UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISUALIZADOR DE ANIMAÇÃO 3D COM SUPORTE AO FORMATO DE ARQUIVOS FBX EM OBJECTIVE-C++

FELIPE SILVINO PEREIRA

BLUMENAU
2010

2010/1-10

FELIPE SILVINO PEREIRA

VISUALIZADOR DE ANIMAÇÃO 3D COM SUPORTE AO FORMATO DE ARQUIVOS FBX EM OBJECTIVE-C++

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

Prof. Paulo César Rodacki Gomes, Dr. - Orientador

BLUMENAU
2010

2010/1-10

VISUALIZADOR DE ANIMAÇÃO 3D COM SUPORTE AO FORMATO FBX EM OBJECTIVE-C++

Por

FELIPE SILVINO PEREIRA

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

Presidente:	Prof. Paulo César Rodacki Gomes , Dr. – Orientador, FURB
Membro:	Prof. Dalton Solano dos Reis, Ms. – FURB
Membro:	Prof. Mauro Marcelo Mattos, Dr. – FURB

Blumenau, 2 de julho de 2010

Dedico este trabalho os amigos, a família, ao meu orientador e a todos aqueles que me ajudaram direta ou indiretamente nesta passagem de mais uma etapa de minha vida.

AGRADECIMENTOS

Aos amigos que entenderam o meu não comparecimento nas atividades festivas devido ao desenvolvimento deste.

As pessoas especiais presentes na minha vida, das quais uma, está sempre presente nos meus pensamentos.

A Música, sempre presente em todos estes momentos.

Ao professor Ms. Dalton Solano dos Reis que sempre me apoiou no início da proposta deste trabalho.

Ao meu orientador, Dr. Paulo César Rodacki Gomes, por ter acreditado na conclusão deste trabalho.

Águas mansas não fazem bons marinheiros.

Provérbio Indiano

RESUMO

Este trabalho descreve o desenvolvimento de um protótipo para visualização de arquivos no formato FBX na plataforma iOS 4 implementado na linguagem Objective-C++, denominado iSceneViewer. O protótipo é baseado na biblioteca OpenGL ES para construção das formas geométricas representadas pela estrutura de dados importada de um arquivo FBX. Esta estrutura de dados está presente no SDK fornecido pela Autodesk que é utilizado pelo protótipo para fazer a interpretação do arquivo FBX. No protótipo, além da visualização do modelo 3D, é possível visualizar as animações e posições de câmeras que foram definidas na confecção da cena na ferramenta usada para geração do arquivo importado. Como limitações, o protótipo não possui suporte a texturas e iluminações que podem estar presentes na cena representada pelo arquivo FBX importado.

Palavras-chave: iPhone. FBX. OpenGL ES.

ABSTRACT

This work describes the development of a prototype for viewing FBX file format on the iOS 4 plataform implemented in Objective-C++ language, called iSceneViewer. The prototype is based in the OpenGL ES library for the construction of the geometric shapes represented by the data structure that was imported from a FBX file. This data structure is present in the SDK provided by Autodesk that is used by the prototype to do the interpretation of the FBX file. In the prototype, in addition to viewing the 3D model, is possible view animations and camera positions that were defined in the confection of scene on tool used to generate the imported file. As limitations, the prototype does not support the textures and lighting that may be present in the scene represented by the FBX file imported.

Key-words: iPhone. iOS 4. FBX. OpenGL ES.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

3D – Três Dimensões

ANSI – *American National Standards Institute*

API – *Application Programming Interface*

ARM – *Advanced Risc Machine*

ASCII – *American Standard Code for Information Interchange*

CPU – *Central Processing Unit*

EGL – *Embedded Graphics Library*

FBX – *FilmBoX*

GLU – *OpenGL Utility Library*

GPU – *Graphics Processing Unit*

OpenGL ES – *Open Graphics Library to Embedded Systems*

RF – Requisito Funcional

RNF – Requisito Não-Funcional

SDK – *Software Development Kit*

PDAs – *Personal Digital Assistants*

PSP – *PlayStation Portabel*

RGBA – *Red Green Blue Alpha*

SDK – *Software Development Kit*

XML – *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Segundo Economiste.com (2008), as tecnologias móveis estão se consolidando. Pode-se dizer que é a febre do momento, pois seu mercado cresce a cada dia e novos aparelhos são lançados já com longas filas de espera. Além disso, o acesso a internet sem custo em lojas, restaurantes e até em parques de fim de semana já é uma realidade e levou milhares de pessoas a adquirirem aparelhos que lhes permitam desfrutar destas novidades.

A capacidade de processamento e armazenamento destes novos dispositivos móveis também é cada vez maior e abriu um leque de opções para quem tem interesse no desenvolvimento de softwares para celulares, principalmente na linha de softwares de entretenimento como jogos, onde estes fatores são cruciais. Novas empresas especializadas na produção deste tipo de software estão surgindo e já se percebe a necessidade de ferramentas para facilitar e auxiliar os desenvolvedores que hoje ainda enfrentam muitos problemas para cumprir seus prazos, tendo em vista a maior complexidade no desenvolvimento de jogos para dispositivos móveis.

Seguindo esta tendência, conforme Area (2010), a Autodesk disponibilizou um novo formato de arquivos chamado *FilmBoX* (FBX) para integração dos modelos 3D desenhados e animados nos seus softwares proprietários, onde o objetivo é facilitar o uso destes modelos em jogos 3D ou mesmo em outras ferramentas de modelagem.

Diante do exposto, este trabalho pretende estudar e pesquisar a plataforma iOS 4 bem como a linguagem Objetive C++ e o formato de arquivo FBX que é capaz armazenar informações sobre modelos 3D já animados. Busca também reunir as especificações, padrões e recursos que fazem parte do pacote de desenvolvimento de software fornecido pela Autodesk. Do mesmo modo, interfaces de programação de dispositivos disponibilizadas pela Apple incorporadas ao iOS 4 serão empregadas para portar ao ambiente móvel a possibilidade de visualização dos modelos e animações 3D importadas de qualquer arquivo gerado no formato FBX (Versão 6.0). O protótipo suporta tanto o tipo de arquivo FBX no formato binário como o tipo de arquivo FBX no formato *American Standard Code for Information Interchange* (ASCII).

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo de software que possibilite a visualização no dispositivo iPhone de modelos 3D animados provenientes de arquivos no formato FBX.

Os objetivos específicos do trabalho são:

a) fazer a leitura de modelos 3D animados no formato de arquivo FBX;
b) visualizar o modelo no iPhone com o uso da *Open Graphics Library to Embedded Systems* (OpenGL ES) e da linguagem Objective-C++;
c) visualização das animações definidas no arquivo lido.

1.2 ESTRUTURA DO TRABALHO

Este trabalho é formado por quatro capítulos. O capítulo 2 apresenta as tecnologias envolvidas, os detalhes do formato de arquivos FBX e também apresenta os trabalhos correlatos. No capítulo 3 é abordada a especificação do protótipo do visualizador 3D, sua operacionalidade e principais funções. O capítulo 4 apresenta as conclusões gerais e possíveis futuras implementações.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes são detalhados a linguagem de programação e o sistema operacional móvel em questão, explorando as características da plataforma iOS 4. Em seguida é feita também uma breve explicação da OpenGL ES e sua utilização com a linguagem Objective-C++. Por fim, são explanadas as principais características do arquivo FBX bem como o *Software Development Kit* (SDK) disponibilizado pela Autodesk.

2.1 LIGUAGEM OBJECTIVE C

Segundo Tenon Intersystems (2010), o Objective-C foi criado principalmente por Brad Cox e Tom Love no início da década de 1980 na empresa deles. A idéia era construir uma linguagem para dar suporte a objetos de modo flexível, possuir um conjunto de bibliotecas funcionais e permitir que fossem empacotados num único formato multiplataforma. Em 1988 Steve Jobs licenciou o Objective-C e liberou sua própria versão do compilador e das bibliotecas da linguagem.

O Objective-C é uma camada sofisticada construída sobre a linguagem C padrão *American National Standards Institute* (ANSI). Nesta camada é adicionado o suporte a programação orientada a objetos, onde a implementação deste paradigma tem sintaxe muito parecida com a sintaxe da linguagem Smalltalk. A linguagem também adiciona o suporte à programação orientada a reflexão, característica que permite que o programador tenha acesso aos nomes dos métodos, classes, atributos e outras informações dos objetos manipulados e inclusive fazer chamadas dinâmicas a eles em tempo de execução. Estas chamadas também são conhecidas como acesso "por nome" (TENON INTERSYSTEMS, 2010).

Uma importante característica herdada do C é a velocidade. Uma linha de comando ou troca de mensagem em Objective-C consome apenas entre 1.5 a 2.0 vezes mais tempo de um programa escrito em C, o que torna esta linguagem forte candidata na definição da arquitetura de programas onde o tempo de resposta é crucial, tais como computação gráfica, desenvolvimento de jogos e outros (ASTRO, 2005).

2.1.1 Objective-C++

Segundo Developer (2010), esse é nome utilizado pela Apple para a linguagem hibrida suportada pelo seu compilador de Objective-C oficial. Esta linguagem permite que seus desenvolvedores utilizem uma união entre códigos escritos nas linguagens C, Objective-C e C++ em seus projetos, bem como a utilização de bibliotecas escritas em qualquer uma destas linguagens.

O compilador identifica pela extensão do arquivo fonte, qual o tipo de código que esta compilando, e assim altera seu comportamento conforme a necessidade. Ou seja, arquivos fonte terminados em .c são interpretados como código C padrão ANSI, já os terminados em .m são interpretados como código Objective-C e por fim, os arquivos terminado em .mm, são identificados como código C++.

Porem existe algumas limitações nesse recurso, como por exemplo, o Objective-C++ não adiciona características das classes do C++ nas classes do Objective-C, nem adiciona recursos Objective-C nas classes C++. Não é possível usar a sintaxe Objective-C para chamar os métodos de um objeto C++ e nem utilizar a herança entre classes escritas em Objective-C com outras escritas em C++, pois as hierarquias de classes devem ser separadas. Ou seja, uma classe C++ não pode herdar de uma classe de Objective-C, e vice-versa (DEVELOPER, 2010).

2.2 SMALLTALK

Segundo UFSC (2010), a primeira linguagem a incorporar facilidades para definir classes de objetos genéricos na forma de uma hierarquia de classes e sub-classes foi a linguagem Simula, idealizada em 1966 na Noruega, como uma extensão da linguagem ALGOL 60.

A linguagem Smalltalk veio logo em seguida, fortemente baseada na linguagem Simula, foi desenvolvida no Centro de Pesquisas da Xerox durante a década de 70, e incorporou o princípio de objetos ativos, prontos a reagir a mensagens que ativam comportamentos específicos do objeto. Ou seja, os objetos em Smalltalk deixaram de ser

meros dados manipulados por programas, e passaram a ser encarados como processadores idealizados individuais e independentes, aos quais podem ser transmitidos comandos em forma de mensagens. Estava modelada a essência da programação orientada a objetos atual.

A sintaxe do Smalltalk é um pouco diferente das tradicionais linguagens orientadas a objetos utilizadas atualmente, porem o princípio de chamada de métodos de um objeto pela instrução <objeto recebedor><mensagem> é mantido (UFSC, 2010).

Na Figura 1, é apresentado o envio das mensagens `clear` e `show` para o objeto `Transcript` em Smalltalk e em Java. O tradicional ponto entre o objeto e a mensagem e os parênteses delimitando os parâmetros da mensagem não existem no Smalltalk.

```
//Sintaxe Smalltalk
Transcript clear.
Transcript show:'Olá Mundo'.
-----------------------------------------------
//Sintaxe Java
Transcript.clear();
Transcript.show("Olá Mundo");
```

Fonte: Adaptado de UFSC (2010).

Figura 1 – Diferenças entre a sintaxe do Smalltalk e do Java

2.3 DISPOSITIVO IPHONE 4

O iPhone 4 é um dispositivo desenhado e fabricado pela Apple Computer que combina telefone celular, *player* de música, vídeo digital e acesso sem fio à internet. Mede cerca de 5,8 centímetros de largura, por 11,5 centímetros de altura, sendo a espessura do aparelho inferior a 1 centímetro, pesa aproximadamente 140 gramas. Sua primeira versão foi lançada nos EUA em 29 de junho de 2007, impressionou a todos com sua interface *multi touch* intuitiva, navegador de internet prático e realmente usual. Na época isso era a grande crítica dos usuários de *smartphones* existentes até então. Porém, a evolução continuou e atualmente já é encontrado a venda a versão 4 do aparelho com até 32 Gb de memória de armazenamento, câmera de 5 *megapixel*, GPS integrado e *Bluetooth*. Atualmente milhares de aplicações dos mais variados tipos, pagas ou gratuitas, são disponibilizadas para *download* na loja virtual da própria Apple que é conhecida como *App Store* (APPLE, 2010).

O iPhone 4 dispõe de um hardware poderoso, onde seu processador é o mesmo que

está presente no dispositivo iPad, e que é denominado de Apple A4. Este processador desenvolvido pela Apple e fabricado pela Samsung utiliza uma abordagem conhecida como *System-On-a-Chip* (SOC), que combina a *Central Processing Unit* (CPU) e a *Graphics Processing Unit* (GPU) um único chip. O Objetivo desta abordagem é obter maior eficiência no processamento e menor perda de tempo nos barramentos, devido à proximidade física das unidades envolvidas. Esta CPU é baseada na arquitetura *Advanced Risc Machine* (ARM) e roda a velocidade aproxima a de 1Ghz (APPLE, 2010).

2.4 SISTEMA OPERACIONAL IOS 4

Segundo Roughlydrafted Magazine (2007), o iOS 4 é uma versão otimizada do Mac OS X, sistema operacional padrão dos computadores Macintosh da Apple Computer. Entre as principais diferenças, pode-se citar que a versão iPhone OS roda sobre um *Hardware* de arquitetura *Advanced Risc Machine* (ARM), contra as arquiteturas x86 e PowerPC ISA na versão Mac OS X. Não existe opção de múltiplos usuários e os mecanismos de acesso e segurança dos arquivos são mais limitados.

Outra diferença importante é que o iOS 4 roda a partir de uma imagem do disco direto da memória *flash* não acessando disco físico diretamente. Isso permite simplificar a restauração do sistema operacional. Esta arquitetura é conhecida pelo termo "*Run from Flash RAM*". Para apresentação dos gráficos em 3D o iPhone OS, no *Software Development Kit* (SDK) disponibilizado pela Apple, utiliza-se da biblioteca OpenGL ES 1.1 que trabalha sobre a placa gráfica PowerVR 3D do iPhone (ROUGHLYDRAFTED MAGAZINE, 2007).

2.5 FORMATO DE ARQUIVO FBX

Segundo Area (2010), o formato FBX desenvolvido pela Autodesk vem sendo aperfeiçoado há alguns anos. Este formato de arquivo tende a tornar-se cada vez mais versátil e universal dado que, sendo de código aberto, permite a adoção por outras marcas de software que integram nas suas plataformas. O FBX permite assim aos criadores de conteúdos 3D um favorecimento da colaboração em nível de arquivos de dados de geometria, uma vez que

podem utilizar diferentes aplicações, partilhando e integrando as suas criações sem limites de compatibilidade.

Este formato permite guardar um grande leque de definições relacionadas com um projeto 3D, onde além da geometria e mapeamento de texturas, é possível incluir também informações referentes a posições de câmeras, animações de objetos, iluminação, estruturas de esqueletos para modelos humanóides e suporte a animação dos modelos (AREA, 2010).

Para manipulação do arquivo, a própria Autodesk disponibiliza para *download* um SDK para os sistemas operacionais Windows, Mac OS X, Linux e Irix. Este kit gratuito de desenvolvimento de software permite que fornecedores de software e hardware adicionem o suporte para FBX em seus produtos de forma simples. (AREA, 2010).

Internamente, o arquivo FBX é divido em várias pequenas estruturas, ou blocos, que trazem as propriedades e valores atribuídos para cada objeto presente na cena 3D. Geralmente o arquivo FBX é gerado de forma binária pelos aplicativos de modelagem e animação 3D, porém estes arquivos binários podem ser convertidos para o formato de arquivo padrão *American Standard Code for Information Interchange* (ASCII) e se for aberto em um aplicativo como o bloco de notas, por exemplo, é possível identificar estas estruturas e blocos, presentes no *layout* do arquivo (AREA, 2010).

Em maiores detalhes sobre a estrutura do arquivo FBX, conforme Naql (2008), o grafo de cena é representado como uma árvore, onde o nó pai tem vários filhos e estes filhos geralmente não possuem outros filhos, ou seja, são nós folha. Esta árvore também não segue uma hierarquia lógica muito comum em representações de grafos de cena. No caso do FBX as informações de um objeto ou modelo ficam espalhadas por vários nós que não necessariamente são filhos uns dos outros e é desta forma que os criadores do formato buscaram separar informações de posição de vértices, luzes e faces das informações de animação, texturas e som.

Na Figura 2 abaixo é esboçada uma possível representação formal da estrutura de arquivo FBX.

```
FBXFile ::= (Objects | Connections | Takes) + <EFO>
Objects ::= "Objects:" "{" (Model) + "}"
Connections ::= "Connections:" "{" (Connect) + "}"
Takes ::= ""Model:" <NAME> ("," <NAME> ) ? "{" (Type | Vertices |
Edges | PolygonVertexIndex | Points) + "}"
Connect ::= "Connect:" <NAME> "," <NAME> ", " <NAME> ("," <NAME> )?
```

Fonte: adaptado de Naql (2008).

Figura 2 – Representação formal do arquivo FBX

Na Figura 3 abaixo é apresentado parte de um arquivo FBX, representando um pequeno grafo de cena contendo um modelo recebendo iluminação gerada por um objeto de luz e preparado para animação, realizando translações no eixo X.

```
...
; Propriedade do objeto
Objects: {
  Model: "Model::Light", "Light" {
    Version: 232
    Properties60:  {
      Property: "Lcl Translation", "Lcl Translation", "A+",11,0,0
      Property: "Lcl Rotation", "Lcl Rotation", "A+",0,0,0
    }
    NodeAttributeName: "NodeAttribute::Light"
  }
  Model: "Model::Mesh", "Mesh" {
    Version: 232
    Properties60:  {
      Property: "Lcl Translation", "Lcl Translation", "A+",0,0,0
      Property: "Lcl Rotation", "Lcl Rotation", "A+",0,0,0
      Property: "Lcl Scaling", "Lcl Scaling", "A+",1,1,1
    }
  }
}
...
```

Fonte: adaptado de Naql (2008).

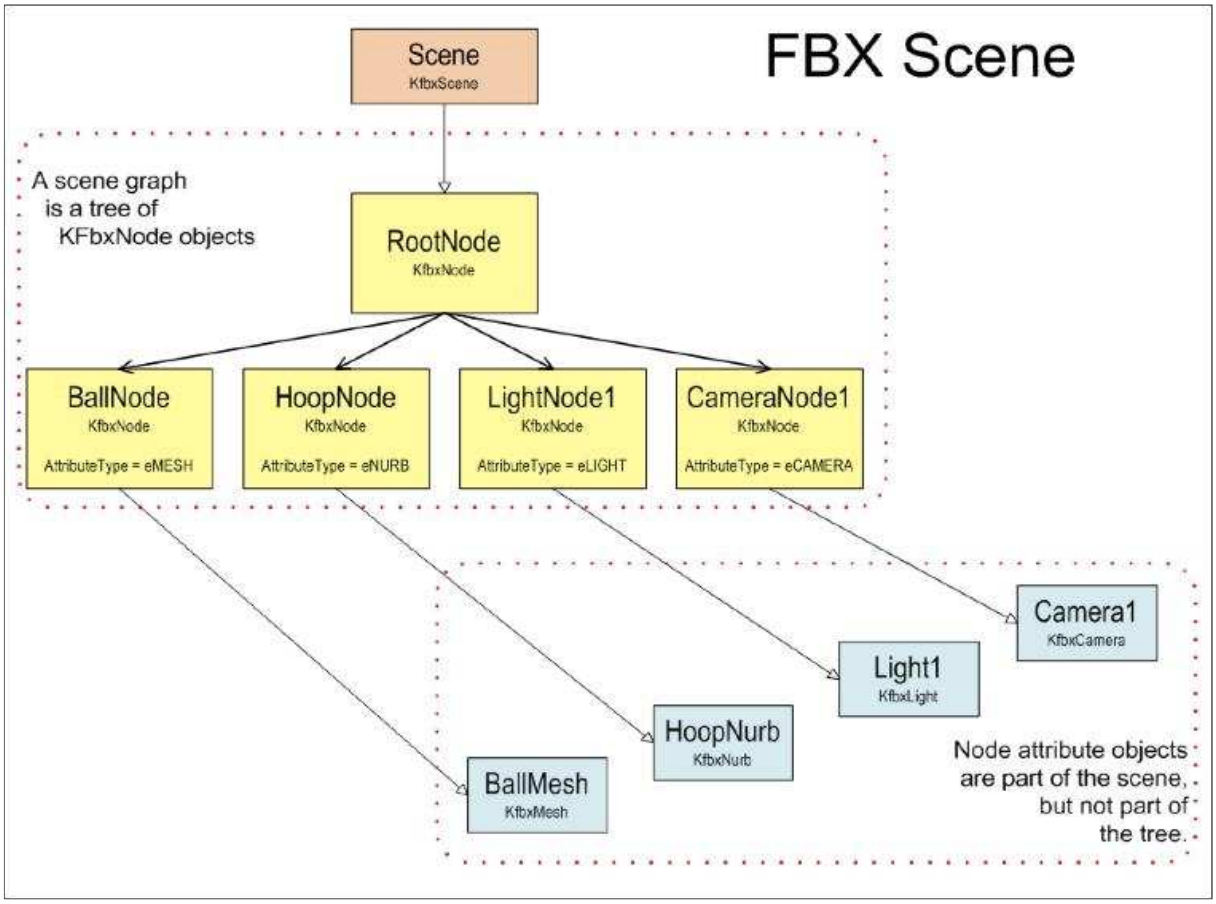
Figura 3 – Representação do Arquivo FBX

2.6 FBX SOFTWARE DEVELOPMENT KIT (SDK)

Segundo AUTODESK (2009), o FBX SDK foi projetado e desenvolvido pela Autodesk, que hoje é a proprietária da tecnologia FBX, e é fornecido gratuitamente para *download* no *site* da mesma. O objetivo do SDK é proporcionar aos desenvolvedores de *softwares* a possibilidade de portar essa tecnologia em seus aplicativos, pois ele não só

permite a interpretação de um arquivo FBX, mas também permite alteração da estrutura de dados e geração de um arquivo editado ou mesmo de um arquivo totalmente novo.

Na Figura 4 é apresentado o diagrama de classes do grafo de cena disponibilizado pelo SDK e que é carregado ao fazer a importação de um arquivo FBX. A classe `KfbxScene` possui uma referência para o nó raiz do grafo, que é um objeto da classe `KfbxNode`, assim como todos os nós filhos. Esses nós possuem o atributo `AtributeType` que identifica qual estrutura de dados o nó armazena e também uma referência para um objeto da classe `KfbxNodeAttritute` que é super classe de todos os tipos de estruturas que um nó pode representar.



Fonte: Adaptado de sdkdoc(2010).

Figura 4 – Grafo de cena do FBX SDK

2.7 OPENGL ES

A biblioteca gráfica OpenGL ES é uma versão da API OpenGL para sistemas embarcados e segundo Khronos (2009), a OpenGL ES 1.0 é baseada na versão 1.3 da OpenGL

original, enquanto a versão 1.1 é baseada em OpenGL 1.5. Atualmente, o grupo Khronos é o responsável pelas especificações da OpenGL ES. Devido à grande variedade de dispositivos móveis disponíveis que poderiam utilizar a OpenGL ES, foi definido um conceito denominado de *profile* (perfil). Um perfil define um subconjunto de funcionalidades de alguma versão completa da OpenGL, acrescido de propriedades específicas para a versão embarcada, como extensões. Alguns perfis existentes são:

a) *Common Profile*: perfil destinado a dispositivos de entretenimento em massa como telefones celulares, *Personal Digital Assistants* (PDAs) e consoles, entre outros;
b) *Common-Lite Profile*: perfil destinado a dispositivos em que a confiabilidade e segurança são os fatores mais importantes da aplicação. Desta forma, define um subconjunto mínimo necessário para que seja viável construir este tipo de aplicação.

Assim como em todas as versões da OpenGL, a versão embarcada também define uma camada que é específica ao sistema onde é executada. Esta camada é definida como *Embedded Graphics Library* (EGL) e define uma API comum para ser usada com OpenGL ES. A implementação desta camada, entretanto, depende do fabricante do hardware.

Algumas diferenças, características e cuidados que devem ser tomados ao fazer o uso da OpenGL ES em relação a OpenGL original são:

a) as variações de funções que aceitam vários parâmetros, de vários tipos não existem na OpenGL ES, à exceção de algumas poucas que foram mantidas;
b) somente o modo de cores *Red Green Blue Alpha* (RGBA) pode ser usado;
c) o único modo de desenho disponível para as primitivas é o sólido, ou seja, os modos de desenho em *wireframe* e pontos não podem ser usados;
d) é recomendado que seja usado o modo em tela cheia (*fullscreen*) em aplicações OpenGL ES. Caso isso não seja feito, pode acontecer de a aplicação trabalhar apenas com meia tela (no telefone), ou problema parecido;
e) não existe a biblioteca *OpenGL Utility Library* (GLU). Entretanto, é possível encontrar na internet implementações das funções disponíveis na GLU original, convertidas para OpenGL ES.

2.8 TRABALHOS CORRELATOS

Com relação a trabalhos correlatos, foram escolhidos cinco projetos que apresentam algumas funcionalidades e características semelhantes a do protótipo desenvolvido neste trabalho. Para tal, foi levado em consideração tanto a compatibilidade com a tecnologia FBX como a utilização da plataforma iPhone OS.

Os projetos selecionados e que serão detalhados logo abaixo nesta seção, são os seguintes: iPhone FBX Viewer (SCRIPTLANCE, 2010), FBX for QuickTime (AUTODESK, 2009), FBX Converter (AUTODESK, 2009), “MJ3I: Um motor de jogos 3D para iPhone OS” (TAKANO, 2009) e o “Reconstrutor de modelos 3D utilizando técnicas de detalhamento” (PISKE, 2008).

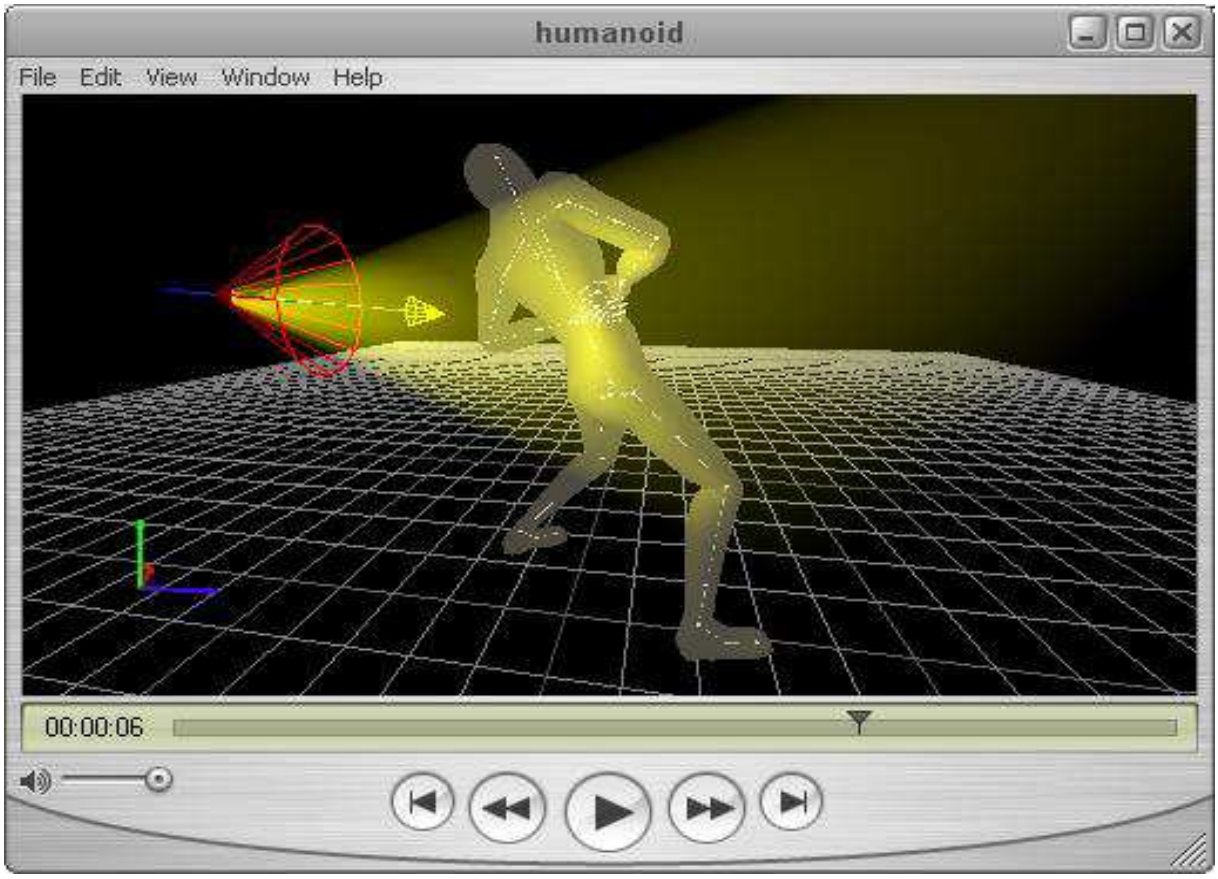
2.8.1 iPhone FBX Viewer

O iPhone Fbx Viewer é um projeto de software que ainda está em desenvolvimento e que também tem como objetivo importar e apresentar modelos 3D animados a partir de um arquivo FBX no iPhone. Detalhes do projeto ainda não foram publicados (SCRIPTLANCE, 2010).

2.8.2 FBX for QuickTime

O FBX for QuickTime é um *plugin* para o QuickTime disponível para plataformas Mac OS X e Windows que, além de visualização, permite maior interação com o modelo importado a partir de um arquivo FBX. A implementação deste *plugin* foi feita utilizando o SDK disponibilizado pela Autodesk para manipulação de arquivo no formato FBX (AREA, 2009).

A Figura 5 apresenta a visualização de um arquivo FBX utilizando o QuickTime com o *plugin* instalado.



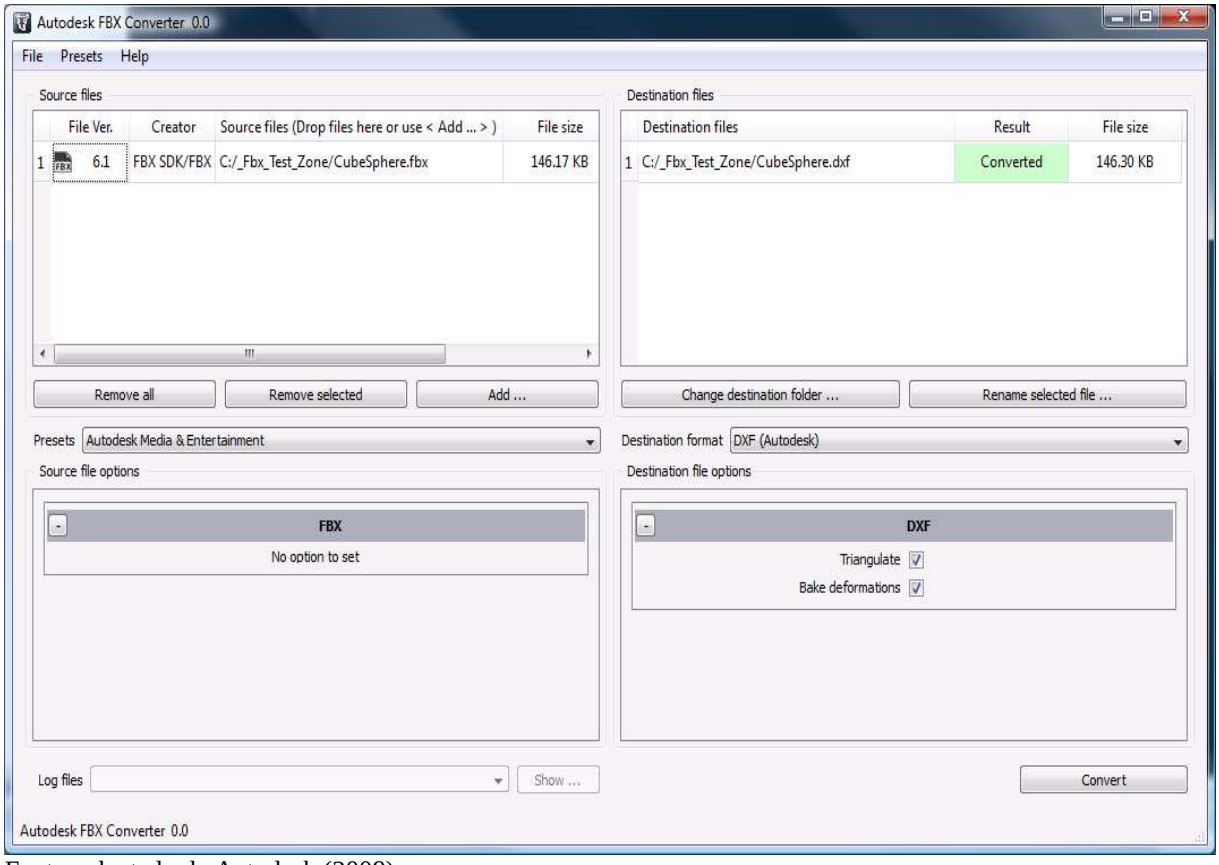
Fonte: adaptado de Autodek (2009).

Figura 5 – Exemplo de utilização do *plugin* FBX for QuickTimer

2.8.3 FBX Converter

O FBX Converter é outro software disponibilizado pela Autodesk para *download* que permite a conversão do formado FBX em outros formatos utilizados pelos demais *softwares* proprietários da Autodesk. Dentre os formatos suportados destaca-se a conversão do FBX para o formato 3DS, formato do 3D Studio Max, e também para o formato *wavefront 3d OBJect* (OBJ) que é um formato de arquivo aberto, mais primitivo que o FBX, porém é reconhecido por diversas outras ferramentas disponíveis hoje no mercado (AREA, 2009).

A Figura 6 apresenta a interface principal do produto FBX Converter na versão para a plataforma Windows.



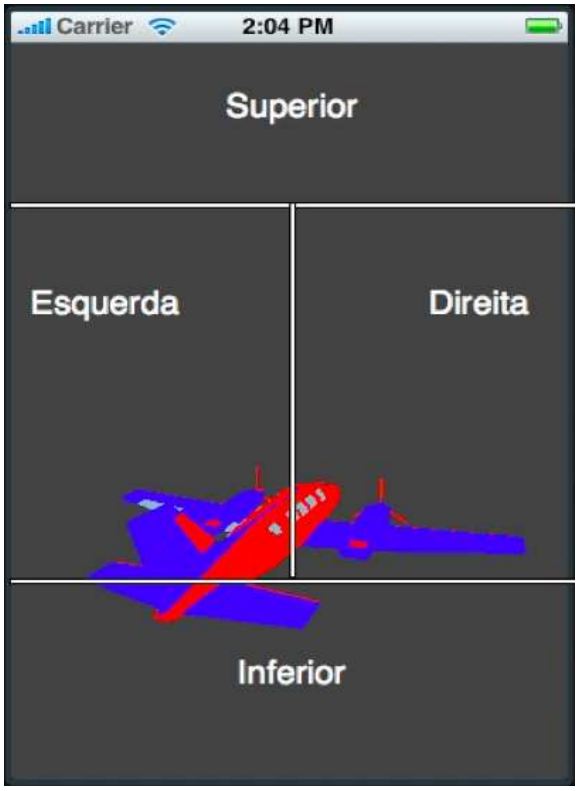
Fonte: adaptado de Autodesk (2009).

Figura 6 – Interface principal do FBX Converter

2.8.4 MJ3I: Um motor de jogos 3D para iPhone OS

Esse motor é baseado na biblioteca OpenGL ES, para construção de formas geométricas e nas bibliotecas do SDK da plataforma iPhone para realização da interação homem máquina (IHC), através de uma tela que permite múltiplos gestos. O motor permite que sejam desenvolvido jogos, sem a necessidade de implementar todos as rotinas básicas e necessárias de um jogo 3D. O motor não implementa rotinas mais complexas como cálculo de física, sistema de partículas ou mesmo o áudio.

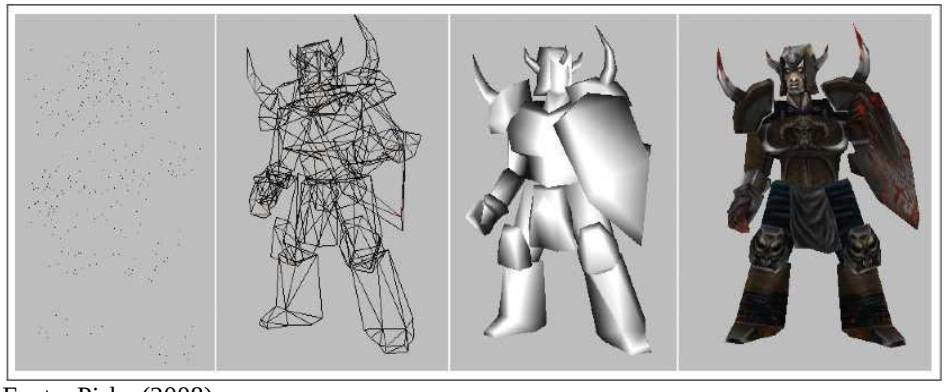
A Figura 7 demonstra um protótipo que utiliza o motor MJ3I para fazer a renderização de um modelo exemplificando um possível jogo 3D.



Fonte: Takano (2010).
Figura 7 – Exemplo de utilização do motor MJ3I

2.8.5 Reconstrutor de modelos 3D utilizando técnicas de detalhamento

Esse projeto é focado para plataforma móvel com suporte a importação de um formato de arquivo que representa modelos 3D, no caso o formato suportado é o *quake2 MoDel format* (MD2). Esta ferramenta, além da importação e visualização do modelo 3D no dispositivo móvel, utiliza técnicas da computação gráfica conhecida como *Level Of Detail* (LOD). O objetivo desta técnica é fazer uso de versões simplificadas de um objeto, assim reduzindo a malha de polígonos e textura de um modelo conforme Figura 8 (PISKE, 2008).



Fonte: Piske (2008).
Figura 8 – Opções de visualizações disponibilizadas pela ferramenta

3 DESENVOLVIMENTO DO PROTÓTIPO

Este capítulo detalha as etapas do desenvolvimento do protótipo. São apresentados os requisitos, a especificação e a implementação do mesmo, mencionando as técnicas e tecnologias utilizadas. Também são comentadas questões referentes à operacionalidade do protótipo e os resultados obtidos.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos apresentados encontram-se classificados em Requisito Funcional (RF) e Requisito Não-Funcional (RNF), os quais são:

a) importar o arquivo FBX com as geometrias dos modelos 3D nele definidos (RF);
b) apresentar os quadros de animação (*keyframes*) definidos no arquivo (RF);
c) permitir aproximação e rotação do modelo importado (RF);
d) permitir iniciar, parar e reiniciar as animações importadas do modelo (RF);
e) exibir os esqueletos (*bones*) definidos no arquivo (RF);
f) exibir posição da câmera e direcionar visão conforme definido no arquivo (RF);
g) ser implementado na plataforma iOS 4 (RNF);
h) ser implementado usando o paradigma da orientação a objetos (RNF).

3.2 ESPECIFICAÇÃO

A especificação do protótipo é apresentada através dos diagramas de casos de uso e de classes, os quais foram confeccionados utilizando a ferramenta *Enterprise Architect*.

3.2.1 Protótipo `iSceneViewer`

O `iSceneViewer` é um protótipo que demonstra a utilização do SDK disponibilizado

pela Autodesk para interpretações contidas em um arquivo FBX, utilizando o Objective-C++ na plataforma iOS 4. Baseado no OpenGL ES para realizar a renderização da cena importada, o protótipo também utiliza a biblioteca UIKit, disponibilizada pela própria Apple, que integrada com o iOS 4 torna possível capturar os toques na tela.

3.2.2 Diagrama de caso de uso

A seguir são apresentados o três diagramas de casos de uso: `Importação Arquivo`, `Animação` e `Modos de Câmera`, com seus respectivos casos de uso, que apresentam as funcionalidades da ferramenta.

O primeiro diagrama, designado `Importação Arquivo` (Figura 9) apresenta os Casos de Uso (UC) de seleção do arquivo e de quais das estruturas presentes nele serão visualizadas pelo usuário. A execução deste caso de uso é necessária para que seja possível acessar as outras funcionalidades do protótipo.

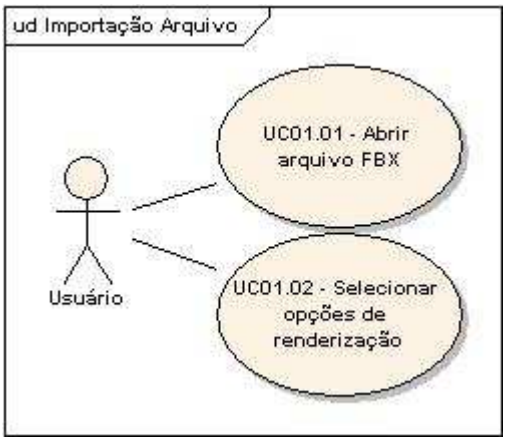


Figura 9 – Diagrama de casos de uso: `Importação Arquivo`

Na Figura 10 são apresentados os cenários de cada caso de uso sendo que, cada caso de uso possui um cenário principal e cenário de exceção se existir.

CASO DE USO	CENÁRIOS
UC01 .01 - Abrir arquivo FBX	**Selecionar e abrir arquivo (Principal)** 01 - O usuário seleciona a opção "Open File" no menu principal. 02 - O protótipo apresenta tela para escolha do arquivo. 03 - O usuário seleciona o arquivo e confirma a abertura. 04 - O protótipo carrega a cena presente no arquivo. **Arquivo corrompido (Exceção)** No passo 03, o arquivo selecionado pode estar corrompido ou ser de uma versão de arquivo FBX não suportada pelo propótipo. O propótipo apresentará mensagem detalhada.
UC01.02 - Selecionar opções de renderização	**Opções de visualização (Principal)** 01 - Usuário seleciona a opção "Render" no menu principal. 02 - O protótipo apresenta uma listas das estruturas importadas, o usuário pode selecionar quais deseja visualizar.

Figura 10 – Descrição dos casos de uso do diagrama: `Importação Arquivo`

Na Figura 11, é apresentado o diagrama de casos de uso `Animação`, que relaciona os casos de uso referentes às funcionalidades para manipulação das animações importadas do arquivo FBX.

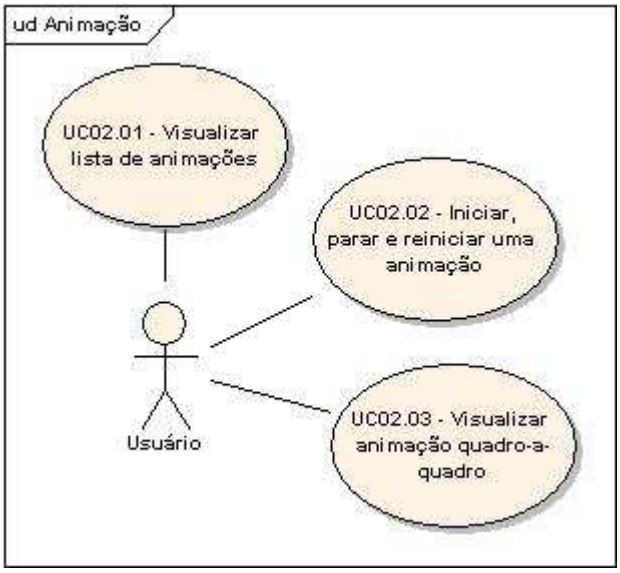


Figura 11 – Diagrama de caso de uso: `Animação`

A Figura 12 apresenta os cenários de cada caso de uso no diagrama de casos de uso: `Animação`.

CASO DE USO	CENÁRIOS
UC02.01 - Visualizar lista de animações	**Selecionar animação (Principal)** 01 - O usuário seleciona a opção "Animation Stack" no menu principal. 02 - O protótipo apresenta uma tela com a lista de animações importadas. 03 - O usuário seleciona uma animação e confirma. 04 - O protótipo inicia imediatamente a animação escolhida. **Lista vazia (Exceção)** No passo 02, pode não haver animações no arquivo FBX importado, o protótipo apresentará uma mensagem detalhada.
UC02.02 - Iniciar, parar e reiniciar uma animação	**Manipular animação (Principal)** 01 - Usuário seleciona entre as opção "Play", "Stop" e "Restart" no menu auxiliar. 02 - O protótipo inicia, para e reinicia a animação atual seleciona, respectivamente.
UC02.03 - Visualizar animação quadro-a-quadro	**Quadro-a-Quadro (Principal)** 01 - Usuário seleciona a opção "Frame-To-Frame" no menu auxiliar. 02 - O protótipo congela animação no quadro atual. 03 - A cada toque do usuário na tela na área de visualização, o protótipo avança para o próximo quadro e reiniciará ao primeiro quadro ao concluir a pilha de quadros da animação corrente.

Figura 12 – Descrição dos casos de uso do diagrama: Animação

Para concluir a apresentação dos casos de uso implementados no protótipo, a Figura 13 demonstra o último diagrama de casos de uso chamado Modos de Câmera, que relaciona os casos de usos referentes às funcionalidades para manipulação da câmera no ambiente 3D e na cena importada.

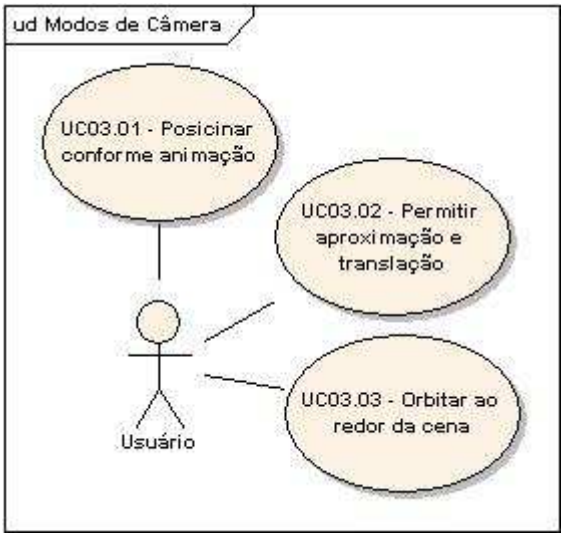


Figura 13 – Diagrama de casos de uso: `Modos de Câmera`

Na Figura 14 são apresentados os cenários de cada caso de uso presente no diagrama de casos de uso `Modos de Câmera`.

CASO DE USO	CENÁRIOS
UC03.01 - Posicionar conforme animação	**Posicionar câmera (Principal)** 01 - O usuário seleciona a opção "Camera" no menu principal, em seguida a opção "Animation". 02 - O protótipo passa a apresentar a cena posicionando a câmera coforme foi definido na confecção do arquivo importado. **Câmera não definida (Exceção)** No passo 01, pode não haver uma câmera definida no arquivo, o protótipo apresentará uma mensagem detalhada.
UC03.02 - Permitir aproximação e translação	**Translação da câmera (Principal)** 01 - Usuário seleciona a opção "Camera" no menu principal, em seguida a opção "Zoom \ Pan". 02 - Ao tocar com 1 dedo na tela de visualização o protótipo faz a translação da câmera conforme o movimento do dedo do usuário sobre a tela. 03 - Ao tocar com 2 dedos na tela simultâneamente o protótipo aproxima ou distância a câmera do ponto central da cena conforme os movimentos.
UC03.03 - Orbitar ao redor da cena	**Orbitar câmera (Principal)** 01 - Usuário seleciona a opção "Camera" no menu principal, em seguida a opção "Orbit". 02 - Ao tocar com 1 dedo na tela, o protótipo faz a translação e rotação da câmera orbitando ao redor do ponto central da cena.

Figura 14 – Descrição dos casos de uso do diagrama: `Modos de Câmera`

3.2.3 Diagrama de Classes

Esta seção apresenta o diagrama de classes do protótipo `iSceneViewer` (Figura 15). Para melhor visualização e entendimento os métodos e atributos privados de todas as classes foram omitidos, a exceção das classes `MyGlut` e `EAGLView`, onde também foram omitidos os métodos públicos.

No diagrama é possível observar que a classe `ES1Renderer` é a principal classe do protótipo e que possui referências para demais classes responsáveis pelas tarefas específicas para a visualização do modelo, movimentação da câmera e animação. A classe `DrawScene` é a classe responsável por fazer a leitura do grafo de cena e chamar os métodos da classe `Draw` conforme o tipo de nodo encontrado no grafo. Esta classe `Draw` por sua vez, faz as chamadas as rotinas da OpenGL ES para desenha os componentes da cena no *viewport*.

A classe `Camera` é a classe responsável por posicionar o ponto e a direção da visão do usuário na cena, bem como responder aos eventos de toque na tela do dispositivo iPhone, conforme o tipo de movimentação de câmera escolhido. Os eventos chegam até a classe `Camera` repassados pela classe `ES1Renderer` que recebe estas as informações de toques já interpretadas pela classe `EAGLView.` A classe `EAGLView` é interface gráfica do protótipo e que recebe os eventos do *menu* principal, repassando-os aos objetos responsáveis pelas operações de cada opção do *menu*. As classes `InitFBXLibrary` e `InitScene` fazem a inicialização do FBX SDK ao fazer a leitura do arquivo FBX a ser visualizado e também faz a inicialização das variáveis locais para posição de câmera e quadro atual da cena, respectivamente.

A classe `MyGLUT` faz a implementação de duas funções da biblioteca GLUT utilizadas pelo o protótipo, pois a GLUT não existe no *framework* da OpenGL ES fornecido pelo iOS 4 SDK. Isto não é uma limitação deste *framework*, mas sim, devido ao fato da OpenGL ES não possuir a biblioteca GLUT, como ocorre na sua versão OpenGL tradicional.

Os nomes de métodos, classes, parâmetros e atributos são todos em inglês para manter o padrão em relação às classes do iOS 4 SDK e também por motivos de preferência pessoal de implementação.

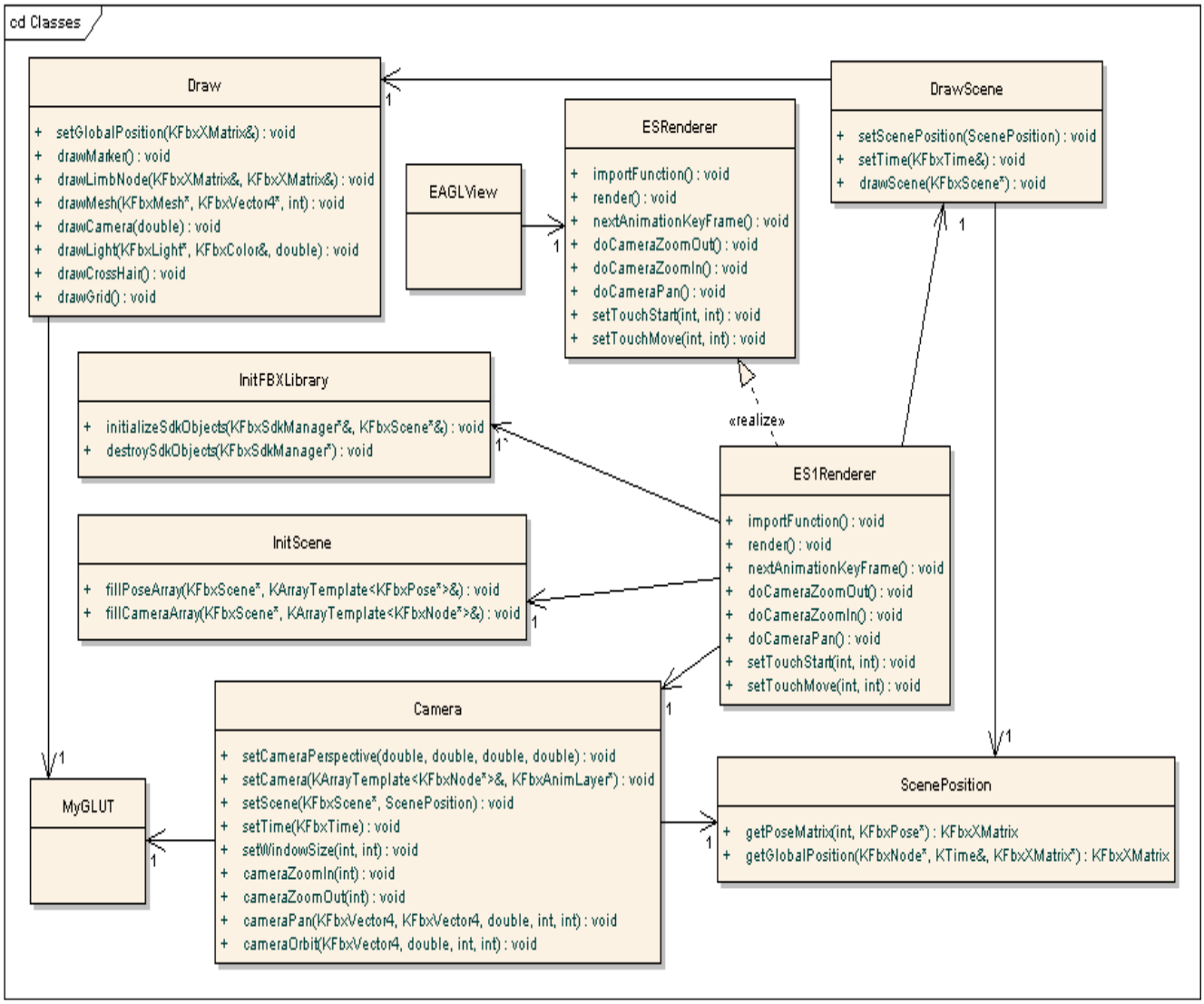


Figura 15 – Diagrama de Classes do `iSceneViewer`

3.3 IMPLEMENTAÇÃO

Esta seção apresenta as bibliotecas e ferramentas utilizadas para o desenvolvimento do protótipo `iSceneViewer`, junto com uma descrição da sua operacionalidade, com trechos do código fonte para um melhor entendimento.

3.3.1 Técnicas e ferramentas utilizadas

As tecnologias utilizadas para o desenvolvimento da proposta foram às seguintes:

a) a linguagem utilizada para realizar a implementação do protótipo foi a Objective-C++;
b) a *Integrated Development Environment* (IDE) utilizada para a codificação do protótipo foi o xCode;
c) para depurar e simular a execução do protótipo, foi utilizado o iPhone Simulator, que funciona integrado com a IDE do xCode;
d) a API utilizada para a geração das imagens gráficas no espaço 3D foi o OpenGL ES, desenvolvida para sistemas embarcados;
e) Para fazer a interpretação do arquivo FBX e o armazenamento da estrutura de dados nele presente foi usado o FBX SDK versão 2011.2.

3.3.2 Desenvolvimento do Protótipo `iSceneViewer`

O protótipo implementado neste trabalho é baseado nos exemplos para a plataforma MAC OS X presentes dentro do próprio FBX SDK. Tais exemplos foram escritos na linguagem C++ e utilizam as bibliotecas OpenGL para fazer o desenhos no ambiente 3D e a GLUT para a fazer interações com usuário através do mouse ou dos comandos no teclado.

No iOS 4 SDK não existe uma versão da biblioteca GLUT e a OpenGL ES possui certa limitações em relação ao OpenGL tradicional. Assim, algumas das diferenças na implementação de um aplicativo gráfico na plataforma iOS 4, bem como comentários sobre iteração com usuário e desenho no ambiente 3D serão apresentadas nessa seção.

3.3.2.1 Configuração do projeto no xCode

Para fazer utilização do FBK SDK é necessário baixar o pacote de instalação para Mac OS X versão 2011.2 para Mac OS X e que hoje ainda está na versão beta, porém, é a primeira versão possível de se compilar juntamente com o iOS 4 SDK. O motivo do uso desta versão, é que as versões anteriores deste SDK eram dependentes do *framework* `CoreServices` não existentes no iOS 4 SDK, somente encontrados no Mac OS X SDK, conforme foi possível observar após as diversas tentativas de compilação das versões 2010.1 e 2010.2 do FBX SDK.

Antes da compilação do projeto, primeiramente foi necessário apontar para os *headers*

do FBX SDK, conforme demonstra a Figura 16, adicionar as bibliotecas `libiconv.dylib` que está presente no iOS 4 SDK e também a `libfbxsdk_gcc4_ub.a` que está presente dentro do diretório onde foi instalado o FBX SDK.

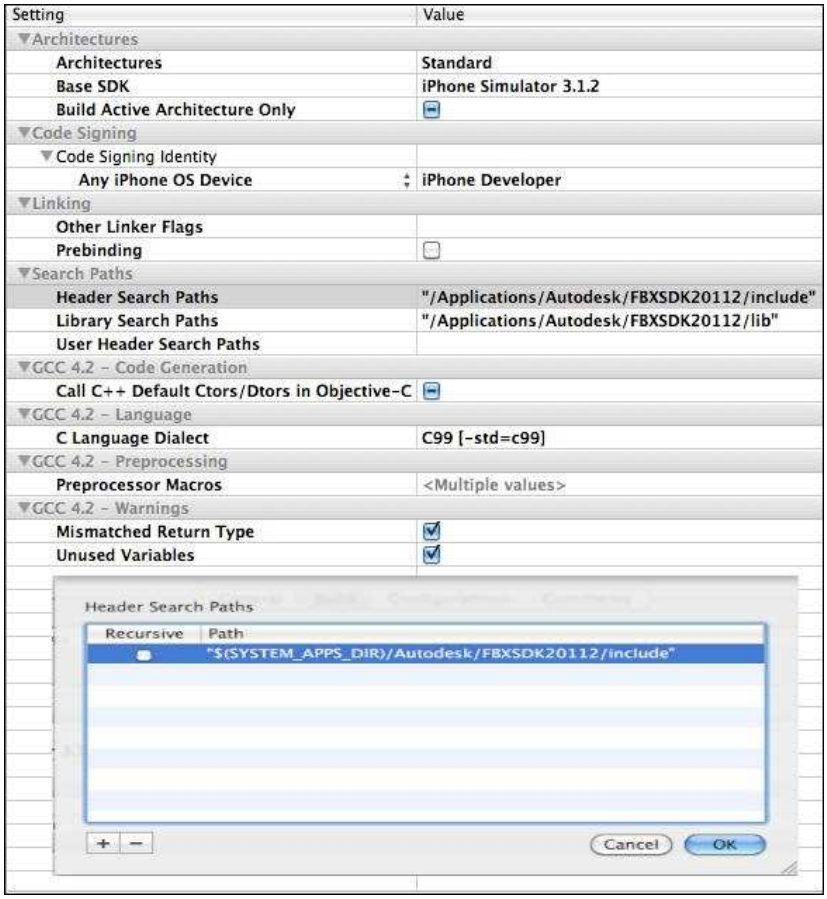


Figura 16 – Configuração do projeto no xCode

Também foi necessário adicionar ao projeto antes da compilação, os *frameworks* `SystemConfiguration`, `CoreFoundation`, `Foundation`, `OpenGLES` e `CFNetwork` conforme demonstra a Figura 17. Após isso, o projeto esta apto a instanciar os objetos necessários para importar um arquivo FBX e utilizar a estrutura de dados fornecida pelo FBX SDK dentro de um aplicativo na plataforma iOS 4.

Figura 17 – *Framework* e bibliotecas adicionadas ao projeto

3.3.2.2 Adaptando OpenGL ao OpenGL ES

A classe `DrawScene` é a classe responsável pela leitura do grafo de cena e de fazer as chamadas aos métodos da classe `Draw` conforme o tipo de nodo encontrado durante a leitura do grafo. A classe `Draw` é responsável pelas chamadas diretamente ao OpenGL ES e através de seus métodos é possível visualizar a principal diferença em relação ao OpenGL. Como os procedimentos `glBegin`, `glEnd` e `glVertex3dv` usados no exemplo estudado não existem na OpenGL em sua versão para sistemas embarcados (OpenGL ES), foi necessário substituí-los pelos procedimentos `glVertexPointer` e `glDrawArrays` possibilitando assim fazer os desenhos das formas geométricas presentes no grafo de cena.

Outra adaptação importante foi a conversão de todos os vetores do tipo de dado `KFbxVector4` para `float array`, isso foi necessário devido ao FBX SDK ter como padrão para `cast` e sobrecarga de operadores, sempre fazer conversões de seus tipos internos para o tipo `double` e não `float`. Já o OpenGL ES foi projetado para trabalhar apenas com tipo de dado `float` nos valores com casas decimais, pois o tipo de dado `double` não é suportado em algumas plataformas embarcadas e comprometeria a compatibilidade da biblioteca.

As figuras 18 e 19 apresentadas abaixo exemplificam essas adaptações, onde na Figura 18 está a versão de um código que utiliza o OpenGL tradicional e na Figura 19 está a versão usando a OpenGL ES e é a versão que foi o usado na implementação do `iSceneViewer`. Ambos os quadros apresentam o mesmo trecho de implementação do método `drawMesh` da classe `Draw`.

```
...
for (lPolygon = 0; lPolygon < lPolygonCount; lPolygon++) {

  int lVertice;
  int lVerticeCount = pMesh->GetPolygonSize(lPolygon);

  glBegin(lGLPrimitive);
  for (lVertice = 0; lVertice < lVerticeCount; lVertice++) {
    glVertex3dv((GLdouble *)pVertexArray[pMesh->
    GetPolygonVertex(lPolygon, lVertice)]);
  }
  glEnd();
}
...
```

Figura 18 – Implementado do método `drawMesh` no OpenGL

```
...
glEnableClientState(GL_VERTEX_ARRAY);

for (lPolygon = 0; lPolygon < lPolygonCount; lPolygon++) {
  int lVertice;
  int lVerticeCount = pMesh->GetPolygonSize(lPolygon);

  float array[lVerticeCount * 3];

  for (lVertice = 0; lVertice < lVerticeCount; lVertice++) {
    array[(lVertice * 3)] = pVertexArray[pMesh->
    GetPolygonVertex(lPolygon, lVertice)][0];

    array[(lVertice * 3) + 1] = pVertexArray[pMesh->
    GetPolygonVertex(lPolygon, lVertice)][1];

    array[(lVertice * 3) + 2] = pVertexArray[pMesh->
    GetPolygonVertex(lPolygon, lVertice)][2];
  }
  glVertexPointer(3, GL_FLOAT, 0, array);
  glDrawArrays(lGLPrimitive, 0, lVerticeCount);
}

glDisableClientState(GL_VERTEX_ARRAY);
...
```

Figura 19 – Implementação do método `drawMesh` no OpenGL ES

3.3.2.3 Adaptando a GLUT

Visto a necessidade de utilizar alguns dos procedimentos fornecidos pelas bibliotecas GLU e GLUT no protótipo, foi criada a classe `MyGLUT` que faz a implementação destes métodos com as devidas adaptações necessárias para a OpenGL ES. As classes `Camera` e `Draw` possuem referencia para a classe `MyGLUT`, onde a classe `Camera` utiliza o método `gluLookAt`, que é o responsável por posicionar a câmera no ambiente 3D fazendo as devidas translações e rotações necessárias de forma que a câmera sempre aponte para um mesmo ponto, e o método `gluPerspective`, que é responsável por informar ao OpenGL ES o tipo de projeção que deve utilizado para dar sensação de profundidade a cena.

A classe `Draw` utiliza a classe `MyGLUT` para fazer chamada ao método `glutWireCube`, usado para desenhar um cubo que representa a posição da luz na cena importada, e para o método `glutWireCone` usado para mostrar a área de efeito da luz sobre a cena importada. Ambos os métodos são chamados pelo método `drawLight` da classe `Draw`.

Na Figura 20 é apresentado como ficou o código de implementação do método `gluPerspective`, é notável a utilização do procedimento `glFrustumf`, onde "f" no final do nome do procedimento representa o tipo dado aceito como parâmetro, que no caso, deve ser do tipo `float` e também da multiplicação por 2 nas variáveis locais `ymin` e `ymax` para evitar que a cena fique "esticada" devido a tela do iPhone ser retangular e não quadrada.

```
- (void)gluPerspective:(GLfloat)fovy asp:(GLfloat)aspect near:(
GLfloat)zNear far:(GLfloat)zFar {
     GLfloat xmin, xmax, ymin, ymax;

     ymax = zNear * tan(fovy * M_PI / 360.0);
     ymin = -ymax;
     xmin = ymin * aspect;
     xmax = ymax * aspect;

     glFrustumf(xmin, xmax, ymin * 2, ymax * 2, zNear, zFar);
}
```

Figura 20 – Implementação do método `gluPerspective`

3.3.2.4 Controle da taxa de quadros por segundo e animação

A classe `EAGLView` é a classe responsável por responder aos eventos de toque na tela e também por iniciar os *loops* principais do protótipo que fazem chamadas para os métodos `render` e `nextAnimationKeyFrame` da classe `ES1Renderer`. Para controlar o intervalo de chamada destes métodos são utilizados dois objetos da classe `CADisplayLink` do *framework* `CoreAnimation` presente no iOS 4 SDK.

O motivo da utilização deste recurso é que o método `render` deve possuir um intervalo de tempo muito pequeno entre uma chamada e outra, porque ele é o método responsável por desenhar as formas geométricas e recalcular o posicionamento de câmera. A demora na chamada ou execução deste método deixa lenta a taxa de quadros, dando a impressão de o aplicativo estar travando. Já o método `nextAnimationKeyFrame` deve respeitar a taxa de intervalos de quadros que esta definida dentro na animação que esta sendo executada, esta informação está contida dentro do arquivo FBX importado e pode ser acessada através do SDK pela classe `KFbxGlobalSettings`. Respeitando essa taxa de atualização, não importa a capacidade de processamento da versão do dispositivo iPhone que está rodando o protótipo, que a sequencia de quadros da animação vai levar o mesmo período de tempo para ser executada em qualquer contexto.

A Figura 21 demonstra os trechos de código dos métodos `initWithCoder` e `starAnimation` da classe `EAGLView` que são os métodos que inicializam os gatilhos utilizados pelo protótipo.

No método `initWithCoder` é criado o gatilho `displayLink` responsável por chamar o método `drawView` que desenha a cena na tela. Esse gatilho responde ao intervalo `displayInterval`, que por *default* vale 1.

Já o método `startAnimation` cria o objeto `animateLink` responsável por chamar o método `nextAnimationKeyFrame` que incrementa o índice do quadro atual a ser desenhado. Esse gatilho responde ao intervalo `animationFrameInterval` que pode variar dependendo na animação escolhida para visualização da sua execução.

```
...
- (id) initWithCoder:(NSCoder*)coder {
...
  displayLink = [NSClassFromString(@"CADisplayLink")
  displayLinkWithTarget:self selector:@selector(drawView:)];

  [displayLink setFrameInterval:displayInterval];

  [displayLink addToRunLoop:[NSRunLoop currentRunLoop]
  forMode:NSDefaultRunLoopMode];
...
}

...
- (void) startAnimation {
  if (!animating) {
    animateLink = [NSClassFromString(@"CADisplayLink")
    displayLinkWithTarget:self
    selector:@selector(nextAnimationKeyFrame:)];

    [animateLink setFrameInterval:animationFrameInterval];

    [animateLink addToRunLoop:[NSRunLoop currentRunLoop]
    forMode:NSDefaultRunLoopMode];

    animating = TRUE;
  }
}
...
```

Figura 21 – Utilização da classe `CADisplayLink` do *framework* `CoreAnimation`

3.3.3 Operacionalidade da implementação

Esta seção apresenta a operacionalidade do `iSceneViewer` através da execução de um estudo de caso. Para demonstração do protótipo foi utilizado o arquivo `humanoid.fbx` que está presente dentro do diretório de exemplos do FBX SDK. Para a execução do protótipo foi usado o iPhone Simulator integrado ao xCode, IDE utilizada para desenvolvimento do protótipo.

A Figura 22 apresenta a tela inicial no `iSceneViewer` onde é visualizado o ambiente 3D sem qualquer cena carregada, sendo apenas possível visualizar um *grid* padrão, que serve para auxiliar na orientação do usuário. Por opção do usuário, a exibição do *grid* pode ser desativada.

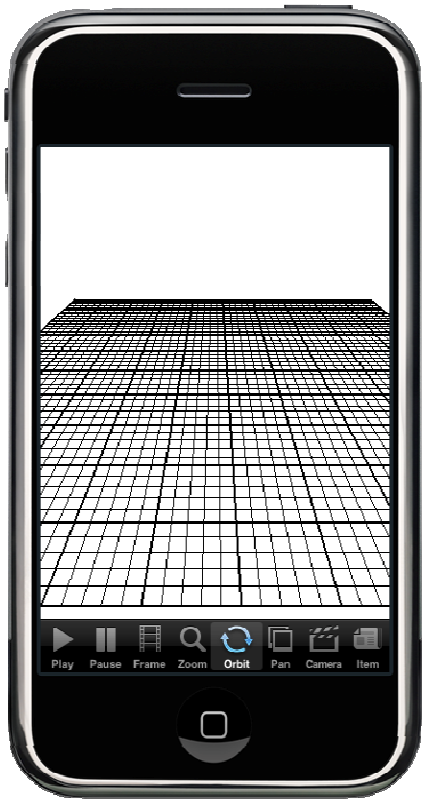


Figura 22 – Tela incial do `iSceneViewer`

A tela inicial já trás o *menu* principal do protótipo, onde para escolha do arquivo a ser visualizado de ser acionado o botão `item` mais a direita do *menu*, onde o protótipo apresenta uma tela para fazer a seleção do arquivo FBX que se deseja visualizar. Neste estudo de caso, o arquivo FBX utilizado é o `humanoid.fbx` que está presente dentro dos exemplos de implementação disponibilizados no diretório de instalação do FBX SDK.

Ao selecionar e confirmar o arquivo desejado, a cena é importada para a estrutura de

dados presente no FBX SDK e então desenhada no *viewport* através das chamadas a OpenGL ES. Caso a cena importada possua animações, o `iSceneViewer` posiciona-se no primeiro quadro da primeira animação na lista de animações do arquivo FBX, como acontece neste estudo de caso, onde a primeira animação é chamada de *run*, conforme demonstra a Figura 23.

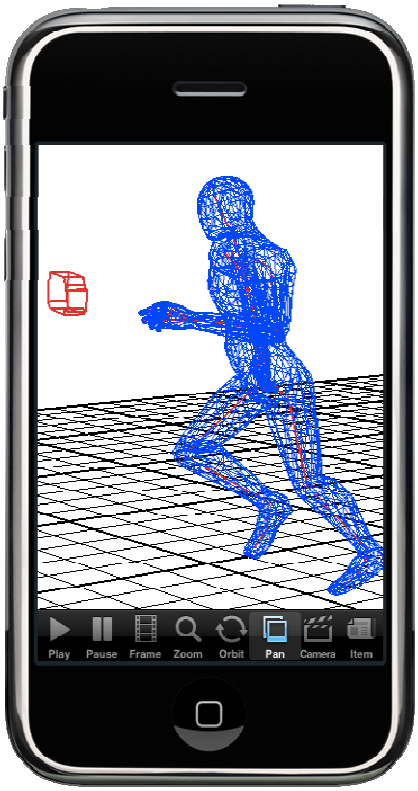


Figura 23 – Visualização da cena presente no arquivo `humanoid.fbx`

Inicialmente, todas as estruturas presentes no arquivo e suportadas pelo protótipo são carregadas no *viewport* e podem ser desativadas de forma total ou parcialmente também usando a opção *item* no *menu* principal. Conforme demonstra a Figuras 24, onde é a estrutura *mesh* não é desenha.

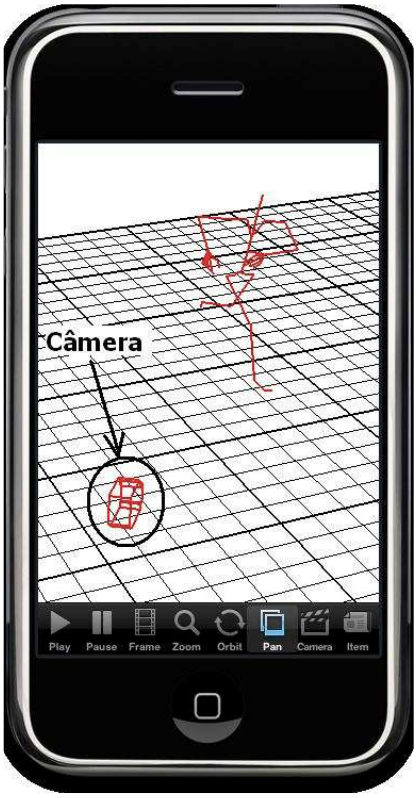


Figura 24 – Visualização parcial das estruturas importadas

Já na Figura 25, apenas estrutura *skeleton* é desenhada no *viewport*. Observe que a câmera e o *grid* não aparecem nesta imagem.

Figura 25 – Visualização da estrutura *skeleton*

Para fazer a visualização das animações importadas basta iniciar a animação corrente no *menu* principal acionando o botão *play*, caso exista. A Figura 26 apresenta uma sequencia de quadros representado a execução da animação *run* feita pelo `iSceneViewer`. É possível observar que a câmera encontra-se na opção padrão da animação, acompanhando a corrida do personagem, conforme foi definido no momento da confecção do arquivo. Para utilização desta opção de visualização deve ser acionar o botão *camera* no *menu* principal.

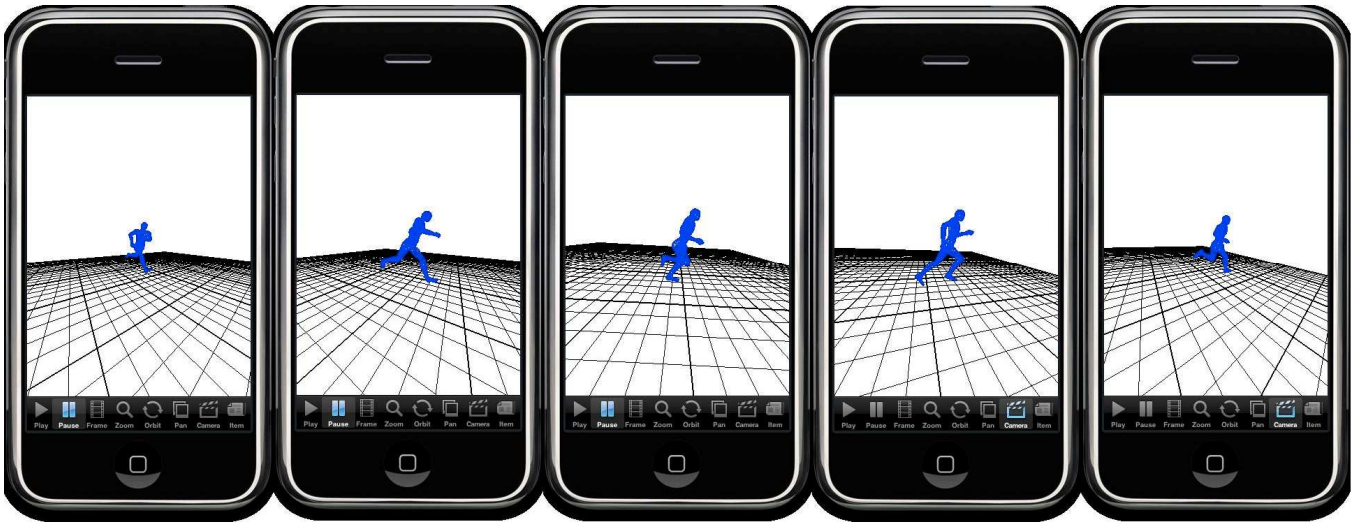


Figura 26 – Animação *run* sendo executada pelo `iSceneViewer`

As opções das diferentes formas de movimentação de câmera pela cena também estão disponíveis no *menu* principal e o comportamento da câmera responde de acordo como os movimentos dos dedos do usuário na tela do iPhone, juntamente com a opção de movimentação de câmera escolhida.

A Figura 27 apresenta a opção *zoom*, onde dois dedos devem ser utilizados para aumentar ou diminuir a distância da câmera em relação à cena. Ao arrastar os dedos sobre a tela do iPhone aumentando a distância entre eles, o protótipo responde diminuindo a distância da câmera em relação à cena, fazendo o movimento conhecido como *zoom in*, ao diminuir a distância entre os dedos, o protótipo faz o movimento contrário, conhecido como *zoom out*.

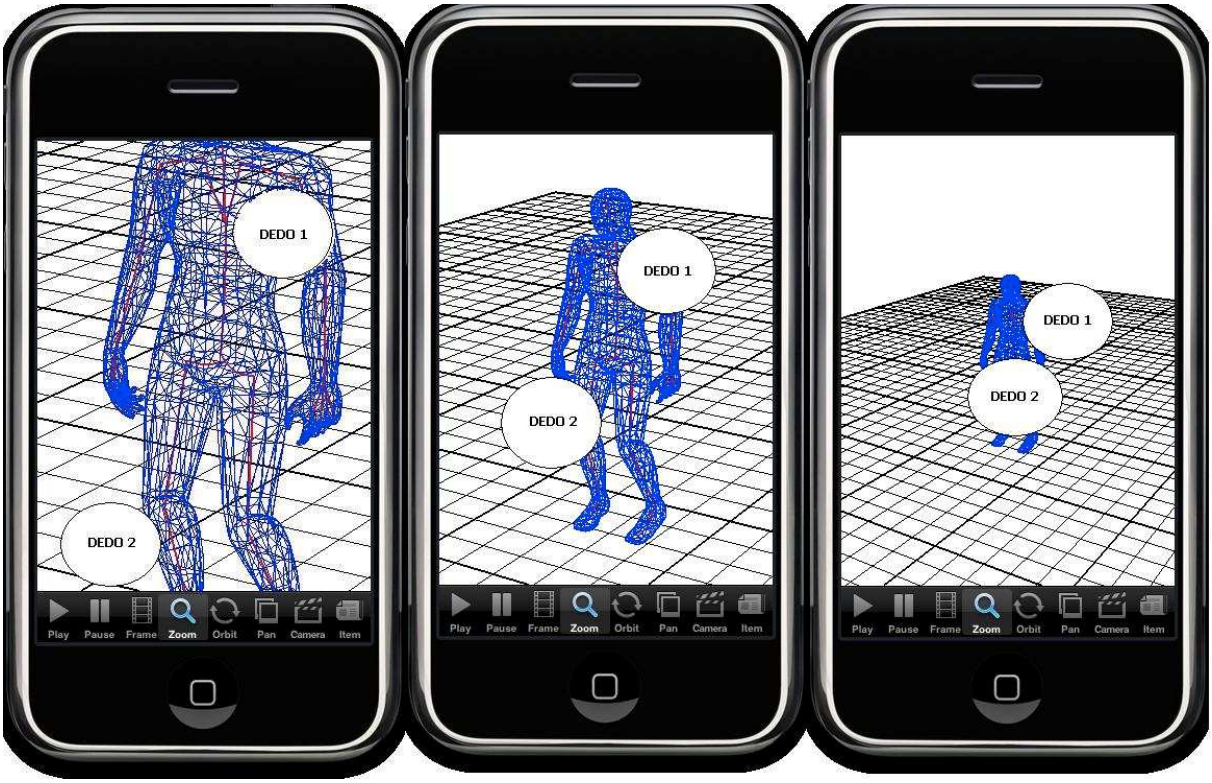


Figura 27 – Movimentação da câmera em modo *zoom*

3.4 RESULTADOS E DISCUSSÃO

O presente trabalho apresentou o projeto e desenvolvimento de um protótipo para visualizar arquivos FBX na plataforma iOS 4, onde foi utilizado o SDK disponibilizado pela Autodesk para fazer a interpretação do arquivo FBX e leitura do grafo de cena. O protótipo permite ao usuário visualizar as formas geométricas que compõem o modelo 3D, o esqueleto, as posições de câmera e também as animações, além de vários modos de movimentação de câmera pelo ambiente 3D.

Apenas o requisito funcional de seleção do arquivo não foi atendido, devido a restrições existentes no acesso a diretórios do dispositivo iPhone, onde uma abordagem para acesso dos arquivo FBX via *download* através da internet, seria mais adequada.

Na Figura 28 é apresentado um comparativo entre alguns dos trabalhos e o protótipo `iSceneViewer`. O trabalho correlato FBX Converter não está neste quadro comparativo, pois se trata de outro tipo de aplicativo, apenas para converter um arquivo do formato FBX para outro tipo de arquivos suportados por ele, não fazendo sentido colocá-lo nas comparações feitas abaixo.

	TAKANO 2009	PISKE 2008	FBX for QuickTime 2009	iSceneViewer 2010
Suporte a tecnologia FBX	Não	Não	Sim	Sim
Suporte a texturas	Não	Sim	Sim	Não
Plataforma móvel	Sim	Sim	Não	Sim
Edição do arquivo FBX	Não	Não	Sim	Não

Figura 28 – Comparativos dos trabalhos correlatos

Em relação aos trabalhos Takano (2009) e Piske (2008) o `iSceneViewer` se destaca pelo suporte a tecnologia FBX, permitindo a visualização de cenas 3D presentes nos arquivo importados deste formato, não se limitando aos formatos MD2 e OBJ que são formatos mais primitivos e que embora muito usados, não seguem a mesma evolução encontrado na tecnologia FBX que temos hoje.

Outro fato importante em relação ao trabalho FBX for QuickTime (2009) é de não existir uma versão para uma plataforma móvel deste *plugin*, embora seja mais completo que o `iSceneViewer`, permitindo inclusive a edição e geração de cenas 3D. Essa limitação foge do desafio inicial deste trabalho que é portar a tecnologia FBX em uma plataforma móvel visto a evolução do mercado para estes tipos de software e a necessidade de ferramentas para auxiliar no desenvolvimento dos mesmos, principalmente na área de jogos.

4 CONCLUSÕES

Os resultados obtidos com o desenvolvimento deste protótipo foram satisfatórios. Os requisitos propostos principais foram cumpridos, alcançando assim o objetivo inicial do trabalho, o desenvolvimento de um protótipo que permitisse a visualização um uma cena 3D importada de um arquivo FBX na plataforma iOS 4 utilizando o Objective-C++.

O `iSceneViewer` permite a importação de um arquivo FBX versão 6.0 e possibilita ao o usuário a visualização dos elementos da cena 3D importada, de forma completa ou parcial, dentre as estruturas suportadas. Quanto às principais vantagens do protótipo estão as funcionalidades de rotação da câmera, aproximação e translação da câmera pelo ambiente 3D e visualização das animações com poucos toques na tela do dispositivo iPhone. Dentre estas funcionalidades destaca-se a de movimentação da câmera pelo ambiante, que proporciona grande comodidade ao usuário que pode observar em detalhes a cena 3D, sem precisar de nenhum outro recurso computacional para isso.

Quanto as tecnologias utilizadas, a IDE de desenvolvimento xCode, se tornou suficiente e eficiente para o desenvolvimento. A integração com o simulador do iPhone facilita muito no desenvolvimento. O FBX SDK disponibilizado pela Autodesk, versão 2011.2 também se mostrou compatível com o iOS 4 SDK possibilitando assim sua utilização no desenvolvimento deste protótipo.

Como limitações o `iSceneViewer` não suporta a visualização de texturas e iluminação possivelmente presentes em uma cena 3D importada visto que essas estruturas são suportadas pela tecnologia FBX. O protótipo também não permite a edição ou criação de novas cenas 3D, embora o SDK utilizado pelo mesmo forneça as classes e estruturas para a implementação destas funcionalidades.

É importante frisar que até o presente momento, não existe na literatura qualquer outro aplicativo ou protótipo capaz de ler e exibir modelos FBX na plataforma iPhone, a não ser o projeto de SCRIPTLANCE (2010), cujos resultados ainda não foram divulgados.

4.1 EXTENSÕES

Como extensão para um trabalho futuro, seria possível a inclusão do suporte a texturas

juntamente com seu mapeamento sobre os objetos 3D importados e também o suporte para representação das luzes da cena, conforme foram definidas no momento da confecção da cena presente em um arquivo FBX importado. Isto seria bastante recomendável, visto que essas informações estão presentes no arquivo FBX e que as classes para fazer a manipulação destas informações também estão disponíveis no SDK.

Sugere-se também a adição de funcionalidades para edição e criação de cenas 3D dentro do próprio iPhone ou até no recém lançado iPad, visto que o SDK também disponibiliza classes para edição e geração um arquivo editado ou mesmo de um novo arquivo FBX.

Por fim, as rotinas que dão o suporte a tecnologia FBX e que fazem a renderização do modelo 3D e de suas animações implementados nesse protótipo, poderiam ser adicionados a um motor de jogos 3D para plataforma iOS 4.

REFERÊNCIAS BIBLIOGRÁFICAS

APPLE. **iPhone 4 technical specifications.** [S.1.], [2010?]. Disponível em: <http://www.apple.com/iphone/specs.html>. Acesso em: 28 jun. 2010.

AREA. **Autodesk FBX®**. [S.l.], [2010?]. Disponível em: <http://area.autodesk.com/fbx>. Acesso em: 29 mar. 2010.

ASTRO. **Objective-C.** Uma iniciação aos conceitos da linguagem. [S.l.], [2005]. Disponível em: <http://www.astro.iag.usp.br/~algol/computacao/ObjCtutorial.html>. Acesso em: 22 mar. 2009.

AUTODESK. **SDK Help.** [S.l.], [2009?]. Disponível em: <http://download.autodesk.com/us/fbx/2010/FBX_SDK_Help/index.html?url=WS1a9193826455f5ff-150b16da11960d83164-6c6f.htm,topicNumber=d0e127>. Acesso em: 20 abr. 2009.

DEVELOPER. **iPhone OS Reference Library.** [S.1.], [2010?]. Disponível em: <http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/ObjectiveC/Articles/ocCPlusPlus.html#//apple_ref/doc/uid/TP30001163-CH10-SW1>. Acesso em: 08 jul. 2010.

ECONOMISTE.COM. **Nomads at last**. [S.l.], [2008?]. Disponível em: <http://www.economist.com/opinion/displaystory.cfm?story_id=10950394>. Acesso em: 29 mar. 2009.

KHRONOS. **OpenGL ES overview**. [S.l.], [2009]. Disponível em: <http://www.khronos.org/opengles/>. Acesso em: 27 mar. 2009.

NAQL, Manfred; SCHURR, Andy; ZUNDORF, Alberts. **Applications of graph transformations with industrial relevances:** revised selected and invited papers**.** Springer, 2008.

PISKE, Tiago. **Reconstrutor de modelos 3D utilizando técnicas de nível de detalhamento.** 2008. 64 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2008/330435_1_1.pdf/>. Acesso em: 27 mar. 2009.

ROUGHLYDRAFTED MAGAZINE. **Daniel Eran Dilger in San Francisco**. [S.l.], [2007?]. Disponível em: <http://www.roughlydrafted.com/2007/07/13/iphone-os-x-architecture-the-mach-kernel-and-ram/>. Acesso em: 27 mar. 2009.

SCRIPTLANCE. **Connecting businesses with programmers**. [S.l.], [2010?]. Disponível em: <http://www.scriptlance.com/projects/1232925001.shtml>. Acesso em: 01 jun. 2010.

SDKDOC. **FBX SDK programmers guide**. [S.l.], [2010?]. Disponível em: <http://area.autodesk.com/img/static/fbx/FBX_SDK_Programmers_Guide_2009_3.pdf>. Acesso em: 30 maio 2010.

TAKANO, Rafael H. **MJ3I: Um motor de jogos 3D para iPhone OS.** 2009. 52 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau. Disponível em: <http://www.bc.furb.br/docs/MO/2010/341418_1_1.pdf>. Acessa em: 20 maio 2010.

TENON INTERSYSTEMS. **World-class networking for macintosh**. [S.1], [2010?]. Disponível em: <http://www.tenon.com/products/codebuilder/Objective-C.shtml>. Acesso em: 02 jun. 2010.

UFSC. **Introdução a programação orientada a objetos com Smalltalk**. [S.1.], [2010?]. Disponível em: <http://www.inf.ufsc.br/poo/smalltalk/jonathan>. Acesso em: 08 maio 2010.