBPositiva*, *contraDominioNegativo* e *contraDominioPositivo* para armazenar no *vboBoundingBox* do objeto de cena. Esse método utiliza a mesma convenção mostrada na seção anterior para identificar os valores de *x*, *y* e *z* correspondentes aos eixos do plano cartesiano.

Voltando ao método *carregaDesenhoFuncao3D*, os *arrays* *posicaoVertices*, *posicaoIndicesMesh* e *posicaoIndicesWireframe* do objeto de cena são esvaziados para deixar de ocupar espaço na memória principal, uma vez que estão armazenados na memória administrada pela GPU.

3.3.2.3 Renderizar nuvem pontos na área de desenho

Nesta seção são mostrados como a nuvem de pontos que estão presentes na memória administrada pela GPU são renderizadas na área de desenho do aplicativo.

O processo de renderização depende do entendimento do programa do *vertex shader* `Shader.vsh`, escrito em *OpenGL Shading Language* (GLSL) e executado diretamente na GPU conforme exibido na Figura 21.

Figura 21 – Detalhes do programa `Shader.vsh`

```

9  attribute vec4 position;
10 attribute vec3 normal;
11
12 varying lowp vec4 colorVarying;
13
14 uniform mat4 modelViewProjectionMatrix;
15 uniform vec4 color;
16 uniform int useNormal;
17 uniform float pointSize;
18
19 void main()
{
20
21   if (useNormal > 0) {
22     vec3 eyeNormal = normalize(normal);
23     vec3 lightPosition = vec3(1.0, 1.0, 1.0);
24
25     float nDotVP = max(0.0, dot(eyeNormal, normalize(lightPosition)));
26
27     colorVarying = color * nDotVP;
28   }
29   else
30   {
31     colorVarying = color;
32   }
33
34   gl_Position = modelViewProjectionMatrix * position;
35   gl_PointSize = pointSize;
36 }
```

Para cada vértice o programa da Figura 21 é executado a partir do método `main` (19). O `attribute position` (9) recebe a coordenada espacial do vértice e o `attribute normal` (10) recebe valores baseados na coordenada espacial do vértice para cálculos de luz. O `varying colorVarying` (12) carrega o resultado dos cálculos de luz para o programa *fragment shader*, que por sua vez atribui o valor de `colorVarying` diretamente ao `gl_FragColor` do *fragment shader* responsável por definir a cor do *pixel*. Os `uniform modelViewProjectionMatrix` (14), `color` (15), `useNormal` (16) e `pointSize` (17) são valores constantes atribuídos a todos os vértices. O `uniform modelViewProjectionMatrix` é uma matriz transformação, seus valores são decorrentes das operações de rotação, translação e *zoom* da câmera e são obtidos a partir do método `getModelViewMatrix` da classe `Transformations`. O `uniform color` é um vetor e possui o valor do tipo `float` da cor correspondente ao padrão *Red Green Blue Alpha* (RGBA). O `uniform useNormal` do tipo inteiro com valor maior que 0 permite executar os cálculos de luz para o `color` atribuído ao `colorVarying`, se o valor for menor ou igual a 0 então somente o `color` é atribuído ao `colorVarying`. Neste caso qualquer objeto gráfico renderizado pelo OpenGL ES utiliza o mesmo programa do *vertex shader*, dessa forma a definição do `uniform useNormal` com valor maior que 0 é aplicado apenas aos objetos gráfico provenientes ao desenho das funções matemáticas. Na sequência o `uniform pointSize` define o tamanho do vértice em *pixels* atribuído diretamente ao atributo `gl_PointSize` (35) do *vertex shader*.

Quando `useNormal` (21) for maior que zero, indica que o cálculo de luz deverá ser aplicado ao resultado da cor do vértice. Dessa forma é normalizada o `attribute normal` do vértice e atribuído a variável `eyeNormal` (22). Criado uma nova variável chamada `lightPosition` (23), que indica que a luz provém da origem dos eixos `x`, `y` e `z`. Calculado o produto vetorial entre `eyeNormal` e a normalização de `lightPosition`, a qual será atribuída a nova variável `nDotVP` (25) do tipo `float`. Na sequência será realizado uma multiplicação do vetor `color` com o valor do `nDotVP` e atribuído ao `colorVarying` (27). Como resultado esperado a intensidade da cor do vértice irá variar decorrente da incidência de luz em sua direção.

Por fim, na Figura 21 o atributo `gl_Position` (34) do *vertex shader* multiplica a posição do vértice definido pelo `attribute position` com a matriz de transformação definido pelo `uniform modelViewProjectionMatrix`.

O método `mostraDesenhoFuncao3DWithPrimitiva` da classe `Render3D` é chamada pelo método `glkView:drawInRect` da classe `RenderScreen`, constantemente invocado pelo OpenGL ES para renderizar todos os objetos gráficos exibidos.

Na Figura 22 para cada objeto de cena com o atributo visível igual a `true` (107) é recuperado as referências de memória `vboVertices` (109), `vboIndicesMesh` (110) e `vboIndicesWireframe` (111), além da cor do objeto obtido a partir da enumeração `Cores` com o atributo `corObjeto` (113). Posteriormente é atribuído o valor 1 ao `UNIFORM_USE_NORMAL`, correspondente ao `uniform useNormal` do *vertex shader*, para permitir o cálculo de luz. Após isso é atribuído o valor do atributo `corObjeto` ao `UNIFORM_COLOR`, correspondente ao `uniform color` do *vertex shader*. Na sequência é chamado o `glBindBuffer` (118) do OpenGL ES, com o tipo do *buffer* `GL_ARRAY_BUFFER` e o `vboVertices`. Desta forma as próximas operações realizadas estão referenciadas ao `vboVertices`. O método do OpenGL ES `glEnableVertexAttribArray` (120) com o parâmetro `GLKVertexAttribPosition` habilita o uso do `attribute position` do *vertex shader*. O próximo método a ser chamado é o `glVertexAttribPointer` (121), que especifica a forma como o OpenGL ES deve interpretar as informações contidas no *buffer* referenciado por `vboVertices`, assim como o tamanho de cada vértice e a quantidade em `bytes` que estes ocupam em memória. De maneira semelhante, o `GLKVertexAttribNormal` (123) permite o uso do `attribute normal` do *vertex shader* e a interpretação do *buffer* `vboVertices`.

Figura 22 – Detalhes do método mostraDesenhoFuncao3DWithPrimitiva

```

99 - (void)mostraDesenhoFuncao3DWithPrimitiva:(EnumPrimitiva)primitiva
100 {
101     if ([self.objetosCena != nil && [self.objetosCena count] > 0]) {
102         ObjetoCena *objetoCena;
103
104         for (int i = 0; i < [self.objetosCena count]; i++) {
105             objetoCena = [self.objetosCena objectAtIndex:i];
106
107             if ([objetoCena.visivel boolValue]) {
108
109                 GLuint vboVertices = objetoCena.vboVertices;
110                 GLuint vboIndicesMesh = objetoCena.vboIndicesMesh;
111                 GLuint vboIndicesWireframe = objetoCena.vboIndicesWireframe;
112
113                 GLKVector4 corObjeto = Cores[[objetoCena.corObjeto integerValue]];
114
115                 glUniform1i(uniforms[UNIFORM_USE_NORMAL], 1);
116                 glUniform4fv(uniforms[UNIFORM_COLOR], 1, corObjeto.v);
117
118                 glBindBuffer(GL_ARRAY_BUFFER, vboVertices);
119
120                 glEnableVertexAttribArray(GLKVertexAttribAttribPosition);
121                 glVertexAttribPointer(GLKVertexAttribAttribPosition, 3, GL_FLOAT, GL_FALSE, sizeof(Vertice), (const GLvoid *) offsetof(Vertice, Position));
122
123                 glEnableVertexAttribArray(GLKVertexAttribAttribNormal);
124                 glVertexAttribPointer(GLKVertexAttribAttribNormal, 3, GL_FLOAT, GL_FALSE, sizeof(Vertice), (const GLvoid *) offsetof(Vertice, Normal));
125
126                 if (primitiva == POINTS)
127                 {
128                     glUniform1f(uniforms[UNIFORM_POINT_SIZE], 3.0f);
129                     NSUInteger tamanhoVtx = NIVEL_DETALHE * NIVEL_DETALHE;
130                     glDrawArrays(GL_POINTS, 0, (int)tamanhoVtx);
131                 }
132                 else if (primitiva == MESH)
133                 {
134                     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboIndicesMesh);
135                     NSUInteger tamanhoIdx = (NIVEL_DETALHE-1) * (NIVEL_DETALHE-1) * 6;
136                     glDrawElements(GL_TRIANGLES, (int)tamanhoIdx, GL_UNSIGNED_INT, 0);
137                 }
138                 else
139                 {
140                     glLineWidth(1.0f);
141                     glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vboIndicesWireframe);
142                     NSUInteger tamanhoIdx = NIVEL_DETALHE * (NIVEL_DETALHE-1) * 4;
143                     glDrawElements(GL_LINES, (int)tamanhoIdx, GL_UNSIGNED_INT, 0);
144                 }
145
146                 if ([self.objetoCenaSelecionado != nil])
147                 {
148                     if ([[self.objetoCenaSelecionado identificador] isEqualToString:[objetoCena
149                         identificador]])
150                     {
151                         [self mostraBoundingBoxWithObjetoCena:objetoCena];
152                     }
153                 }
154             }
155         }
156     }
157 }

```

Ainda na Figura 22 se a primitiva passada por parâmetro desse método for `POINTS` (126), então é passado o valor 3 para o `UNIFORM_POINT_SIZE`, correspondente ao `uniformPointSize` do *vertex shader*, para deixar o ponto maior e mais visível. Na sequência é chamado o método `glDrawArrays` do OpenGL ES, com a primitiva `GL_POINTS` e com a quantidade de vértices, dado pela multiplicação da constante `NIVEL_DETALHE` por ela mesma. Ao final da execução do método `glDrawArrays`, o OpenGL ES pode começar a desenhar utilizando o programa do *vertex shader* e *fragment shader*. Contudo se a primitiva for `MESH` (132), então o método `glBindBuffer` do OpenGL ES é chamado com o tipo de *buffer* `GL_ELEMENT_ARRAY_BUFFER` e o `vboIndicesMesh`. Neste momento o OpenGL ES tem o conhecimento dos vértices e dos índices, pois faz referência ao *buffer* `vboVertices` e o `vboIndicesMesh`. A partir da chamada ao método `glDrawElements` do OpenGL ES, com os parâmetros `GL_TRIANGLES` e tamanho do índice `tamanhoIdx`, o OpenGL ES pode começar a desenhar os vértices utilizando o programa do *vertex shader* e *fragment shader*. Por fim, se a

primitiva `for WIREFRAME (138)`, o processo será semelhante a primitiva `MESH`, referenciando o tipo de `buffer GL_ELEMENT_ARRAY_BUFFER` com o `vboIndicesWireframe` e chamado `glDrawElements` com o parâmetro `GL_LINES`, para que o OpenGL ES possa começar a desenhar.

Ao final do método `mostraDesenhoFuncao3DWithPrimitiva` conforme a Figura 22, se o objeto de cena `objetoCena` for igual ao objeto de cena selecionado pelo usuário `objetoCenaSelecionado` (147), então é chamado o método dessa classe `mostraBoundingBoxWithObjetoCena` com o parâmetro `objetoCena`. Este método desenha as linhas que compõe o *Bounding Box* e utiliza o `vboBoundingBox` presente no objeto de cena, passado como parâmetro.

3.3.3 Operacionalidade da implementação

As seções a seguir mostram em detalhes a operacionalidade da implementação do aplicativo VisEdu-MAT 2.0. Inicialmente é apresentado o aplicativo em modo 3D e na sequência em modo 2D. O Apêndice A mostra o manual do usuário que contém mais informações sobre a operacionalidade da implementação.

3.3.3.1 VisEdu-MAT 2.0 modo 3D

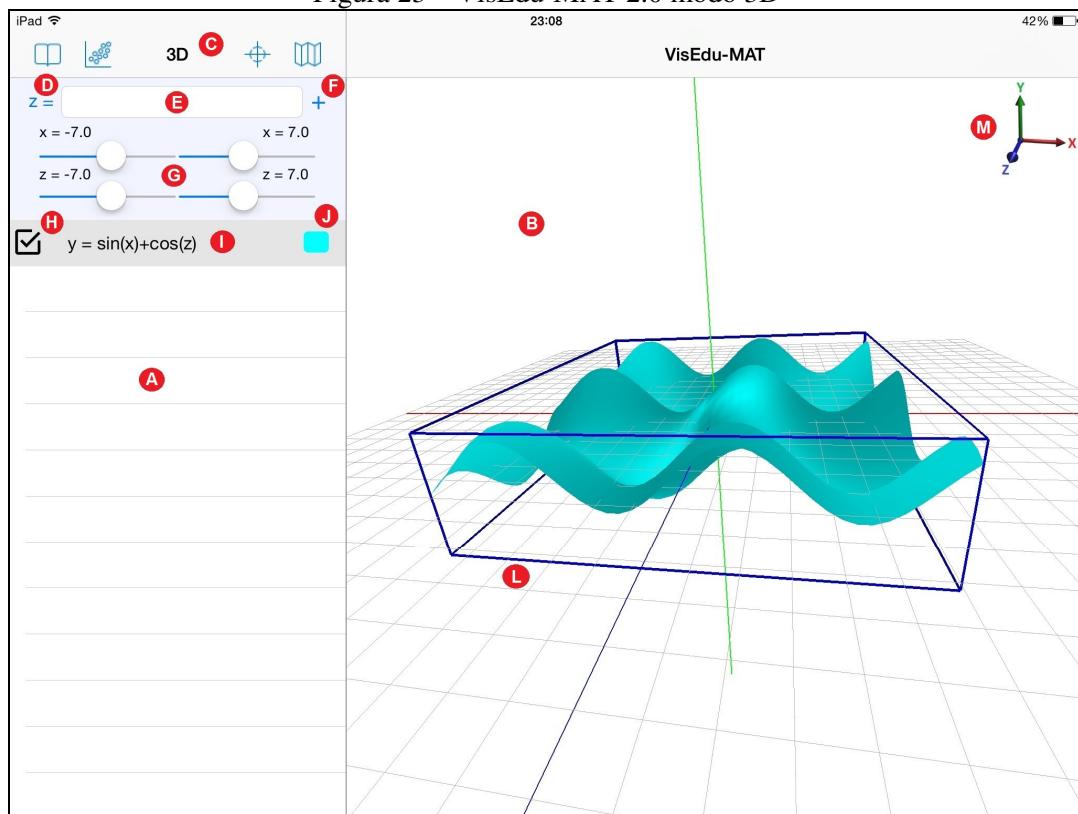
Ao iniciar o aplicativo VisEdu-MAT 2.0 o usuário visualiza a tela principal no modo 3D como mostrado na Figura 23. Esta tela possui a lista de funções matemáticas (A), a área de desenho (B) e a barra de botões (C). Observando a barra de botões da esquerda para a direita são identificados os botões: exibir biblioteca, mudar primitiva gráfica, centralizar câmera e mudar modo.

Para inserir uma nova função matemática no aplicativo, o usuário pode alterar o contradomínio clicando sobre o botão `contradominio` (D). A escolha do contradomínio altera o eixo a qual será exibido o desenho da função matemática. O VisEdu-MAT 2.0 utiliza o sistema de orientação mão direita, conforme exibido na imagem de orientação dos eixos (M). Desta forma se o `contradominio` escolhido for `x`, o desenho da função matemática será exibido no eixo horizontal. Se o `contradominio` for `y`, o desenho será exibido no eixo vertical. Por fim se o `contradominio` for `z`, o desenho será exibido no eixo frontal.

Após a escolha do contradomínio, o usuário insere o texto da função matemática no campo editável (E) e pressiona o botão + (F) para adicionar a função matemática no aplicativo. Se a função matemática inserida conter erros, o aplicativo irá exibir uma caixa de diálogo com a descrição do erro encontrado, senão esta será adicionada a lista de funções

matemáticas (A). Para cada função matemática (I) da lista de funções matemáticas é possível alterar sua visibilidade, clicando sobre o checkbox visibilidade (H) e sua cor, clicando sobre o botão cor (J).

Figura 23 – VisEdu-MAT 2.0 modo 3D



Por padrão, todas as funções matemáticas inseridas pelo usuário serão geradas no intervalo de valores de -7 a 7 para cada variável dos sliders do domínio (G). Além disso, os sliders do domínio serão y e z, z e x ou x e y se o contradomínio for x, y ou z respectivamente.

A Figura 23 mostra a seleção de uma função matemática (I) realizada pelo usuário. Imediatamente são desenhadas linhas nos limites do desenho da função matemática que formam a estrutura de uma caixa. Essas linhas são conhecidas como Bounding Box (L), que permitem a visualização da função matemática selecionada na área de desenho (B).

Para exibir a biblioteca didática de funções matemáticas, o usuário deverá clicar no primeiro botão à esquerda na barra de botões (C). Ao clicar neste botão, uma tela será exibida para que o usuário possa visualizar alguns exemplos didáticos.

Para mudar a primitiva gráfica dos desenhos das funções matemáticas, o usuário deve clicar no segundo botão da esquerda pra direita na barra de botões (C). Este botão permite o usuário alternar entre as primitivas mesh, wireframe ou points, que correspondem aos tipos de desenho malha de pontos, aramado ou pontos respectivamente.

A interação do usuário sobre área de desenho (B) ocorre através de gestos e toques na tela. Como no VisEdu-MAT 2.0 não permite a mudança do posicionamento dos objetos de cena, todas as operações de translação, *zoom* e rotação são realizadas diretamente na câmera da área de desenho, dando a impressão que são os objetos que são alterados. Se o usuário tocar com um dedo na área de desenho e arrastar, o aplicativo translada a câmera nesta direção. Se o usuário tocar com dois dedos na área de desenho e distanciar ou aproximar seus dedos, o aplicativo aumenta ou diminui o *zoom* da câmera respectivamente. Se o usuário tocar com dois dedos na área de desenho e arrastar, o aplicativo rotaciona a câmera nesta direção.

Para centralizar a câmera de modo que os eixos sejam exibidos de acordo com o sistema de orientação mão direita (M), o usuário deve clicar no terceiro botão da esquerda pra direita na barra de botões (C).

Por fim, para alternar entre o modo 3D e o modo 2D, o usuário deve clicar no último botão da esquerda pra direita da barra de botões (C).

3.3.3.2 VisEdu-MAT 2.0 modo 2D

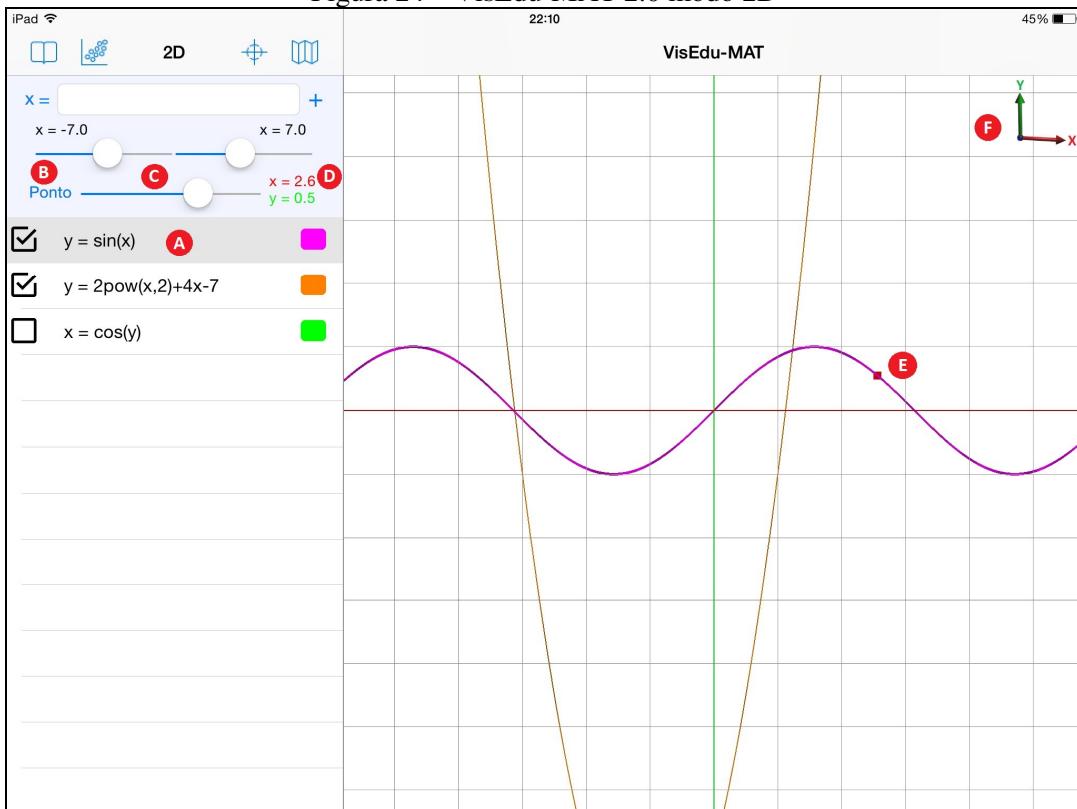
Ao alterar o modo de visualização para 2D, o VisEdu-MAT 2.0 irá carregar a tela conforme apresentado na Figura 24. O processo de criação e exibição de uma função matemática é o mesmo que no modo 3D exibido na seção anterior.

No modo 2D a câmera é ortográfica exibindo os valores dos eixos x e y, conforme a imagem de orientação (F). Além disso, o slider do tracing é mostrado para realizar o controle de *tracing* (C).

Conforme a Figura 24 quando uma função matemática (A) é selecionada pelo usuário, a cor de seu desenho fica mais intensa e controle de *tracing* é habilitado. O controle de *tracing* permite que o usuário visualize um ponto (E) sobre o desenho da função matemática selecionada. Ao interagir com o slider do tracing (C), o ponto sobre o desenho da função matemática é atualizado, assim como os valores do campo x e campo y (D). Para deixar de exibir este ponto na área de desenho, basta o usuário clicar no botão Ponto (B).

Na área de desenho, o usuário pode apenas interagir com a câmera para realizar *zoom*. Se o usuário tocar com dois dedos na área de desenho e distanciar ou aproximar seus dedos, o aplicativo aumenta ou diminui o *zoom* da câmera respectivamente.

Figura 24 – VisEdu-MAT 2.0 modo 2D



3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta um aplicativo para dispositivos iPad para visualização de material educacional, no qual fosse possível criar e exibir uma função matemática explícita além de interagir com a mesma em uma cena.

No objetivo inicial deste trabalho pretendia-se utilizar a *framework* Accelerate (Apêndice B) da plataforma iOS, a fim de realizar os cálculos complexos das funções matemáticas. Para a validação sintática das funções matemáticas pretendia-se desenvolver a especificação da gramática na ferramenta Gals. Porém ao começar a implementação foi identificado a biblioteca DDMathParser, responsável por realizar validação e cálculo de funções matemáticas. Desta forma essa biblioteca foi utilizada para atender essas funcionalidades. Quanto ao *framework* Accelerate, optou-se por deixar como uma extensão do trabalho desenvolvido. A utilização desse *framework* possivelmente melhorará o desempenho dos cálculos das funções matemáticas realizados na implementação.

No desenvolvimento do aplicativo foi identificado baixo desempenho na velocidade de cálculos e renderização dos desenhos ao se alterar os valores do domínio de uma função em tempo real. A solução foi realizar esse processo de cálculos e renderização somente após o usuário finalizar o toque no *slider* (componente de tela), responsável por alterar os valores do domínio da função matemática selecionada.

Para a avaliação qualitativa do aplicativo foi realizada uma entrevista com o professor de matemática Londero (2014), que destacou a necessidade do aplicativo em exibir funções e operadores básicos de forma mais prática, sem necessidade de acessar ao manual do usuário. Desta forma foi atendido esse requisito exibindo uma imagem contendo algumas funções e operadores na biblioteca didática do aplicativo, além de adicionar essa imagem ao manual do usuário. Outra sugestão do professor foi a opção de mostrar os planos coordenados (Anexo A), no modo 3D, que acompanhe seu tamanho de acordo com os valores limites do domínio dos objetos de cena. Contudo optou-se por deixar essa sugestão como uma extensão do trabalho desenvolvido.

3.4.1 Testes de desempenho do aplicativo

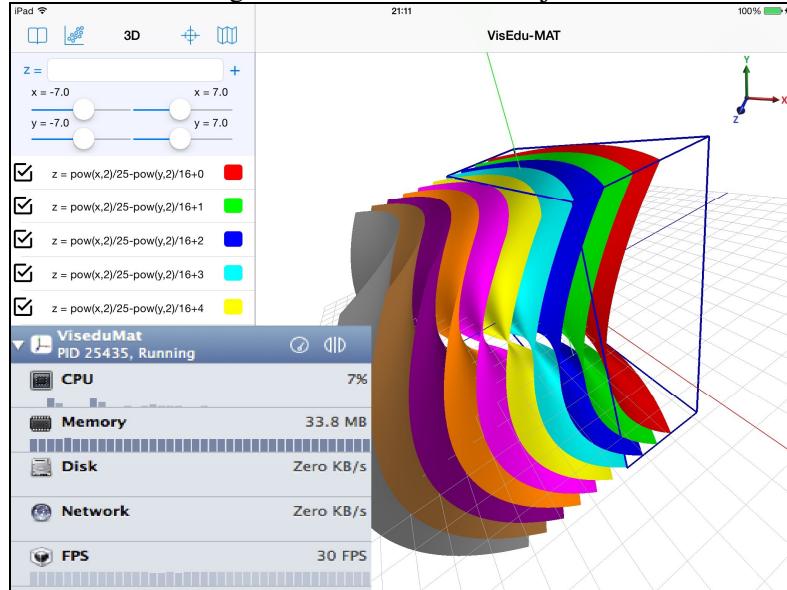
Para avaliar o desempenho do aplicativo foram aplicados testes adicionando novas funções matemáticas através do campo editável. O cenário escolhido foi o modo 3D, pois exige mais dos recursos de *hardware* para este tipo de aplicação.

A ferramenta utilizada para os testes foi o Instruments 6.1 (56160) que vem instalado juntamente com o Xcode 6.1 (6A1052d). Essa ferramenta permite a visualização do uso da CPU, memória, disco, rede e *Frames Per Second* (FPS). Nos testes a seguir não será relevante o valor exibido em rede, pois o aplicativo não realiza comunicação de dados.

Dos critérios utilizados para medir o desempenho foram observados o uso da CPU, que sofre variações decorrentes aos cálculos dos pontos que formam o desenho da função matemática e o processo de renderização; a memória, que aumenta com a quantidade de desenhos preparados para ser exibidos na área de desenho; o disco, que mede o consumo de espaço em disco usado pelo recurso de armazenamento local; o FPS, que exibe o processamento gráfico de todos os elementos que compõe a área de desenho.

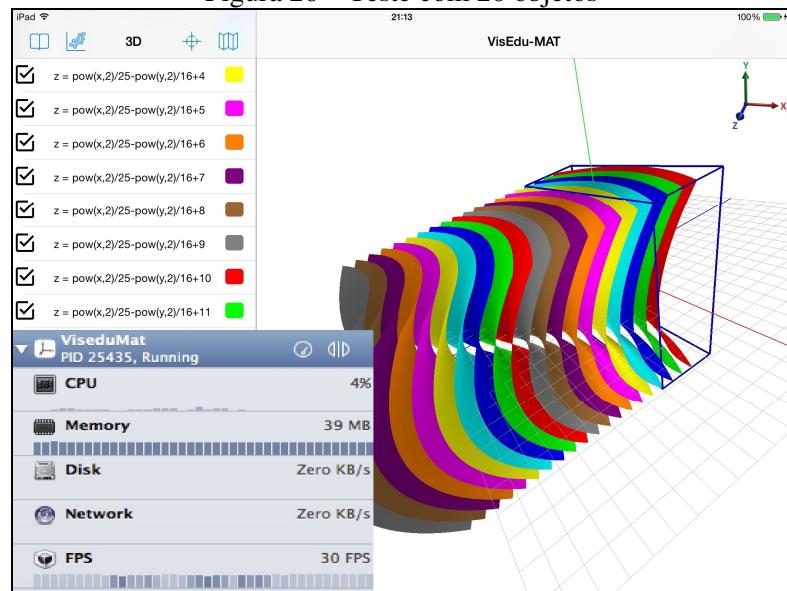
O objeto escolhido para testes foi a parabolóide hiperbólica e será utilizado nas 6 baterias de testes a seguir. No primeiro teste foram adicionados 10 objetos de cena no eixo z conforme visualizado na Figura 25.

Figura 25 – Teste com 10 objetos



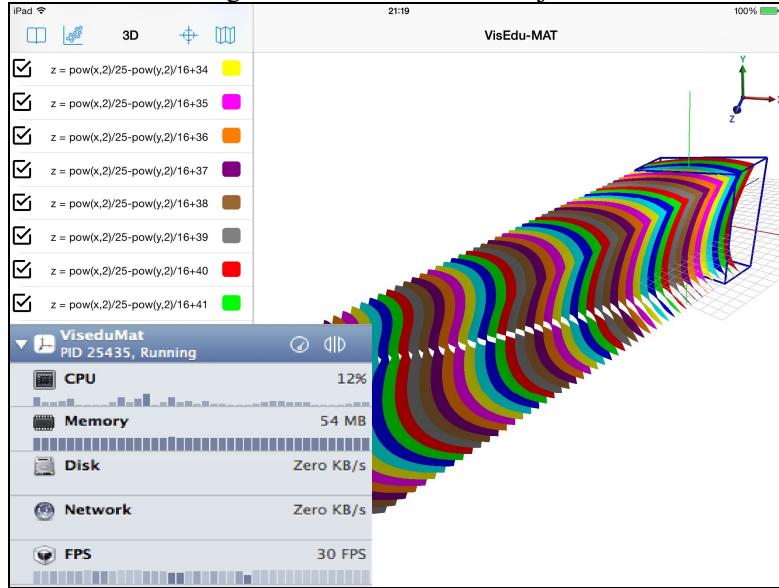
No segundo teste foram adicionados mais 10 objetos de cena também no eixo z conforme é verificado na Figura 26.

Figura 26 – Teste com 20 objetos



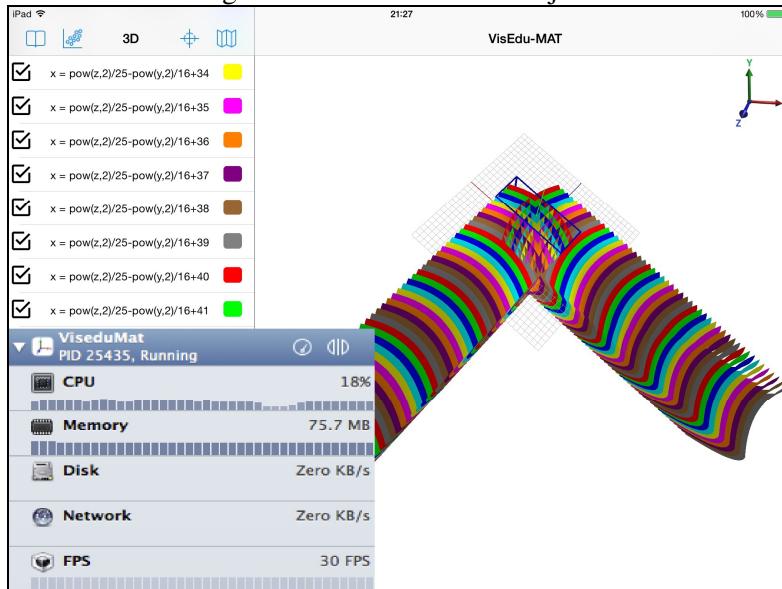
Ainda no eixo z, foram adicionados mais 30 objetos de cena, totalizando 50 objetos assim como exibidos na Figura 27.

Figura 27 – Teste com 50 objetos



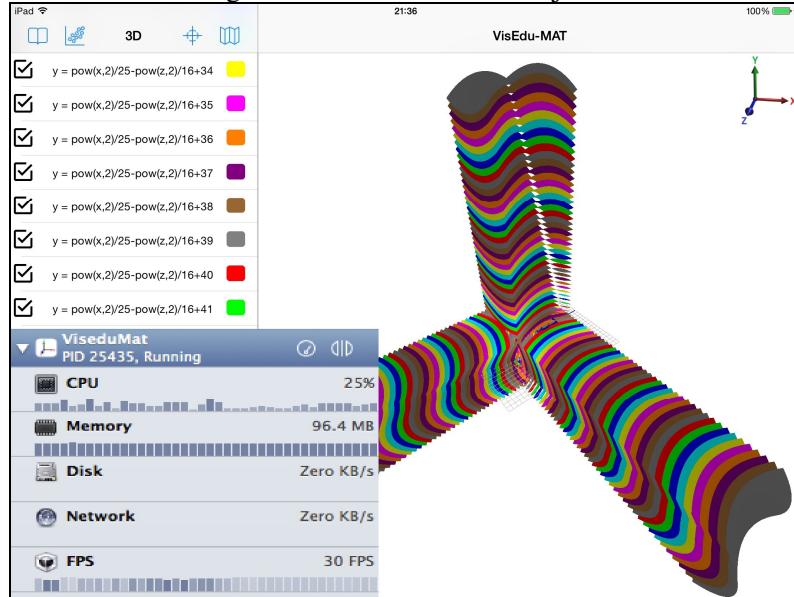
No eixo x foram adicionados 50 objetos, totalizando 100 objetos na área de desenho. A Figura 28 mostra com detalhes esses objetos.

Figura 28 – Teste com 100 objetos



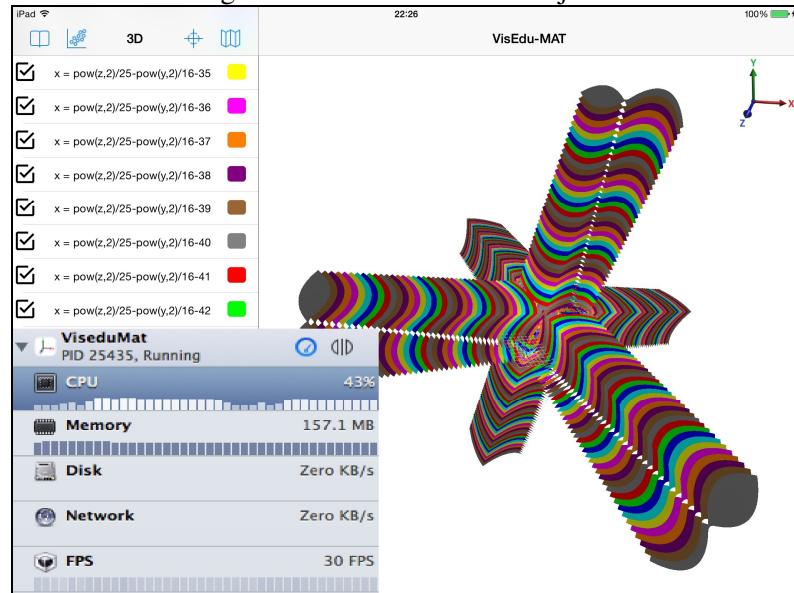
O eixo y também recebe 50 objetos, totalizando 150 objetos na área de desenho. A Figura 29 mostra todos os eixos com os objetos adicionados.

Figura 29 – Teste com 150 objetos



Para finalizar foram adicionados mais 50 objetos em cada eixo, totalizando 300 objetos conforme exibido na Figura 30.

Figura 30 – Testes com 300 objetos



O Quadro 9 demonstra os resultados dos testes realizados no aplicativo, destacando o uso da CPU, da memória, do espaço em disco e da taxa FPS para cada um dos seis testes realizados.

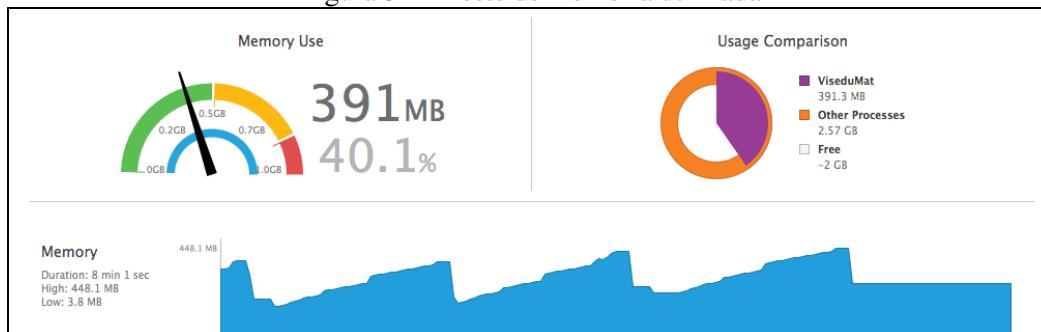
Quadro 9 – Testes de desempenho realizados no aplicativo

| | 10 objetos | 20 objetos | 50 objetos | 100 objetos | 150 objetos | 300 objetos |
|---------|------------|------------|------------|-------------|-------------|-------------|
| CPU | 7% | 4% | 12% | 18% | 25% | 43% |
| Memória | 33.8 Mb | 39 Mb | 54 Mb | 75.7 Mb | 96.4 Mb | 157.1 Mb |
| Disco | 0 Kb | 0 Kb | 0 Kb | 0 Kb | 0 Kb | 0 Kb |
| FPS | 30 | 30 | 30 | 30 | 30 | 30 |

Conforme observado no Quadro 9 em relação ao processamento da CPU, percebe-se que há um aumento significativo passando de 12% (50 objetos), 25% (150 objetos) e chegando aos 43% de seu uso assim que adicionado 300 objetos. De acordo com os números de memória de cada bateria de testes, pode-se concluir que a cada 50 objetos adicionados ao aplicativo é necessário em torno de 20 Mb de memória disponível ou se preferir 0.4 Mb de memória a cada objeto adicionado. As quantidades utilizadas em disco para armazenamento local das funções matemáticas são tão pequenas que não aparecem nos testes. Por fim, os valores de processamento em 30 FPS não mudaram mesmo com 300 objetos adicionados na área de desenho.

Após outros testes foi evidenciado que o aplicativo não está liberando memória corretamente, pois ao sair do modo 3D para o modo 2D e retornar ao modo 3D o uso da memória está sempre aumentando conforme mostrado na Figura 31.

Figura 31 – Teste de memória utilizada



A Figura 31 mostra um caso de testes com 40 objetos adicionados. Inicialmente estes ocupavam 49 Mb de memória, por fim estão ocupando 391 Mb. Os picos do gráfico exibidos na figura demonstram o momento em que foi alternado do modo 3D para o modo 2D e ligeiramente retornado ao modo 3D.

A solução para este problema é buscar uma estratégia correta para liberar a memória administrada pela GPU, no momento em que se alterna entre os modos 2D e 3D, através das referências de memória VBO dos objetos gráficos.

3.4.2 Comparativo entre os trabalhos correlatos

O Quadro 10 lista as principais características e funcionalidades em comum dos trabalhos correlatos (ver seção 2.4) com o aplicativo VisEdu-MAT 2.0 desenvolvido neste trabalho.

Quadro 10 – Comparaçāo dos trabalhos correlatos e o VisEdu-MAT 2.0

| | VisEdu-MAT 2.0 * | VisEdu-MAT 1.0 | Quick Graph | Grafikal De Ageio |
|---|--------------------|--------------------|-------------|-------------------|
| Desenvolvido em plataforma | iPad | Web | iPad/iPhone | iPad/iPhone |
| Modo de visualização 2D e 3D | Sim | Sim | Sim | Sim |
| Illuminação em modo 3D | Sim | Sim | Sim | Sim |
| Permite ampliar, mover e girar a câmera | Sim | Sim | Sim | Sim |
| Suporta funções explícitas e implícitas | Somente explícitas | Somente explícitas | Sim | Sim |
| Sistemas de coodernadas: cartesiano, polar, cilíndrico e esférico | Somente cartesiano | Somente cartesiano | Sim | Sim |
| Possui função <i>tracing</i> em modo 2D | Sim | Sim | Sim | Sim |
| Faz validação sintática ao digitar função | Sim | Sim | Sim | Sim |
| Salva funções utilizadas | Sim | Sim | Sim | Sim |
| Possui biblioteca de exemplos didáticos | Sim | Sim | Sim | Sim |
| Permite alterar valores do domínio | Sim | Sim | Sim | Sim |

* Trabalho desenvolvido

Conforme observado no Quadro 10 tanto o VisEdu-MAT 1.0 quanto o VisEdu-MAT 2.0 suportam apenas funções explícitas, enquanto que o Quick Graph e o Grafikal De Ageio suportam funções explícitas e implícitas. O sistema de coordenadas disponível no VisEdu-MAT 1.0 e VisEdu-MAT 2.0 é apenas o cartesiano, enquanto que o Quick Graph e o Grafikal De Ageio disponibilizam os sistemas de coordenadas: cartesiano, polar, cilíndrico e esférico. Por fim, todos os trabalhos correlatos e o VisEdu-MAT 2.0 atendem as características e funcionalidades restantes exibidas no Quadro 10.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um aplicativo para dispositivos iPad para visualização de funções matemáticas explícitas. No desenvolvimento foram encontradas dificuldades para trabalhar com a linguagem Objective-C e o OpenGL ES causado pela falta de experiência. Contudo todos os objetivos propostos foram alcançados ao substituir a implementação do analisador sintático e a especificação da gramática pelo uso da biblioteca DDMathParser. Esta biblioteca realiza a validação e o cálculo de cada ponto que compõe o desenho da função matemática do aplicativo desenvolvido e demonstrou-se muito eficiente, atendendo as necessidades dos tipos de funções explícitas.

O VisEdu-MAT 2.0 está disponível tanto em modo 2D quanto em 3D. Permite a inserção de uma função matemática explícita realizando sua validação e exibindo como resultado um desenho gráfico. Permite escolher o contradomínio ao inserir uma nova função matemática. Os valores do domínio da função matemática são parametrizáveis. Há possibilidade de mudar a cor e a visibilidade da função matemática. O tipo de primitiva pode ser alterado no modo 3D. Todas as funções matemáticas são gravadas no banco de dados do iPad. Uma lista de funções permite a seleção e destaque do desenho de uma função matemática. Permite exibir o *tracing* de um ponto no modo 2D. Aplica iluminação no modo 3D para exibir profundidade do desenho. Por fim, a interação com o usuário é realizada de maneira prática e intuitiva.

As principais limitações deste aplicativo é a incapacidade de suportar funções implícitas e utilizar somente o sistema de coordenadas no plano cartesiano. De acordo com os testes do aplicativo VisEdu-MAT 2.0 a memória não está sendo liberada ao alternar entre o modo 3D e modo 2D.

A importância deste trabalho é o sucesso da portabilidade do aplicativo VisEdu-MAT na plataforma iOS, uma vez que ele compõe o projeto VisEdu-MAT (2013). Desta forma espera-se que o aplicativo continue em constante evolução e aperfeiçoamento, pois a iniciativa claramente irá contribuir no aprendizado da matemática.

4.1 EXTENSÕES

Durante as etapas finais do desenvolvimento do VisEdu-MAT 2.0, foram identificados algumas melhorias e sugestões para futuras extensões, são elas:

- a) utilizar o *framework* Accelerate para realizar os cálculos das funções;
- b) suportar a entrada funções matemáticas implícitas no modo 2D e 3D;
- c) adicionar novas funções e operadores customizados a biblioteca DDMathParser;

- d) aceitar vírgula ao invés de ponto como separador decimal;
- e) permitir alterar o sistema de coordenadas para cartesiano e polar no modo 2D;
- f) permitir alterar o sistema de coordenadas para cartesiano, cilíndrico e esférico no modo 3D;
- g) adicionar valores numéricos nos eixos de orientação;
- h) adicionar setas nas pontas dos eixos de orientação;
- i) permitir a inspeção de valores de uma função matemática no modo 3D, assim como é realizado no modo 2D através do *tracing*;
- j) gerar, importar e compartilhar via *Bluetooth* e nuvem, arquivos de texto e imagem das funções matemáticas no modo 2D e 3D;
- k) mostrar planos coordenados no modo 3D;
- l) ajustar o aplicativo para que libere memória ao alternar entre os modos 2D e 3D;
- m) usar um editor de funções mais próximo da simbologia usada em aula, com nomes de funções em português.

REFERÊNCIAS

- APPLE. **OpenGL ES on iOS.** [Cupertino], 2013a. Disponível em:
[<http://developer.apple.com/library/ios/#documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLESontheiPhone/OpenGLESontheiPhone.html#/apple_ref/doc/uid/TP40008793-CH101-SW1>](http://developer.apple.com/library/ios/#documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/OpenGLESontheiPhone/OpenGLESontheiPhone.html#/apple_ref/doc/uid/TP40008793-CH101-SW1). Acesso em: 14 ago. 2013.
- _____. **vDSP programming guide.** [Cupertino], 2013b. Disponível em:
[<https://developer.apple.com/library/ios/#documentation/Performance/Conceptual/vDSP_Programming_Guide/About_vDSP/About_vDSP.html#/apple_ref/doc/uid/TP40005147-CH201-SW1>](https://developer.apple.com/library/ios/#documentation/Performance/Conceptual/vDSP_Programming_Guide/About_vDSP/About_vDSP.html#/apple_ref/doc/uid/TP40005147-CH201-SW1). Acesso em: 05 set. 2013.
- APPSTORE. **Grafikal De Ageio:** by Apple. [Cupertino], 2013a. Disponível em:
[<https://itunes.apple.com/br/app/grafikal/id519933025?mt=8>](https://itunes.apple.com/br/app/grafikal/id519933025?mt=8). Acesso em: 12 ago. 2013.
- _____. **Quick graph:** by Apple. [Cupertino], 2013b. Disponível em:
[<https://itunes.apple.com/br/app/quick-graph-your-scientific/id292412367?l=en&mt=8>](https://itunes.apple.com/br/app/quick-graph-your-scientific/id292412367?l=en&mt=8). Acesso em: 12 ago. 2013.
- BALBO, Antonio R.. **Diferenciação implícita.** [Bauru], 2014. Disponível em:
[<http://wwwp.fc.unesp.br/~arbalbo/arquivos/derivadaimplicitaediferenciais.pdf>](http://wwwp.fc.unesp.br/~arbalbo/arquivos/derivadaimplicitaediferenciais.pdf). Acesso em: 12 set. 2014.
- BARBOSA, Rommel M. **Ambientes virtuais de aprendizagem.** Porto Alegre: ArTmed, 2005. 182 p.
- BONJORNO, José R.; GIOVANNI, José R.; GIOVANNI JÚNIOR, José R.. **Matemática fundamental:** 2. grau, volume único. São Paulo : Ed. FTD, 1994. 560 p.
- GINSBURG, Dan; MUNSHI, Aaftab; SHREINER, Dave. **OpenGL ES 2.0 programming guide.** Indianapolis: Pearson Education, 2009. 457 p.
- GITHUB. **DDMathParser add new functions.** [San Francisco], 2014a. Disponível em:
[<https://github.com/davedelong/DDMathParser/wiki/Adding-New-Functions>](https://github.com/davedelong/DDMathParser/wiki/Adding-New-Functions). Acesso em: 02 nov. 2014.
- _____. **DDMathParser add new operators.** [San Francisco], 2014b. Disponível em:
[<https://github.com/davedelong/DDMathParser/wiki/Adding-New-Operators>](https://github.com/davedelong/DDMathParser/wiki/Adding-New-Operators). Acesso em: 02 nov. 2014.
- _____. **DDMathParser home.** [San Francisco], 2014c. Disponível em:
[<https://github.com/davedelong/DDMathParser>](https://github.com/davedelong/DDMathParser). Acesso em: 02 nov. 2014.
- INTMATH. **Graphs of the trigonometric functions.** [Singapore], 2014. Disponível em:
[<http://www.intmath.com/trigonometric-graphs/trigo-graph-intro.php>](http://www.intmath.com/trigonometric-graphs/trigo-graph-intro.php). Acesso em: 11 dez. 2014.
- KRAUSS, José Ricardo. **Visedu-mat: visualizador de material educacional, módulo de matemática.** 2013. 63 f, il. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau, 2013. Disponível em: <http://www.bc.furb.br/docs/MO/2013/356126_1_1.pdf>. Acesso em: 26 ago. 2014.
- LEARNOPENGLS. **Left-handed and right-handed coordinate systems.** [Washington], 2014. Disponível em: <http://www.learnopengl.com/understanding-opengl-matrices/left_right_hand/>. Acesso em: 13 set. 2014.

LONDERO, Evandro F. **Entrevista sobre elicitação de requisitos.** Entrevistadores: Osvaldo Bay Machado e Dalton Solano dos Reis. Blumenau. 2014. Entrevista feita através de conversão – não publicada.

MARTINS, José A.; MIOTTO, Fernanda; VILLAS-BOAS, Valquíria. **Novas metodologias para o ensino médio em ciências, matemática e tecnologia.** Brasília, D.F : Ed. ABENGE, 2011. 386 p.

MSDN. **Displaying graphics:** what is a stencil buffer? [Washington], 2013. Disponível em: <<http://msdn.microsoft.com/en-us/library/bb976074.aspx>>. Acesso em: 03 set. 2013.

OLIVEIRA, Vera B.; VIGNERON, Jacques M. J. **Sala de aula e tecnologias.** São Bernardo do Campo: Umesp, 2005. 142 p.

PAIVA, Manuel R.. **Matemática:** 2. grau, volume 1. São Paulo : Ed. Moderna, 1995. 653 p.

RIDEOUT, Philip. **iPhone 3D programming.** 3. ed. Sebastopol: O'Reilly Media, 2010. 420 p.

SHOEP, Frank. **Vectorizing with vDSP and vecLib.** [S.l], 2006. Disponível em: <<http://www.ffnn.nl/pages/articles/apple-mac-os-x/vectorizing-with-vdsp-and-veclib.php>>. Acesso em: 26 ago. 2013.

VISEDU-MAT. **Visualizador de material educacional, módulo de matemática.** [Blumenau], 2013. Disponível em: <http://gcb.inf.furb.br/?page_id=1147>. Acesso em: 26 ago. 2014.

APÊNDICE A – Manual do Usuário

Neste apêndice é apresentado o manual de ajuda do usuário, que pode ser acessado através do botão **biblioteca** na barra de botões. O manual possui várias páginas, desta forma será exibida cada uma delas.

Figura 32 – Manual do usuário página 1

1

Manual do Usuário

O aplicativo VisEdu-MAT 2.0 permite a exibição de funções matemáticas explícitas no modo 2D e 3D em dispositivos iPad. Neste documento o usuário será capaz de entender as funcionalidades propostas pelo aplicativo.

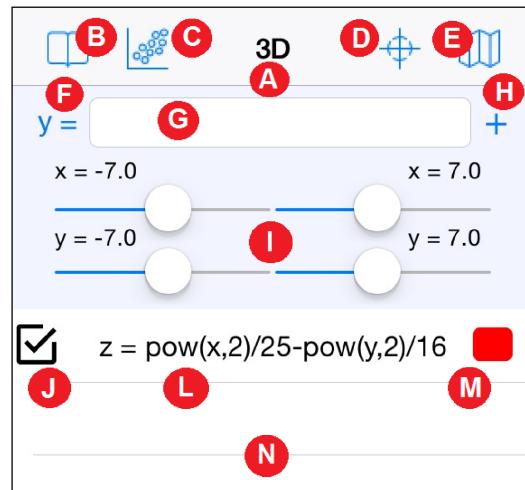
A imagem abaixo mostra a tela principal do VisEdu-MAT 2.0 em modo 3D. Posicionado a esquerda desta tela são mostrados os principais controles e a lista de funções matemáticas. A área de desenho que compreende a maior área está posicionada a direita desta tela. Nesta área são exibidos os desenhos correspondentes a função matemática presente na lista de funções.

A screenshot of the VisEdu-MAT 2.0 application interface. The top status bar shows 'iPad', signal strength, battery level at 42%, and the time '23:00'. The top navigation bar has icons for 2D, 3D, and other functions. Below the navigation bar is a control panel with sliders for 'x = -7.0' and 'x = 7.0', and a slider for 'z = -7.0' and 'z = 7.0'. To the right of the control panel is a 3D coordinate system with axes labeled x, y, and z. The main drawing area shows a 3D grid and a blue curve plotted on it. The curve starts from the bottom left, goes up to a peak, and then down again, resembling a wave or a function like y = sin(x). The overall interface is clean and modern, designed for touch interaction on an iPad.

Figura 33 – Manual do usuário página 2

2

Principais Controles



- A – Modo selecionado 2D ou 3D
- B – Exibe biblioteca didática
- C – Muda primitiva no modo 3D (*Mesh, Wireframe, Points*)
- D – Centraliza câmera
- E – Altera entre modo 2D e 3D
- F – Muda contradomínio da função
- G – Campo editável da função
- H – Insere nova função
- I – Altera e exibe valores do contradomínio da função selecionada
- J – Altera visibilidade da função
- L – Texto da função inserida
- M – Cor do desenho resultante da função inserida
- N – Lista das funções matemáticas inseridas

Figura 34 – Manual do usuário página 3

3

Funcionalidades no Modo 3D

- O usuário insere uma função no campo editável, configura o contradomínio (x, y, z) e clica no botão +.

A screenshot of a software interface. A text input field contains the mathematical expression $y = \text{pow}(x,2)+\text{pow}(y,2)$. To the right of the input field is a blue rectangular button with a white plus sign (+).

- O aplicativo exibe uma mensagem de erro caso a função inserida for inválida.



- Se a função matemática passar pela validação será mostrado o desenho resultante na área de desenho. O desenho no canto superior direito mostra a orientação nos eixos x, y e z de acordo com o sistema de coordenadas.

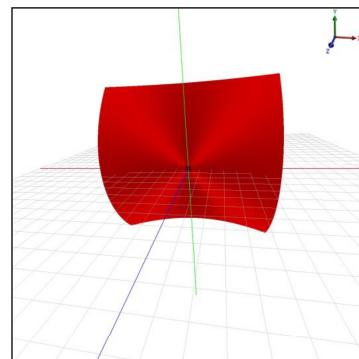


Figura 35 – Manual do usuário página 4

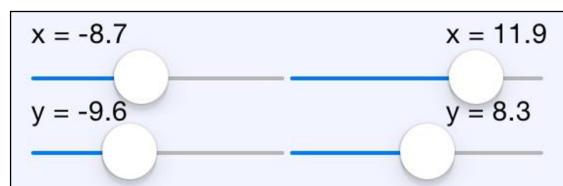
4

4. Ao selecionar uma função na lista de funções esta ficará destacada com a cor cinza.



$z = \text{pow}(x,2)/25-\text{pow}(y,2)/16$

5. Os controles de domínio permitem interação com o usuário. Eles são atualizados no momento em que a função for selecionada.



6. O *Bounding Box* (caixa nos limites da borda) exibido sobre o desenho da função é o resultado da seleção de uma função matemática.

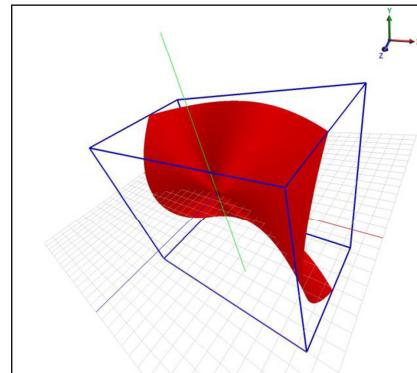
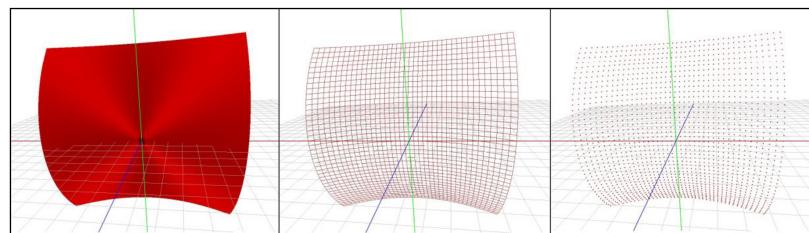


Figura 36 – Manual do usuário página 5

5

7. A primitiva do desenho está disponível no modo 3D a partir do botão mudar primitiva. Da esquerda para a direita são mostradas as primitivas disponíveis do tipo *Mesh*, *Wireframe* e *Points*.



8. Para excluir função da lista de funções, o usuário deverá arrastar o dedo da direita à esquerda sobre uma função na lista de funções e clicar no botão Delete.

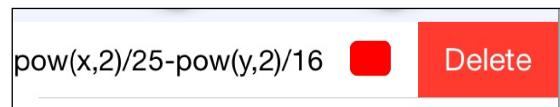


Figura 37 – Manual do usuário página 6

6

Funcionalidades no Modo 2D

O modo 2D possui as funcionalidades básicas do modo 3D, exceto pela visualização de primitivas nos desenhos e a seleção do tipo *Bounding Box*.

1. O usuário insere com sucesso uma função, conforme visualizado na lista de funções matemáticas.



2. O modo 2D habilita controle de *tracing*. Ao selecionar a função na lista de funções o usuário pode interagir com o controle *tracing* para inspecionar o valor em um determinado ponto da função matemática.



3. O desenho resultante é mostrado na área de desenho. A seleção da função na lista de funções deixa o desenho desta função com sua cor destacada. O ponto do *tracing* é exibido sobre o desenho da função e permite o usuário visualizar o ponto que está sendo inspecionado.

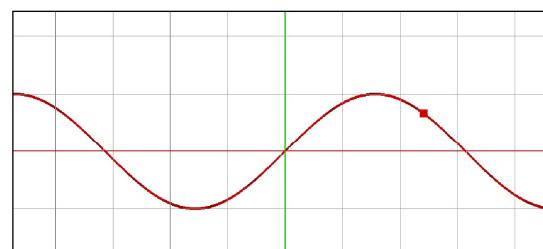
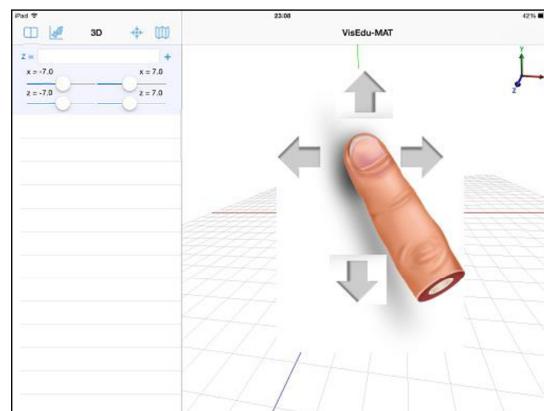


Figura 38 – Manual do usuário página 7

7

Interação na Área de Desenho

- O usuário pode realizar a translação da câmera presente na área de desenho utilizando apenas um dedo sobre essa área. Funcionalidade habilitada somente no modo 3D.



- Com dois dedos sobre a área de desenho ao distanciar ou aproximar os mesmos ocorrerá o *zoom* da câmera.

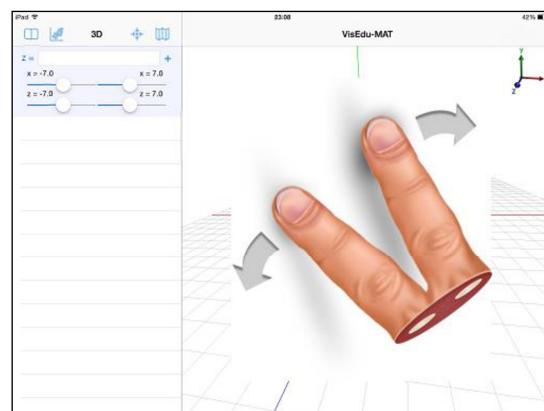


Figura 39 – Manual do usuário página 8

8

3. Para rotacionar a câmera da área de desenho, basta colocar dois dedos sobre esta área e movimentar os dedos para uma direção. Funcionalidade habilitada somente no modo 3D.

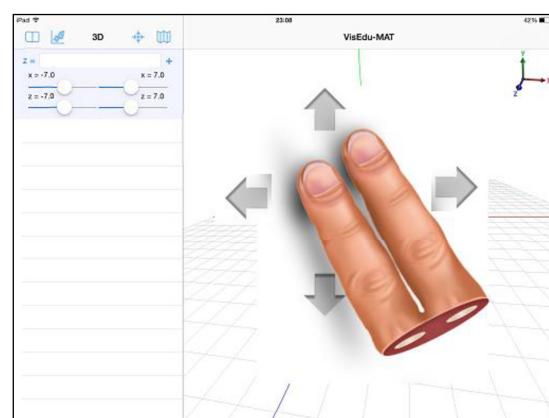


Figura 40 – Manual do usuário página 9

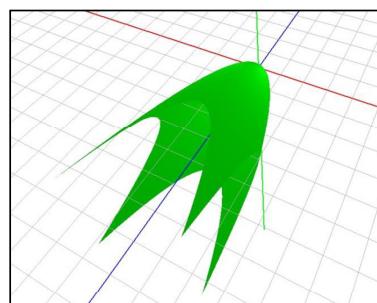
9

Biblioteca Didática Modo 3D

Abaixo são mostrados exemplos de funções matemáticas explícitas suportadas no aplicativo VisEdu-MAT 2.0 na versão 3D.

Paraboloide Infinita

$$z = \text{pow}(x,2)+\text{pow}(y,2)$$



Paraboloide Hiperbólica

$$z = \text{pow}(x,2)/25-\text{pow}(y,2)/16$$

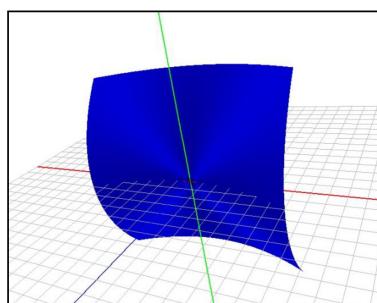


Figura 41 – Manual do usuário página 10

10

Biblioteca Didática Modo 2D

Abaixo são exibidas funções trigonométricas como exemplo de funções matemáticas explícitas suportadas no aplicativo VisEdu-MAT 2.0 na versão 2D.

Função seno de x

$$y = \sin(x)$$

**Função cosseno de x**

$$y = \cos(x)$$

**Função tangente de x**

$$y = \tan(x)$$

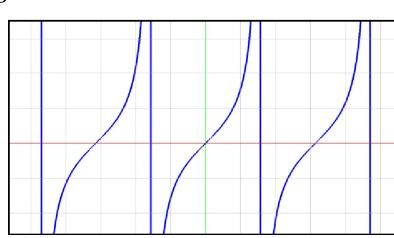


Figura 42 – Manual do usuário página 11

11

Funções Suportadas

O VisEdu-MAT 2.0 utiliza a biblioteca de licença gratuita DDMParser para realizar a validação e cálculos da função matemática. As funções suportadas por padrão desta biblioteca são disponibilizadas neste documento e foram retirados do seguinte site:
<https://github.com/davedelong/DDMParser/wiki/Built-in-Functions>.

Functions that take > 1 parameter

- `sum()` - returns a sum of the passed parameters
- `count()` - returns the number of passed parameters
- `min()` - returns the minimum of the passed parameters
- `max()` - returns the maximum of the passed parameters
- `median()` - returns the median of the passed parameters
- `stddev()` - returns the standard deviation of the passed parameters
- `average()` - returns the average of the passed parameters
- `random()` - returns a random integer. Can take 0, 1, or 2 parameters. The first parameter (if given) is the lower bound of the random integer. The second parameter (if given) is the upper bound of the random integer.
- `nthroot()` - returns the n^{th} root of a number. For example, `nthroot(27,3)` returns the cube root of 27, or 3.

Functions that take 1 parameter:

- `sqrt()` - returns the square root of the passed parameter
- `log()` - returns the base 10 log of the passed parameter
- `ln()` - returns the base e log of the passed parameter
- `log2()` - returns the base 2 log of the passed parameter
- `exp()` - returns e raised to the power of the passed parameter
- `ceil()` - returns the passed parameter rounded up
- `floor()` - returns the passed parameter rounded down

Figura 43 – Manual do usuário página 12

12

Funções Suportadas (continuação)

- The trigonometric functions:

- `sin()` , `cos()` , `tan()`
- Their inverses (`asin` , `acos` , `atan`)
- Their reciprocals (`csc` , `sec` , `cotan`)
- The reciprocals of the inverses (`acsc` , `asec` , `acotan`)
- The hyperbolic variations of all the above functions (`sinh` , `cosh` , `tanh` ,
`asinh` , `acosh` , `atanh` , `csch` , `sech` , `cotanh` , `acsch` , `asech` , `acotanh`)
- The versine functions (`versin` , `vercosin` , `coversin` , `covercosin` ,
`haversin` , `havercosin` , `hacoversin` , `hacovercosin` , `exsec` , `excsc` ,
`crd`)

- `dtor()` - converts the passed parameter from degrees to radians
- `rtod()` - converts the passed parameter from radians to degrees

Functions that take no parameters ("constant functions"):

- `phi()` - returns the value of ϕ (the Golden Ratio). Also recognized as `ψ()`
- `pi()` - returns the value of π . Also recognized as `π()`
- `pi_2()` - returns the value of $\pi/2$
- `pi_4()` - returns the value of $\pi/4$
- `tau()` - returns the value of τ . Also recognized as `τ()`
- `sqr2()` - returns the value of the square root of 2
- `e()` - returns the value of e
- `log2e()` - returns the value of the log base 2 of e
- `log10e()` - returns the value of the log base 10 of e
- `ln2()` - returns the value of the log base e of 2
- `ln10()` - returns the value of the log base e of 10

Figura 44 – Manual do usuário página 13

13

Operadores Suportados

O VisEdu-MAT 2.0 utiliza a biblioteca de licença gratuita DDMParser para realizar a validação e cálculos da função matemática. Os operadores suportados por padrão desta biblioteca são disponibilizados neste documento e foram retirados do seguinte site:
<https://github.com/davedelong/DDMParser/wiki/Operators>.

| Operator | Function | Description |
|---------------------------|---------------------|------------------------------------|
| Standard Operators | | |
| + | add | addition and unary positive |
| - or - | subtract and negate | subtraction and negation |
| * or x | multiply | multiplication |
| / or ÷ | divide | division |
| % | mod OR percent | modulus or a percentage of a value |
| ! | factorial | factorial |
| ** | pow | exponentiation |
| ° or ° | dtror | converts the value to radians |

Figura 45 – Manual do usuário página 14

14

Operadores Suportados (continuação)

| Bitwise Operators | | |
|----------------------|--------|-----------------------|
| & | and | bitwise and |
| | or | bitwise or |
| ^ | xor | bitwise xor |
| ~ | not | bitwise not |
| << | lshift | bitwise left shift |
| >> | rshift | bitwise right shift |
| Comparison Operators | | |
| == or = | l_eq | equal |
| != | l_neq | not equal |
| < | l_lt | less than |
| > | l_gt | greater than |
| <= or =< or ≤ or âf; | l_ltoe | less than or equal |
| >= or => or ≥ or âe; | l_gtoe | greater than or equal |
| Logical Operators | | |
| && or ^ | l_and | logical and |
| or v | l_or | logical or |
| ! or ~ | l_not | logical not |

Figura 46 – Manual do usuário página 15

15

Lista de Operadores e Funções Básicas

O quadro abaixo exibe a lista de operadores e funções básicas.

| Operador | Notação | Exemplo |
|---------------|---------|-------------------------|
| Soma | + | $5 + 2$ |
| Subtração | - | $10 - 2$ |
| Multiplicação | * ou x | $5 * 5$ ou 5×5 |
| Divisão | / | $6 / 3$ |
| Função | Notação | Exemplo |
| Potência | pow() | pow(x, 2) |
| Raiz quadrada | sqrt() | sqrt(9) |
| Logaritmo | log() | log(2) |
| Seno | sin() | sin(1) |
| Cosseno | cos() | cos(2) |
| Tangente | tan() | tan(3) |

APÊNDICE B – Framework Accelerate

Este apêndice mostra a pesquisa realizada sobre o *framework* Accelerate presente no SDK do iOS. Este *framework* não foi utilizado no desenvolvimento do VisEdu-MAT 2.0, entretanto pode ser utilizado em futuras extensões.

O *framework* Accelerate, formado por várias APIs, é uma coleção de funções e objetos que podem ser usados para obter um grande aumento na velocidade do processamento na *Central Processing Unit* (CPU) quando se está trabalhando com conjunto de dados vetoriais, permitindo realizar operações matemáticas complexas ou cálculos de imagem.

Segundo Apple (2013b), a API *vector Digital Signal Processing* (vDSP) fornece funções matemáticas para aplicações como voz, som, processamento de áudio e vídeo, processamento de radar, análise sísmica e processamento de dados científicos, podendo operar em tipos de dados reais e complexos, incluindo conversões de tipo de dados, *Fast Fourier Transform* (FFT), operações vetoriais e escalares.

Em todas as funções vDSP os argumentos são passados via referência, iniciando com prefixo `vDSP_`, que pode ser seguido de `v` se a função opera com valores de vetor na entrada, ou `vs` para valores escalares. As funções podem terminar com `D`, demonstrando que a operação funciona com números de ponto flutuante de dupla precisão, como exemplo na função de multiplicação entre vetores `vDSP_vmulD`, que é basicamente o mesmo que `vDSP_vmul`, a não ser por funcionar em números de dupla precisão (SHOEP, 2006).

Na Figura 47 é demonstrado o uso de uma simples função aritmética vetorial de adição, resultando no vetor de saída, a soma dos dois vetores de entrada.

Figura 47 – Função aritmética vetorial de adição

```
#include <Accelerate/Accelerate.h>
int main() {
    float vetorEntrada1[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    float vetorEntrada2[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    float vetorSaida[8];
    vDSP_vadd(vetorEntrada1, 1, vetorEntrada2, 1, vetorSaida, 1, 8);
}
```

No exemplo da Figura 47 são passados na função `vDSP_vadd` como argumentos os vetores de entrada e vetor de saída. Ao fim desses, como próximo argumento, pode ser notado o número `1`, que significa número de passos no vetor. Neste caso deve ser `1`, porém se estivesse trabalhando com vetores que guardam informações de cores simbolizadas por três inteiros seqüenciais como o padrão *Red Green Blue* (RGB), então poderia ser utilizado para a soma de valores em um canal de cor. Por fim, no último parâmetro foi informado o tamanho dos vetores utilizados.

Neste exemplo são passados na função `vDSP_vadd` como argumentos os vetores de entrada `vetorEntrada1` e `vetorEntrada2`, e vetor de saída `vetorSaida`. O valor 1 após `vetorEntrada1` e `vetorEntrada2` significa que a adição será realizada com o intervalo de um passo e o valor 8 apenas informa o tamanho dos vetores utilizados.

De forma semelhante, na Figura 48 a função de multiplicação `vDSP_vsmul`, do tipo aritmética escalar, multiplica o valor de `pi` para cada elemento do vetor de entrada, resultando o vetor de saída.

Figura 48 – Função aritmética escalar de multiplicação

```
#include <Accelerate/Accelerate.h>
#include <math.h>

int main() {
    float vetorEntrada[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    float vetorSaida[8];
    float pi = M_PI;

    vDSP_vsmul(vetorEntrada, 1, &pi, vetorSaida, 1, 8);
}
```

Além das funções vetoriais e escalares de adição, subtração, multiplicação e divisão, existem outras como: `vDSP_vdist` que calcula a distância entre elementos em dois vetores; `vDSP_vindex` que copia elementos de um vetor de entrada em um vetor de saída, baseado nos índices do segundo vetor de entrada; `vDSP_mmov` que copia elementos de um vetor em outro; `vDSP_vfill` que preenche todos os elementos de um vetor para um determinado escalar; `vDSP_vclr` que preenche com zero todos os elementos de um vetor.

Outra API do framework Accelerate é a vecLib, que possibilita acesso a funções trigonométricas, logarítmicas, exponenciais, utilizadas de forma semelhante ao vDSP, a não ser pela ordem dos parâmetros e os tipos de dados. Iniciado por `vv`, indica que a função trabalha e resulta em um vetor de ponto flutuante de dupla precisão, contudo se o nome da função for finalizado com `f`, a função opera com ponto flutuante de precisão simples (SHOEP, 2006).

No exemplo da Figura 49 é possível verificar a chamada de função trigonométrica que calcula a tangente a partir do vetor de entrada, preenchendo os valores resultantes no vetor de saída.

Figura 49 – Função tangente

```
#include <Accelerate/Accelerate.h>

int main() {
    float vetorEntrada[8] = {1, 2, 3, 4, 5, 6, 7, 8};
    float vetorSaida[8];

    vvtnaf(vetorSaida, vetorEntrada, 8);
}
```

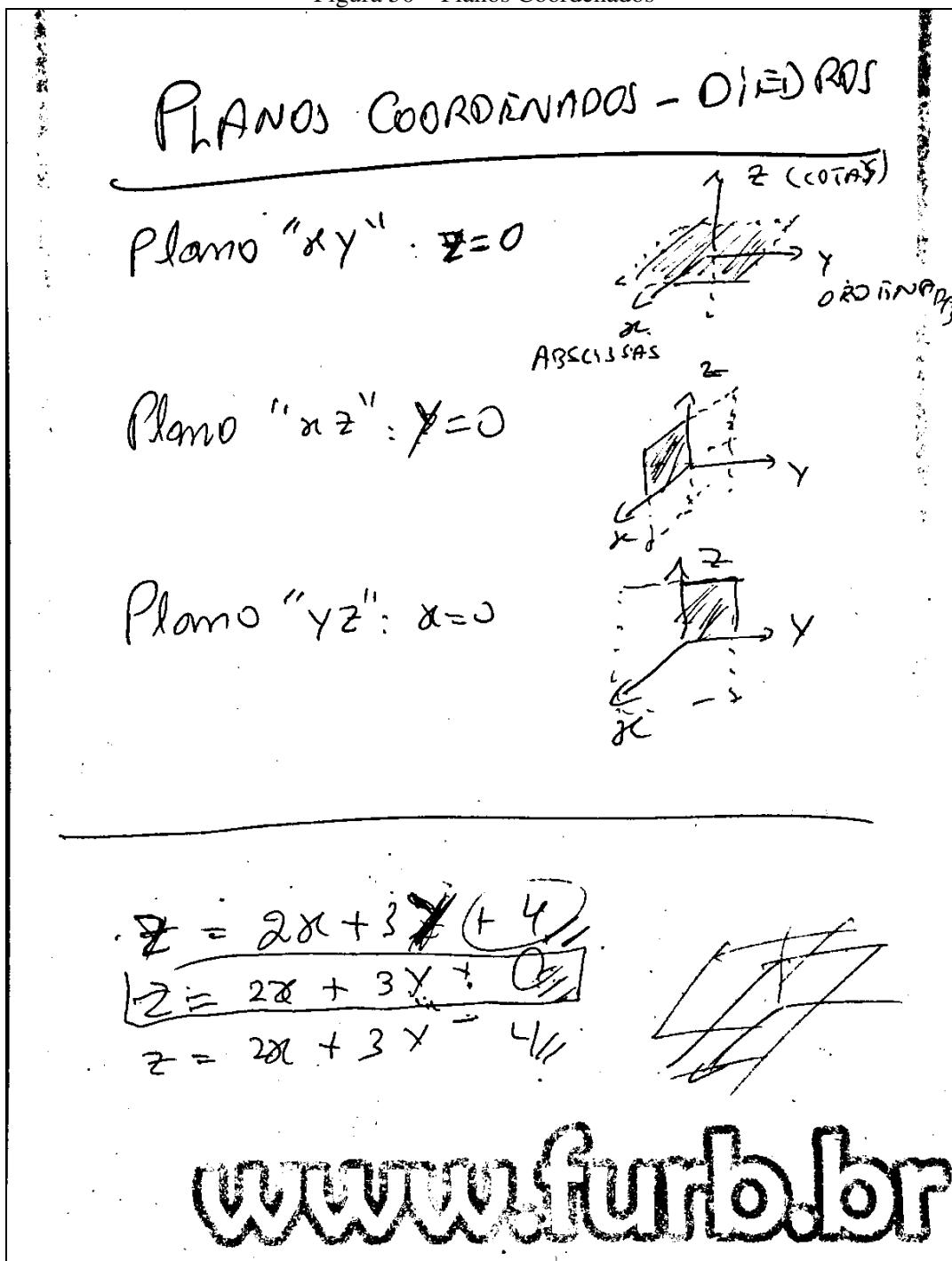
Entre as funções logarítmicas e exponenciais são encontradas: `vvlogf` que calcula o logaritmo natural de cada elemento do vetor; `vvlog10f` que calcula o logaritmo na base 10

para cada elemento do vetor; `vvexpf` que calcula o exponencial de cada elemento do vetor; `vvrecf` que calcula o inverso para cada elemento do vetor.

ANEXO A – Planos Coordenados

Este anexo contém informações decorrentes da entrevista feita com o professor de matemática Londero (2014). Na Figura 50 é mostrada a explicação do professor sobre os planos coordenados.

Figura 50 – Planos Coordenados



Fonte: Londero (2014).