RAPHAEL PETEGROSSO
RENATO HIROAKI TANO
WILLIAM SHINJI TANIGUCHI

ANÁLISE DE VIABILIDADE DA INTEGRAÇÃO DE COMPONENTES 3D (DIRECT3D) COM COMPONENTES NATIVOS PARA .NET COMPACT FRAMEWORK

São Paulo

2008

RAPHAEL PETEGROSSO
RENATO HIROAKI TANO
WILLIAM SHINJI TANIGUCHI

ANÁLISE DE VIABILIDADE DA INTEGRAÇÃO DE COMPONENTES 3D (DIRECT3D) COM COMPONENTES NATIVOS PARA .NET COMPACT FRAMEWORK

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para Conclusão do Curso de Engenharia de Computação

Orientadora:
Prof. Dra. Selma Shin Shimizu Melnikoff

Co-Orientador:
Eng. Bruno Toshio Sugano

São Paulo
2008

RAPHAEL PETEGROSSO
RENATO HIROAKI TANO
WILLIAM SHINJI TANIGUCHI

ANÁLISE DE VIABILIDADE DA INTEGRAÇÃO DE COMPONENTES 3D (DIRECT3D) COM COMPONENTES NATIVOS PARA .NET COMPACT FRAMEWORK

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para Conclusão do Curso de Engenharia de Computação

São Paulo
2008

FICHA CATALOGRÁFICA

Tano, Renato Hiroaki
Análise de viabilidade da integração de componentes 3D (Direct3D) com componentes nativos para .net compact framework / R.H. Tano. -- São Paulo, 2008.
p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Frameworks 2. 3D studio I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

ERRATA

PÁGINA	LINHA	ONDE SE LÊ	LEIA-SE

DEDICATÓRIA

Aos meus pais, Nádia e Moacir, que me educaram e me fizeram enxergar os caminhos da vida e do meu desenvolvimento pessoal.

À minha namorada Pamela, que me apóia sempre e me dá forças para querer sempre mais.

E aos colegas e professores da universidade, que contribuíram para que estes cinco anos de faculdade fossem uma fase inesquecível.

-Raphael Petegrosso

À minha família, que me apoiou em todos os momentos da minha vida, oferecendo todo o suporte necessário para a minha caminhada até aqui.

Aos meus amigos, que sempre estiveram perto em todos os momentos, sejam eles bons ou ruins.

E a minha namorada e companheira, Adriana, por toda a ajuda, carinho, compreensão e atenção durante todos os anos juntos.

-Renato Hiroaki Tano

Aos meus pais, Alice e Celestino, que me apoiaram em todos os momentos, servindo de alicerce em todas as minhas conquistas.

E aos meus amigos, em especial aos colegas e professores de Escola Politécnica, pelos momentos de diversão, de estudo e de companheirismo que dividimos.

-William Shinji Taniguchi

AGRADECIMENTOS

À Professora Selma Melnikoff, pelo incentivo, orientação e disposição em todos os momentos durante o ano.

Ao Bruno Toshio Sugano, por toda a orientação técnica e não técnica quando foi necessário.

A Escola Politécnica da Universidade de São Paulo, que nos deu a oportunidade de aprendizagem e crescimento.

Ao Departamento de Engenharia de Computação e Sistemas Digitais (PCS), pelo Curso Cooperativo de Engenharia da Computação.

À Empresa I.ndigo, pelo apoio tanto em equipamentos quanto em horas, durante todo o projeto desenvolvido.

Ao Edmar Miyake, pelo apoio, amizade e comprometimento, oferecendo suporte para todos do grupo.

RESUMO

A utilização de interfaces que fazem uso de componentes tridimensionais é cada vez mais comum em dispositivos móveis. Celulares, *PDA's* e *Smartphones* apresentam esse tipo de interface como diferencial na disputa de um mercado em crescente adoção, por sua maior integração com o usuário.

A plataforma *.NET*, da Microsoft, por meio do *.NET Compact Framework*, é utilizada por parte considerável da comunidade desenvolvedora de aplicações móveis, porém carece de ferramentas para utilização de componentes tridimensionais para interface, em especial em conjunto com os componentes visuais nativos da plataforma.

O trabalho descrito a seguir tem como objetivo analisar a viabilidade da utilização de componentes tridimensionais implementados sobre a tecnologia *Direct3D Mobile* em conjunto com componentes nativos, por meio do desenvolvimento destes componentes, levando em consideração o seu uso pelo usuário final da aplicação e do desenvolvedor.

ABSTRACT

The use of tridimensional user interfaces is becoming each time more popular in mobile devices. Cellphones, PDA's and Smartphones are using this kind of interface as a differential factor in a fast growing market, due to its bigger integration with the final user.

The .NET platform, from Microsoft, through the .NET Compact Framework, is used by a large part of the mobile application development community, but lacks a set of tools for use tridimensional components, specially when used together with native visual components.

This study aims to analyze the feasibility of using tridimensional components, implemented on top of the Direct3D Mobile technology, together with native components, through the development of these components, taking into consideration its use by end users' of the application and the developer himself.

LISTA DE ILUSTRAÇÕES

Figura 1 - Interfaces visuais de aplicações típicas para IPhone e para Windows Mobile 4
Figura 2 - Apple Newton Message Pad 100 7
Figura 3 - HP iPAQ 200 Enterprise 8
Figura 4 - Interface do Windows Mobile 5.0 9
Figura 5 - Tela de memória do Windows Mobile 5.0 10
Figura 6 - Arquitetura do .NET Compact Framework 12
Figura 7 - Exemplos de templates de Projeto no Visual Studio 14
Figura 8 - Árvore de arquivos para projeto de aplicação para dispositivos móveis (esquerda) e para aplicação Web (direita) 15
Figura 9 - Template de projeto para aplicação Web customizado. 16
Figura 10 - Exibição Design Time WYSIWYG (esquerda), e exibição real 17
Figura 11 - Arquitetura básica de um dispositivo móvel 18
Figura 12 - Principais componentes do Direct3D 18
Figura 13 - Lista de Triângulos 22
Figura 14 - Faixa de Triângulos 22
Figura 15 - Leque de Triângulos 23
Figura 16 - Visualização dos Vértices na Renderização de um Cubo 24
Figura 17 - Relação entre um Buffer de Índices e um Buffer de Vértices 25
Figura 18 - Transformação de Translação 28
Figura 19 - Transformação de Escala 28
Figura 20 - Transformação de Rotação em Relação ao Eixo X 29
Figura 21 - Transformação de Rotação em Relação ao Eixo Y 29
Figura 22 - Transformação de Rotação em Relação ao Eixo Z 29
Figura 23 - Aplicação de Textura a um Triângulo 30
Figura 24 - Arquitetura do Sistema 34
Figura 25 - ImageButton via Direct3D 37
Figura 26 - ImageButton pressionado 37
Figura 27 - Paisagem via Direct3D 38
Figura 28 - Paisagem via Direct3D 38
Figura 29 - Exemplo de Uso de TabContainer 39
Figura 30 - Renderização de Meshe criado pelo método Mesh.Box 48
Figura 31 - Textos em Formas de Sprites 49
Figura 32 - Sistemas de Coordenadas 50
Figura 33 - UserControl 53
Figura 34 - Interface em Design Time utilizando DesignTimeAttributes.xmta 55
Figura 35 - Template de projeto disponivel para utilização 58
Figura 36 - Árvore de arquivos com referência aos componentes Direct3D 59
Figura 37 - Diagrama de Casos de Uso 78

LISTA DE TABELAS

Tabela 1 - Versões do .NET Compact Framework.....*13*
Tabela 2 - Tipos de vértice do Direct3D Mobile.....*21*

LISTA DE ABREVIATURAS E SIGLAS

PDA: *Personal Digital Assistant*

VS: *Visual Studio*

.NET CF: *.NET Compact Framework*

IDE: *Integrated Development Environment*

API: *Application Programming Interface*

GDI: *Graphics Device Interface*

CLR: *Common Language Runtime*

JIT: *Just-In-Time*

GC: *Garbage Collection*

WYSIWYG: *What You See Is What You Get*

POC: *Proof of Concept*

FAQ: *Frequently Asked Questions*

DLL: *Dynamic-link Library*

SUMÁRIO

1. INTRODUÇÃO 2

1.1. OBJETIVO 2

1.2. MOTIVAÇÃO 2

1.3. JUSTIFICATIVA 3

1.4. METODOLOGIA DE TRABALHO 4

1.5. ESTRUTURA DE TRABALHO 6

2. CONCEITOS E TECNOLOGIAS ENVOLVIDAS 7

2.1. PERSONAL DIGITAL ASSISTANT (PDA) 7

2.2. MICROSOFT WINDOWS MOBILE 5.0 9

2.2.1. GERENCIAMENTO DA MEMÓRIA DE DADOS E DO SISTEMA OPERACIONAL 9

2.2.2. AUMENTO DA CAPACIDADE DA BATERIA 10

2.2.3. SUPORTE AO DIRECT3D MOBILE 11

2.2.4. SUPORTE AO .NET COMPACT FRAMEWORK 2.0 11

2.3. MICROSOFT .NET COMPACT FRAMEWORK E O MICROSOFT VISUAL STUDIO 2005 11

2.3.1. CRIAÇÃO DE PROJETO CUSTOMIZADO 14

2.3.2. DESIGN TIME CUSTOMIZADO 16

2.4. MICROSOFT DIRECT3D MOBILE 17

2.4.1. DEVICE 19

2.4.2. *BUFFER* DE VÉRTICES E *BUFFER* DE ÍNDICES 21

2.4.3. MESHES 25

2.4.4. SPRITES 26

2.4.5. SISTEMAS DE COORDENADAS 27

2.4.6. TRANSFORMAÇÕES 27

2.4.7. TEXTURAS 29

2.5. Considerações Finais 30

3. PROTÓTIPO 31

3.1. ESPECIFICAÇÃO DE REQUISITOS 31

3.2. ARQUITETURA DE SOFTWARE 32

3.3. SELEÇÃO DE TECNOLOGIA 34

3.3.1. MICROSOFT XNA 35

3.3.1.1. RESULTADOS DA ANÁLISE DO MICROSOFT XNA 35

3.3.2. DIRECT3D MOBILE 36

3.3.2.1. RESULTADOS DA ANÁLISE DO DIRECT3D MOBILE 36

3.4. DESENVOLVIMENTO DO CUBO 3D 38

3.4.1. CONSIDERAÇÕES INICIAIS 39

3.4.2. DIFICULDADES INICIAIS 39

3.4.3. O DESENVOLVIMENTO DO FORMULÁRIO 3D 40

3.4.4. DESENHO DO CUBO 3D 41

3.4.5. ROTAÇÃO DO CUBO 3D 43

3.4.6. CAPTURA DE TELA 45

3.4.7. TROCA DOS COMPONENTES EXIBIDOS 46
3.5. PLACA 3D 47
3.6. DESIGN TIME CUSTOMIZADO 51
3.6.1. UTILIZAÇÃO DE USERCONTROLS 53
3.6.2. O ARQUIVO DESIGNTIMEATTRIBUTES.XMTA 54
3.6.3. UTILIZAÇÃO DE UM DESIGNER ASSOCIADO AO CONTROLE 3D 56
3.7. PROJETO CUSTOMIZADO 56
3.8. CONSIDERAÇÕES FINAIS 59
4. CONSIDERAÇÕES FINAIS 61
4.1. CONCLUSÕES 61
4.2. CONTRIBUIÇÕES 61
4.3. TRABALHOS FUTUROS 63
5. REFERÊNCIAS BIBLIOGRÁFICAS 66
6. APÊNDICES 70
6.1. APÊNDICE A – ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE 70
6.2. APÊNDICE B – MODELO DE CASOS DE USO 77

1. INTRODUÇÃO

1.1. OBJETIVO

O objetivo deste projeto de formatura é propor a integração, em um mesmo aplicativo, entre componentes tridimensionais, projetados com auxílio da plataforma *Direct3D Mobile* e componentes de formulário nativos (*Windows Forms*) da plataforma *.Net Compact Framework*, da Microsoft.

A integração proposta tem como base a análise através de um protótipo para implementar o uso de componentes tridimensionais em conjunto com componentes nativos. Estes componentes seriam uma forma estilizada em três dimensões de alguns componentes de formulário comumente utilizados em aplicações baseadas no .*Net Compact Framework*.

O protótipo tem ainda, como objetivo secundário, prover a utilização dos componentes tridimensionais da mesma forma que os nativos, sendo transparente o seu uso aos usuários finais, desenvolvedores de aplicações móveis.

1.2. MOTIVAÇÃO

Nos dias atuais, dispositivos computacionais móveis estão cada vez mais presentes no cotidiano das pessoas, seja para fazer ligações telefônicas por meio de aparelhos de telefonia celular, como para realizar o controle de estoque de uma cadeia de suprimentos, por meio de *PDA's*.

No ano de 2008, o Brasil foi listado como 81º economia entre 195, em um estudo sobre a penetração do uso de telefonia celular, contando com 52,9 usuários de telefonia celular por 100 habitantes (United Nations Conference on Trade and Development). No entanto, apesar do Brasil ter uma base grande de usuários de telefonia celular, a utilização de aparelhos conhecidos como *smartphones*, que têm capacidade computacional maior em relação aos aparelhos convencionais, ainda é pequeno. No Brasil, segundo estimativas realizadas sobre números do mercado, feita por uma das maiores operadoras de telefonia celular do país, a base de usuários de *smartphones* representa ainda menos de 1% do total, enquanto em países da Europa Ocidental esse porcentual chega a 18% (ComputerWorld Brasil).

Esse cenário, contudo, já mostra características de mudança, baseadas em um número crescente de adoções de *smartphones*, além atenção cada dia maior por parte das empresas distribuidores de smartphones, como a RIM, detentora da marca *BlackBerry* e a Apple, com a linha de produtos *IPhone*. Ambas entraram apenas recentemente de forma oficial no país, com a Apple, por exemplo, iniciando a disponibilização de seus aparelhos apenas em setembro deste ano (Vivo).

Somando-se a esse público crescente estão os usuários habituais de *PDA's*, representados por empresas que utilizam soluções de computação móvel nas mais diversas áreas, como o controle de cadeia de suprimentos ou em sistemas de automação de força de vendas. Empresas como a Wella ou a Eucatex utilizam sistemas baseados em *PDA's* para a automação de sua força de vendas há alguns anos, formando parte de um mercado também crescente.

A força desse mercado corporativo pode ser medida pelo crescimento de empresas especializadas no desenvolvimento destes sistemas de automação, como as brasileiras W-ITS e I.ndigo Tecnologia.

Ambos os mercados, de smartphones como o IPhone e de *PDA's*, mostram interesse em diversificar o visual de suas aplicações, tendo como expoente o próprio IPhone, que tem como diferencial a utilização de componentes tridimensionais em sua interface. No mês de setembro de 2008, foi lançado também pela T-Mobile em parceria com o Google o aparelho G1, que conta com características semelhantes com relação a interface do IPhone (*CrunchGear*).

Há uma grande parcela ainda de aparelhos baseados na plataforma Windows Mobile, que não conta ainda com uma tecnologia semelhante nem com ferramentas de apoio para composição de interfaces tridimensionais em ambiente de formulários. Essa limitação, em uma plataforma conhecida, serviu de motivação para esse trabalho.

1.3. JUSTIFICATIVA

Ao analisar o mercado de computação móvel, principalmente no uso em *smartphones* e *PDA's*, ficou claro que ainda no Brasil a penetração da plataforma Windows Mobile é significativa. Aos desenvolvedores de aplicações para essa plataforma são disponibilizados apenas componentes visuais antiquados para criação de formulários, em um contexto que incluía as alterações visuais

introduzidas pelo IPhone, como se pode verificar na Figura 1, que apresenta lado a lado uma aplicação utilizando Windows Mobile /.Net Compact Framework e uma aplicação típica do IPhone.

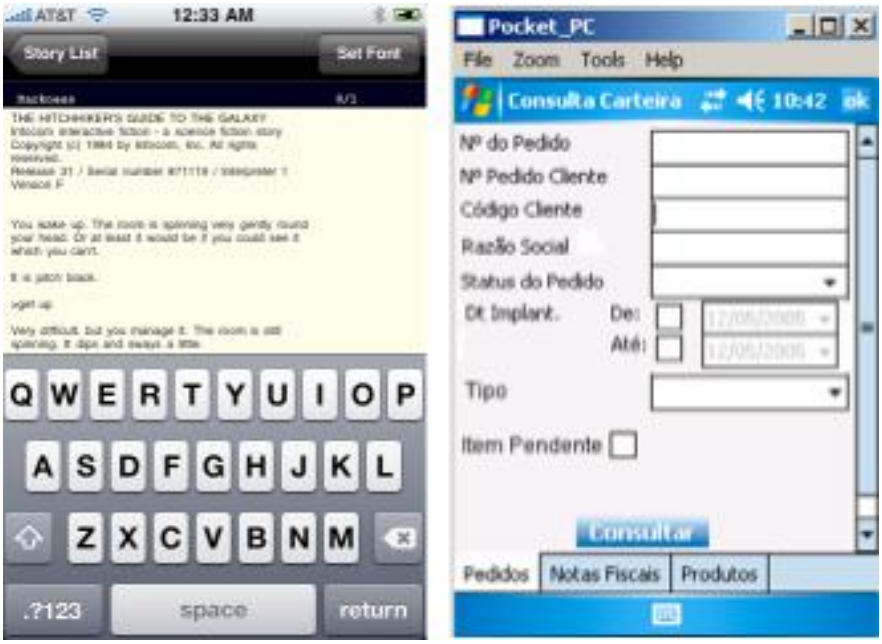

Figura 1 - Interfaces visuais de aplicações típicas para IPhone e para Windows Mobile

Na plataforma Windows Mobile existe a API DirectX da Microsoft, que tem como parte a API Direct3D, responsável pela renderização de modelos em três dimensões, em sua versão móvel (*Direct3D Mobile*). A API Direct3D é utilizada principalmente para o desenvolvimento de jogos.

1.4. METODOLOGIA DE TRABALHO

A metodologia utilizada no projeto de formatura seguiu as seguintes etapas:

- Definição do Escopo do Trabalho e Requisitos de Sistema:
  Foi feita a definição de quais são os componentes, já existentes ou não no .NET Compact Framework, que poderiam ter sua interface alterada para o ambiente tridimensional e dentre estes, quais seriam foco do projeto. Foram definidos nessa fase também os requisitos alvo para o sistema protótipo a ser desenvolvido.
- Estudo das tecnologias disponíveis:

Para desenvolver um ambiente tridimensional nos equipamentos alvo (*PDA's*), estão disponíveis diversas API's, entre elas a API GDI, nativa do sistema operacional Windows Mobile, e o Direct3D Mobile. Cada API tem pontos positivos e negativos, em relação ao escopo do projeto. Realizou-se, então, um estudo sobre cada tecnologia.

- Realização de Provas de Conceito:
  A partir das tecnologias disponíveis para a implementação do projeto, foram realizadas Provas de Conceitos, utilizando de forma prática as funcionalidades oferecidas pelas API's escolhidas.
- Escolha da Tecnologia:
  A partir das provas de conceito e do estudo previamente realizado, foi escolhida a tecnologia a ser utilizada para o desenvolvimento do projeto, com objetivo de escolher a tecnologia que fosse mais abrangente do ponto de vista técnico, oferecendo dessa forma a maior facilidade de uso, maior número de possíveis usuários e maior quantidade de referências.
- Definição de Arquitetura de Desenvolvimento:
  Com a tecnologia a ser utilizada, foi definida qual a arquitetura seria adequada para a implementação na plataforma Windows Mobile.
- Especificação Funcional do Protótipo:
  O protótipo a ser desenvolvido foi definido nessa etapa. Dentre os documentos gerados, estão o documento de especificação de requisitos e um diagrama de Casos de Uso do protótipo.
- Desenvolvimento do Protótipo:
  Nessa fase, foi desenvolvido o protótipo de integração de componentes tridimensionais utilizando como base a sua especificação e a arquitetura definidas.
- Análise do Protótipo e Conclusões
  Ao final do desenvolvimento do protótipo, com base nas dificuldades e soluções encontradas, foi feita uma análise do processo que permitiu extrair idéias e conclusões sobre a viabilidade da integração dos componentes, objetivo do estudo.

1.5. ESTRUTURA DE TRABALHO

Este documento segue a seguinte estrutura:

- Capítulo 1 (Introdução):
  Apresenta objetivo, motivações, justificativas e a metodologia do trabalho.
- Capítulo 2 (Conceitos e Tecnologias Envolvidas):
  Contextualiza o leitor em aspectos técnicos específicos utilizados no desenvolvimento do trabalho.
- Capítulo 3 (Protótipo):
  Descreve como o protótipo foi desenvolvido, quais as suas funcionalidades e como funciona a sua integração com outras ferramentas.
- Capítulo 4 (Considerações Finais):
  Analisa os resultados obtidos em relação ao objetivo do projeto, as conclusões, as contribuições deste trabalho e indica possíveis trabalhos futuros com base neste.

2. CONCEITOS E TECNOLOGIAS ENVOLVIDAS

Neste capítulo são apresentados os conceitos relacionados a *Personal Digital Assistant* (PDA), além das tecnologias e plataformas utilizadas, todas da Microsoft, como o *Windows Mobile 5.0*, *.NET Compact Framework*, *Visual Studio 2005* e *Direct3D Mobile*.

2.1. PERSONAL DIGITAL ASSISTANT (PDA)

O primeiro *PDA* da história foi o *CASIO PF-3000* , lançado pela *GO Corp.* em 1983 (Voidware). O termo *PDA* foi lançado pela *Apple* em um evento tecnológico em 1992 (LowEndMac) durante o lançamento do *Apple Newton* (Figura 2).

Figura 2 - Apple Newton Message Pad 100

Existem *PDA*'s de diversas configurações e funcionalidades. Pode ser utilizado somente como um computador portátil ou possuir também a função de telefone móvel. Basicamente contém programas como calendário, lista de tarefas e lista de contatos.

Muitos provêem algum tipo de conectividade, tais como, *internet* sem fio (*wireless*), infra-vermelho (*irDA*) ou *bluetooth*. *PDA*'s que permitem conexão com a *internet* geralmente possuem suporte para a navegação e *emails*.

A entrada de dados pelo usuário pode ser feita de várias maneiras. Existem equipamentos dotados de *touch screen*, em que o usuário pode inserir dados ou abrir programas diretamente pelo visor do aparelho. Em um outro tipo de aparelho, o usuário interage somente através de botões. Existem ainda *PDA*'s que são dotados de teclado com todo o alfabeto para ajudar na entrada de dados.

Uma das principais funcionalidades dos *PDA*'s é a sua possibilidade de sincronismo. Através dele é possível transferir dados, lista de contatos e até *emails* para um *PC*. Com isso, o usuário pode sincronizar seus dados, tanto para fazer um *backup* para o caso de perda, como também para escrever *emails*, lista de tarefas no computador.

Atualmente os *PDA*'s estão se tornando mais compactos, rápidos e com um poder de armazenamento maior, e um exemplo pode ser visto na Figura 3.

Figura 3 - HP iPAQ 200 Enterprise

2.2. MICROSOFT WINDOWS MOBILE 5.0

O *Windows Mobile* é um sistema operacional destinado a dispositivos móveis tais como *Pocket PC*, *PDA*, entre outros. Devido ao fato de ser desenvolvido pela *Microsoft*, o *Windows Móbile* é muito semelhante a versão do *Windows* para *Computadores* do tipo *PC*.

Especificamente, o *Windows Mobile 5.0* foi lançado em Maio de 2005 (Microsoft) em um evento da própria *Microsoft*, e sua interface pode ser vista na Figura 4.

Figura 4 - Interface do Windows Mobile 5.0

As principais mudanças sobre o seu antecessor, o *Windows Mobile 2003 SE* são as seguintes:

- Aumento da capacidade de bateria;
- Gerenciamento da memória de dados e do sistema operacional;
- Suporte ao *Direct 3D Mobile;*
- Suporte ao *.NET Compact Framework 2.0*.

2.2.1. GERENCIAMENTO DA MEMÓRIA DE DADOS E DO SISTEMA OPERACIONAL

Com o advento do *Windows Mobile 5.0*, o sistema de memória dos *PDA*'s foi modificado de tal forma que permita que os dados não sejam perdidos caso a bateria acabe inesperadamente. Para que isso seja possível foi introduzido o conceito de Memória Persistente ao PDA.

A arquitetura de memória persistente se baseia nos *PC*'s convencionais, ou seja, a memória *RAM* é reservada para o uso dos programas e do sistema operacional – Ver Figura 5. Para o armazenamento de dados é utilizado a memória *ROM*, que não necessita de energia para persistir os dados. Com isso, caso a alimentação do *PDA* seja interrompida, os dados não são apagados.

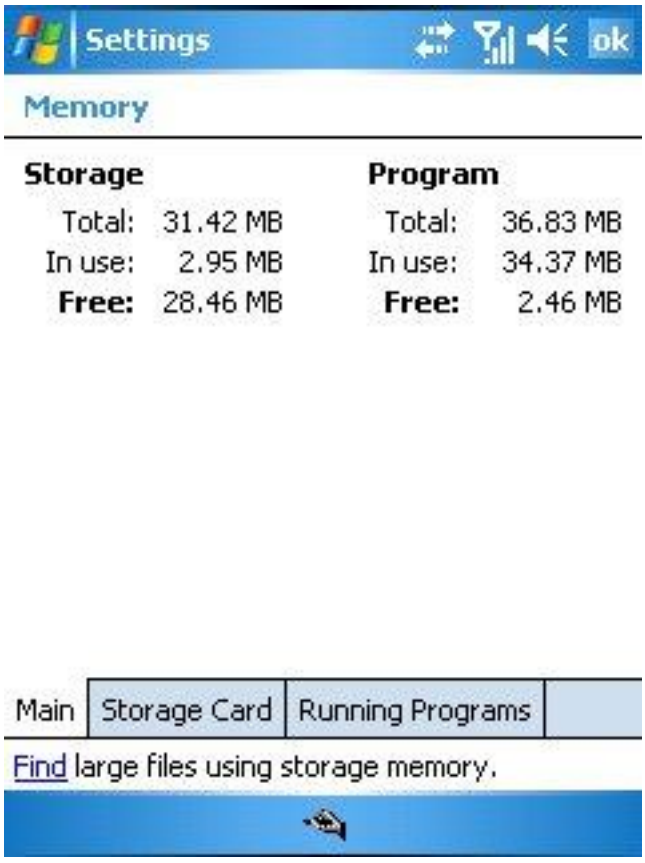

Figura 5 - Tela de memória do Windows Mobile 5.0

2.2.2. AUMENTO DA CAPACIDADE DA BATERIA

Devido ao fato de todos os dados serem armazenados na memória *ROM*, como explicado na seção anterior, o *PDA* não tem a necessidade de utilizar energia para guardar toda a massa de dados do usuário. Isso é importante, visto que o *PDA* tem a função de *sleep*, em que o *PDA* fica com todos os recursos desligados, porém

continua alimentando a memória *RAM* para que, caso o usuário deseje retornar o uso, todos os programas estejam ainda no mesmo estado.

2.2.3. SUPORTE AO DIRECT3D MOBILE

O suporte ao *Direct3D Mobile* foi um grande avanço para os dispositivos móveis. O principal efeito de tal suporte foi a possibilidade de desenvolver jogos utilizando gráficos 3D, expandindo o mercado de aplicações para Windows Mobile.

2.2.4. SUPORTE AO .NET COMPACT FRAMEWORK 2.0

O *Windows Mobile 5.0* possui suporte ao *.NET Compact Framework 2.0*, incorporando as novas ferramentas disponíveis a partir desta versão da plataforma. As características do *.NET Compact Framework 2.0* são descritas no capítulo 2.3.

2.3. MICROSOFT .NET COMPACT FRAMEWORK E O MICROSOFT VISUAL STUDIO 2005

O *.NET Compact Framework* (*.NET CF*) é um ambiente de desenvolvimento criado especificamente para o uso em dispositivos móveis em que os recursos são limitados quando comparados com um computador *desktop*. Ele é executado sobre o sistema operacional *Microsoft Windows CE* e depende da máquina virtual *common language runtime* (*CLR*), desenvolvida especificamente para execução em dispositivos móveis.

O *CLR* pode ser considerado o principal elemento do *.NET CF*. Consiste de uma aplicação que similar a uma máquina virtual responsável em traduzir o código em alto nível das diversas linguagens do *.NET CF* para a linguagem de máquina do dispositivo em que o sistema está rodando.

As principais características do *CLR* no *.NET CF* são descritas a seguir:

Portabilidade – O *CLR* garante a arquitetura necessária para a execução em diversos dispositivos móveis, de acordo com as limitações de cada equipamento como tamanho de tela, opções de hardware, etc.

Class Loader and Verifier – Tem a função de carregar fisicamente os arquivos e as dependências do sistema, além de liberar a sua utilização.

Justi-In-Time (JIT) Compiler – Principal elemento do *CLR*. Tem como função compilar o código, gerando assim o sistema em linguagem nativa do dispositivo. Leva em conta as limitações do hardware e recursos de cada dispositivo.

Garbage Collection (GC) – Responsável por liberar as posições de memória que não estão sendo mais utilizados pelo *.NET CF*. Pelo fato de os dispositivos móveis geralmente possuírem menos memória do que computadores *desktop*, a função do *GC* é de extrema importância para o melhor desempenho dos sistemas.

Multithreading – Permite a utilização de *threads*, ou seja, instâncias do programa que são executados em paralelo.

A Figura 6 mostra a arquitetura de do *.NET Compact Framework*.

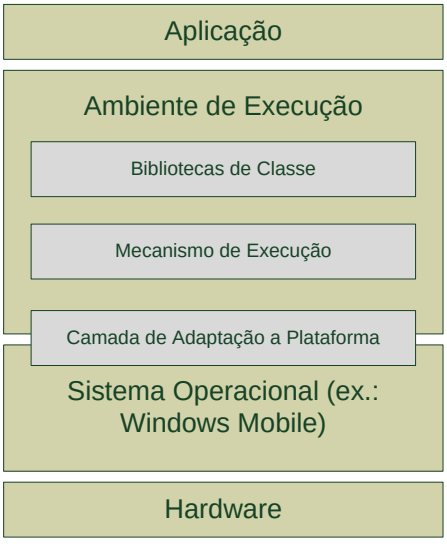

Figura 6 - Arquitetura do *.NET Compact Framework*

O *.NET CF*, da mesma forma que o *.NET Framework*, apresenta estrutura modular, em que as funcionalidades são divididas em módulos, os *Assemblies (runtimes).* Cada um desses módulos deve ser carregado juntamente com a aplicação de acordo com a utilização no sistema.

A Tabela 1 mostra as versões do *.NET Compact Framework 2.0*.

Tabela 1 - Versões do *.NET Compact Framework*

Nome da Versão	Número da Versão	Data de Lançamento
1.0 RTM	1.0.2268.0	Final de 2002
1.0 SP1	1.0.3111.0	Desconhecida
1.0 SP2	1.0.3316.0	Desconhecida
1.0 SP3	1.0.4292.0	Janeiro 2005
2.0 RTM	2.0.5238.0	Outubro 2005
2.0 SP1	2.0.6129.0	Junho 2006
2.0 SP2	2.0.7045.0	Março 2007
3.5 Beta 1	3.5.7066.0	Maio 2007
3.5 Beta 2	3.5.7121.0	Desconhecida
3.5 RTM	3.5.7283.0	19/11/2007
3.5	3.5	25/01/2008
3.5 SP1	3.5 SP1	Agosto 2008

O *.NET Compact Framework* versão 2.0 estende o seu sucessor, *o .NET Compact Framework 1.0*. Além de melhorias nas funcionalidades existentes na versão anterior, provê novas funcionalidades para o desenvolvimento em dispositivos móveis.

O *Visual Studio (VS)* é um conjunto de ferramentas de desenvolvimento para aplicações *ASP.NET, Web Services*, Aplicações *Desktop* e Aplicações *Mobile*. As linguagens *Visual Basic*, *Visual C#*, *Visual J#* e *Visual C++* utilizam o *VS* como *IDE* de desenvolvimento. Isso possibilita que recursos e ferramentas possam ser compartilhados entre as linguagens para criação de sistemas multilinguagem.

Para aplicações móveis, o *VS 2005* oferece ferramentas que integram o desenvolvimento e instalação dos aplicativos em dispositivos móveis.

O *Visual Studio* também provê emuladores que possibilitam depurar e implantar os sistemas sem a necessidade de dispositivos físicos. Também provêm arquivos com extensão *CAB* para a fácil compactação dos arquivos de instalação para a implantação dos sistemas.

Juntamente com a integração da *IDE* com o Dispositivo Móvel através do programa *ActiveSync*, a depuração e implantação de sistemas tornou-se muito similar de como é feito em sistemas *Web* e *Desktop*.

2.3.1. CRIAÇÃO DE PROJETO CUSTOMIZADO

O *Visual Studio*, como *IDE*, centraliza o desenvolvimento de aplicações da plataforma *.NET*. Uma das vantagens em ter o ambiente de desenvolvimento centralizado é que projetos das mais diversas categorias podem utilizar ferramentas comuns, aumentando a familiaridade do usuário com o ambiente de desenvolvimento, diminuindo assim a curva de aprendizado necessária para o início do desenvolvimento.

Para fazer valer essa vantagem da *IDE*, os diferentes tipos de aplicações que podem ser desenvolvidas utilizando o *Visual Studio* são dividas em *templates* de Projeto. Estes *templates* configuram a *IDE* para utilizar as ferramentas pertinentes ao desenvolvimento específico daquele tipo de aplicação, além de prover a base do projeto a ser desenvolvido. Exemplos de *templates* podem ser vistos na Figura 7.

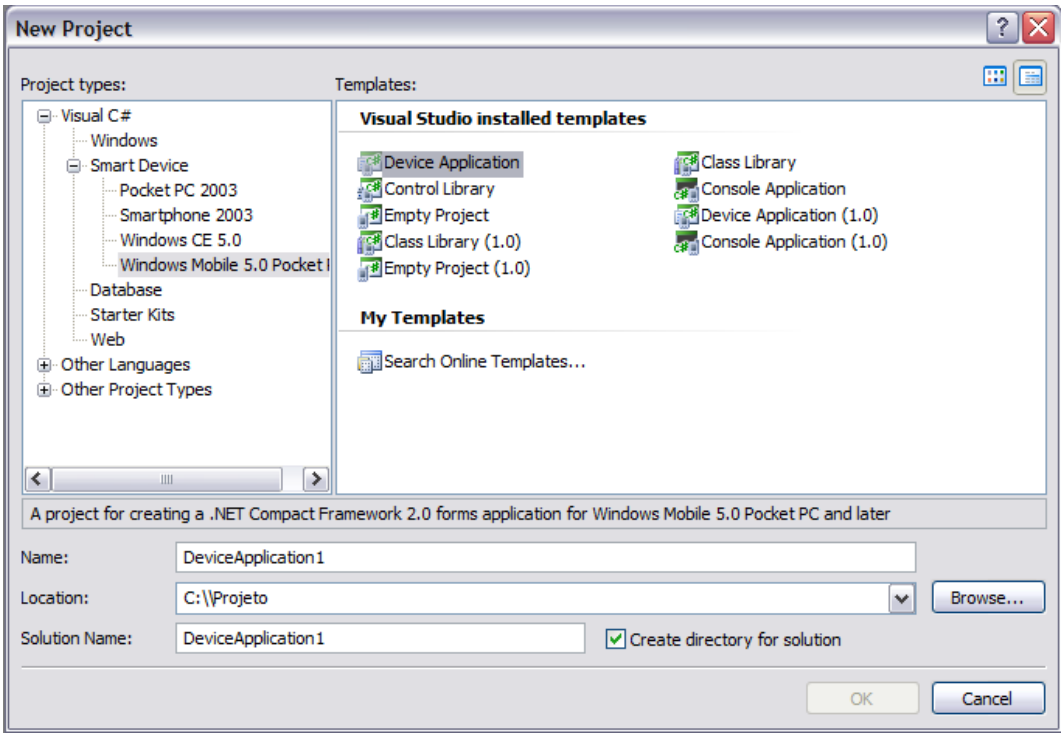

Figura 7 - Exemplos de templates de Projeto no Visual Studio

Essa base em geral se traduz na forma de uma árvore com os arquivos mínimos que devem ser implementados para o projeto em questão. O desenvolvedor pode então seguir a hierarquia de exemplo para desenvolver sua aplicação de acordo com os padrões sugeridos pela mantenedora da *IDE*, no caso, a Microsoft.

O desenvolvimento a partir de árvores de arquivo e hierarquias é importante para a rapidez com que o desenvolvedor utiliza a *IDE*. Cada tipo de aplicação conta com uma árvore diferente, a fim de prover uma melhor semântica para as especificidades que podem ocorrer em projetos diferentes.

Um exemplo prático é a utilização de bibliotecas específicas para aplicações Web, que são inseridas automaticamente quando selecionado o tipo correto de projeto. Outro exemplo seriam aplicações para dispositivos móveis, que tem um conjunto de bibliotecas específicas e diferentes de uma aplicação Web, também configuradas automaticamente quando utilizado o *templates* de projeto correto. A Figura 8 mostra a árvore padrão de arquivos de um projeto para dispositivos móveis, comparada a árvore padrão de arquivos de um projeto para aplicações Web.

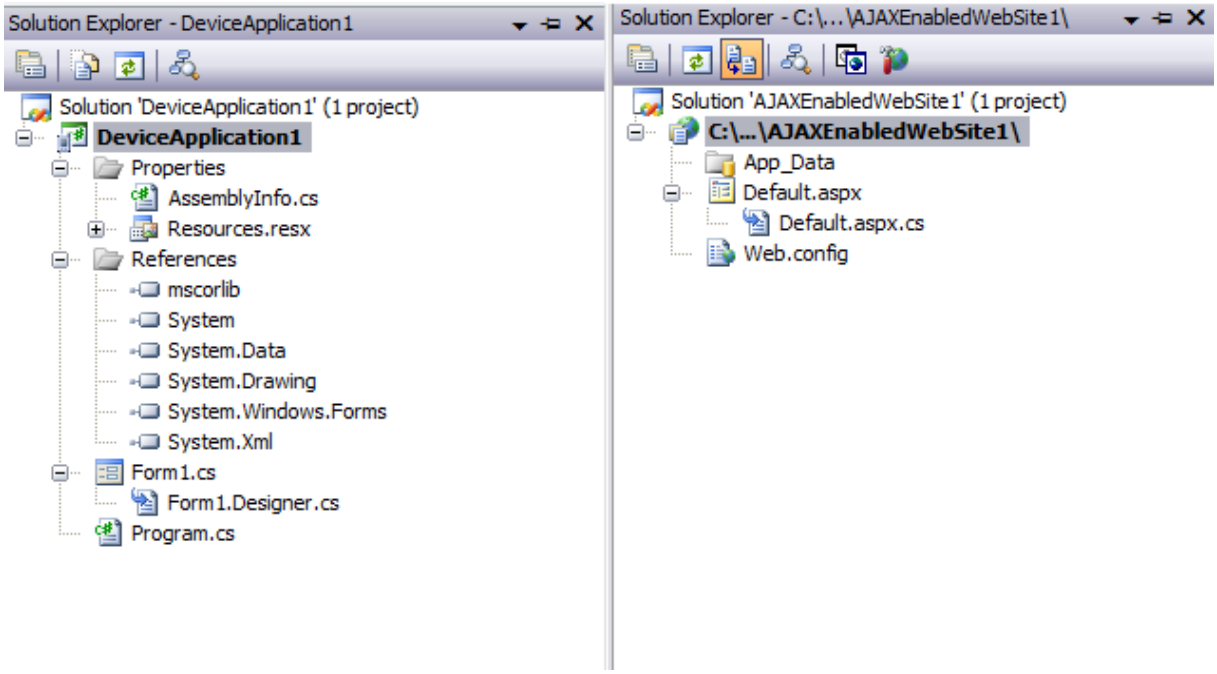

Figura 8 - Árvore de arquivos para projeto de aplicação para dispositivos móveis (esquerda) e para aplicação Web (direita)

É possível ainda criar *templates* de projeto próprios, mantendo todas as funcionalidades dos *templates* disponíveis inicialmente pelo *Visual Studio*. Este

processo pode ser repetido com a adição de outras bibliotecas e componentes, gerando um novo *framework* de desenvolvimento.

O *Visual Studio* prevê esse tipo de necessidade por parte dos usuários, como por exemplo na utilização da mesma estrutura por diversos membros de uma equipe de projeto, provendo funcionalidades que visam estender os *templates* padrões para adaptar ainda mais o ambiente para o usuário.

Como exemplo, pode-se ver a Figura 9, que mostra a disponibilidade de um projeto customizado para criação de um projeto para aplicação Web. Este projeto tem um nome diferente, e uma árvore de arquivos alterada.

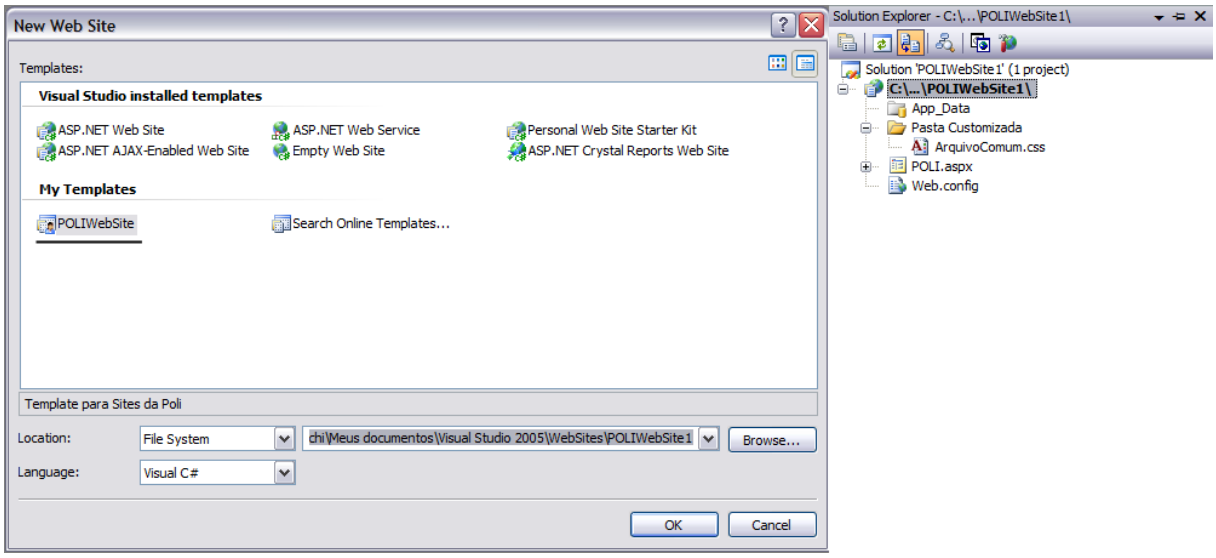

Figura 9 - Template de projeto para aplicação Web customizado.

2.3.2. DESIGN TIME CUSTOMIZADO

O conceito de *Design Time* no *Visual Studio* é referente à utilização de componentes visuais para indicar, ao desenvolvedor, como a aplicação que está sendo desenvolvida será mostrada ao usuário, implementando o conceito de *WYSIWYG* (*What You See Is What You Get*).

Esta indicação não necessariamente é idêntica ao que o usuário final irá encontrar ao utilizar a aplicação, sendo apenas necessário que o desenvolvedor possa ter uma idéia, algo similar ao componente usado, durante o desenvolvimento.

A utilização de *Design Time WYSIWYG* é comumente usada nos tipos de projeto que fazem referência à utilização de interfaces humano-computador rica,

novamente exemplificados por aplicações Web ou aplicações para dispositivos móveis. A Figura 10 mostra a diferença entre a exibição em *Design Time*, a exibição real e a Listagem 1 mostra o código que gera um componente na forma de botão para aplicações Web.

Figura 10 - Exibição Design Time WYSIWYG (esquerda), e exibição real

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Listagem 1 - Código gerador de botão

Assim, como os projetos customizados, o *Design Time* pode ser também estendido, provendo a capacidade de implementar uma exibição diferente aos componentes customizados. O desenvolvedor de componentes customizados pode implementar, então, uma exibição que auxilie o utilizador do componente, não necessariamente mostrando a forma final do seu componente.

2.4. MICROSOFT DIRECT3D MOBILE

O *Direct3D Mobile* é uma *API* da *Microsoft* que fornece suporte para o desenvolvimento de aplicações 3D em dispositivos móveis baseadas no *Windows CE*.

A versão *Mobile* difere da versão de computadores *desktops* por ter eliminado suporte a gráficos 3D, pois as limitações de *hardware* dos dispositivos móveis não permitiriam desenvolver.

O *Direct3D* é um componente do *DirectX*, agindo como uma camada de abstração na comunicação de uma aplicação 3D com o *driver* do dispositivo utilizado. Esta camada de abstração encontra-se no mesmo nível do *kernel* ou do *GDI* do sistema operacional, porém, uma vez que o *Direct3D* se conecta diretamente ao *driver* de vídeo, seu desempenho é melhor que o do *GDI*. A Figura 11 ilustra essa camada no contexto da arquitetura básica de um dispositivo móvel.

Figura 11 - Arquitetura básica de um dispositivo móvel

Os elementos principais da *API* podem ser visto na Figura 12, onde podem ser notados três componentes: *device*, recursos (*resources*) e cadeias de troca (*swap chains*).

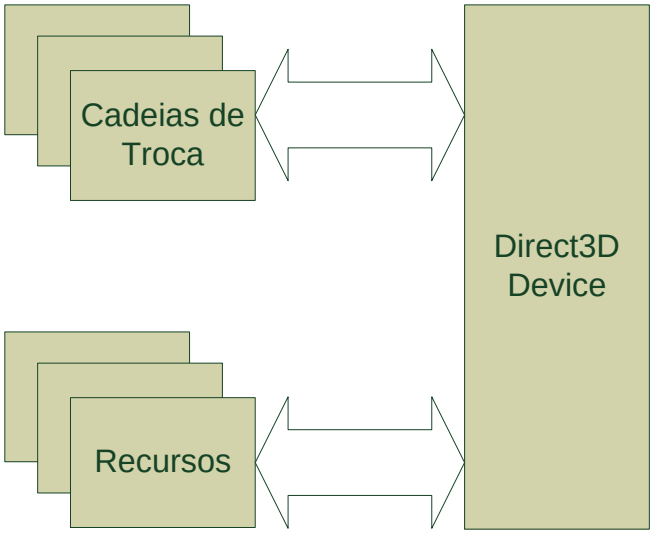

Figura 12 - Principais componentes do Direct3D

O *device*, é o elemento do *Direct3D* responsável por fazer a renderização das cenas 3D, além de disponibilizar opções de tela-cheia ou janela, e tela monocromática ou multicromática.

Os recursos utilizados pelo *Direct3D* podem ser divididos em dois grupos, onde todos os elementos implementam uma mesma interface *IDirect3DMobileResource*. O primeiro deles corresponde aos elementos de texturas que implementam a interface *IDirect3DMobileBaseTexture*, como superfícies, luzes e materiais, responsáveis por disponibilizar os elementos que trazem cores e simulam texturas e luzes em um cena 3D. O segundo grupo corresponde a *buffers* que

contém elementos como, por exemplo, índices e vértices, responsáveis por formar as imagens 3D quando dispostos em conjunto.

As subseções desse capítulo descrevem, em detalhe, os recursos do Direct3D utilizados no projeto de formatura.

O último elemento do *Direct3D* é representado pelas cadeias de troca. Em vez de renderizar a imagem diretamente na tela, o *Direct3D* primeiro desenha as imagens em *buffers* de *pixels*, conhecidos como *front buffer* e *back buffer*, os quais tem seu conteúdo trocado quando há a atualização da imagem a ser visualizada.
Os itens a seguir contêm uma descrição dos elementos do *Direct3D Mobile* mais utilizadas no projeto, os quais encontram-se dentro dos três componentes principais descritos até então, neste capítulo.

2.4.1. DEVICE

O *device* é o componente de renderização, que designa o processo de obtenção de imagens digitais, do *Direct3D*, encapsulando e armazenando cada um dos estados que serão renderizados. Além disso, o *device* realiza operações de transformação e de luzes e rasteriza imagens a uma superfície (transforma a imagem em um mapa de bits).

Este componente é responsável por acessar os *drivers* de vídeo, que são controlados através de um *middleware* que abstrai o dispositivo físico, ou seja, as operações são realizadas sem que se haja a necessidade de se conhecer como as renderizações são enviadas pelo *driver* para o dispositivo físico. Dessa forma, o *device* age como se fosse o próprio dispositivo físico.

Um *device* pode ser criado utilizando a linguagem de programação *C#*, como pode ser visto na Listagem 2:

```
      Device device = new Device(0, DeviceType.Hardware, this,
CreateFlags.SoftwareVertexProcessing, pres);
```

Listagem 2 - Criação de um objeto device.

Os parâmetros utilizados na criação se referem ao índice do adaptador gráfico (primeiro parâmetro, de valor zero), ao tipo de processamento, realizado via hardware (*DeviceType.Hardware*), ao objeto que é responsável pela renderização

(*this*, um *form* do tipo *top-level*), e ao objeto que contém os parâmetros de apresentação (*pres*).

Níveis de Cooperação do Device

Uma das características do *device* é ser responsável pelo controle do nível de cooperação de uma aplicação 3D no dispositivo móvel. Isso significa que, com ele, se pode determinar se a aplicação será visualizada em tela cheia ou em uma janela. Dessa forma, o *device* pode controlar o gerenciamento de vídeo do dispositivo, uma vez que, caso a aplicação seja executada em tela cheia, não haverá o processamento de vídeo das aplicações que estiverem sendo executadas em conjunto.

O nível de cooperação de um *device* deve ser configurado como um parâmetro de apresentação do *device* conforme a Listagem 3:

```
PresentParameters pres = new PresentParameters();
pres.Windowed = true;
```

Listagem 3 - Parâmetros de apresentação

Controle de Recursos

O *device* é também responsável por controlar os recursos disponíveis no *Direct3D*. Dessa forma, com ele podem-se aplicar texturas, materiais, efeitos de luzes, além de controlar as posições das entidades dentro do sistema de coordenadas. Com isso, aumenta-se a realidade dos elementos 3D. Para utilizar uma textura em *devices* deste projeto, pode-se fazer conforme apresentado na Listagem 4:

```
TextureLoader.FromFile(device, "texture.bmp");
```

Listagem 4 - Utilização de texturas

Como pode ser notado, a textura do arquivo *texture.bmp* está sendo aplicada à instancia do *device* utilizando o método estático *FromFile* da classe *TextureLoader*, definida pelo *Direct3D*.

O controle de recursos também inclui o gerenciamento do *buffer* de vértices e do *buffer* de índices, responsáveis por conter os elementos que, em conjunto, dão forma aos elementos 3D, construindo a sua moldura base.

2.4.2. *BUFFER* DE VÉRTICES E *BUFFER* DE ÍNDICES

Um vértice pode ser definido geometricamente como um ponto onde duas arestas de um polígono se encontram.

No *Direct3D*, um vértice é uma estrutura que pode conter mais informações do que simplesmente a sua posição no plano XYZ. Ele pode ter também informações de cor, texturas, transformações, entre outras. A Tabela 2 contém os principais tipos de vértices do *Direct3D Mobile*.

Tabela 2 - Tipos de vértice do Direct3D Mobile

Classe	Conteúdo do Vértice
PositionOnly	Somente informações x, y e z.
Transformed	Somente transformações x e y.
PositionColored	Coordenadas x, y e z e informações de cores do vértice,
TransformedColored	Coordenadas x e y e informações de cores do vértice.
TransformedTextured	Coordenadas x e y e informações de texturas, em coordenadas u e v

Os vértices podem ser combinados utilizando-se um *buffer* de vértices, os quais podem ser renderizados na tela através da instância de *device* utilizada. Dessa forma, fornecendo um *buffer* de vértices ao *device* e informando de que forma organizar os vértices é possível renderizar os modelos 3D desejados. Os principais métodos de organização de vértices são: Lista de Triângulos, Faixa de Triângulos e Leque de Triângulos.

No método de Lista de Triângulos, os vértices são agrupados três a três, desenhando um triângulo para cada tripla, como mostrado na Figura 13.

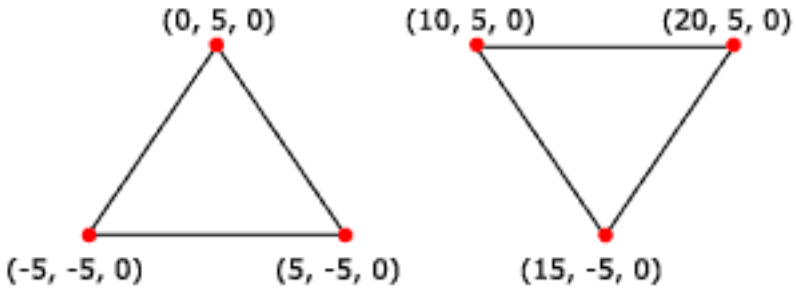

Figura 13 - Lista de Triângulos

Para desenhar triângulos desta forma, configura-se o *device* conforme a Listagem 5:

```
device.DrawPrimitives(PrimitiveType.TriangleList, 0, 2);
```

Listagem 5- Desenho de triângulos por lista de triângulos

Assim, informa-se o *device* para desenhar uma lista de triângulos (*TriangleList*), iniciando pelo elemento 0 do *buffer* de vértices. O último elemento indica que serão desenhadas 2 primitivas, ou seja, 2 triângulos.

Usando-se o método Faixa de Triângulos, ilustrada na Figura 14, os três primeiros vértices formarão um triângulo e, para cada vértices seguinte, um novo triângulo será formado utilizando-se os dois pontos anteriores.

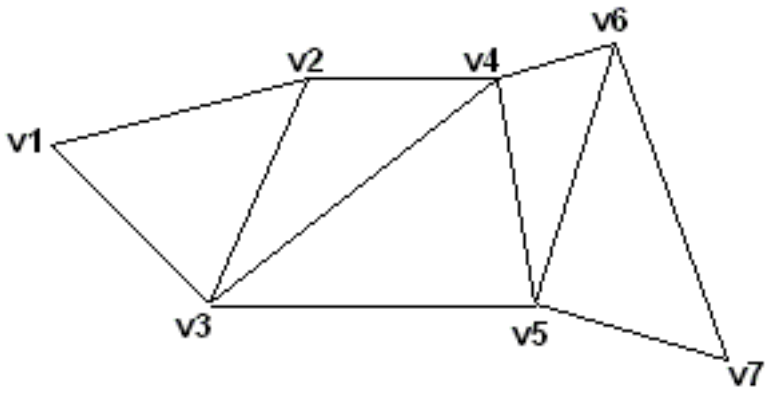

Figura 14 - Faixa de Triângulos

Para desenhar esta primitiva, configura-se o *device* conforme a Listagem 6:

```
device.DrawPrimitives(PrimitiveType.TriangleStrip, 0, 1);
```

Listagem 6 - Desenho de triângulos usando faixa de triângulos

Assim, informa-se o *device* para desenhar uma faixa de triângulos (*TriangleStrip*), iniciando pelo elemento 0 do buffer de vértices. O último elemento indica que será desenhada somente 1 faixa de triângulos.

Por fim, no método Leque de Triângulos, os três primeiros vértices formarão um triângulo e, para cada vértices seguinte, um novo triângulo será formado utilizando-se o primeiro pontos e o anterior. A Figura 15 ilustra esse método.

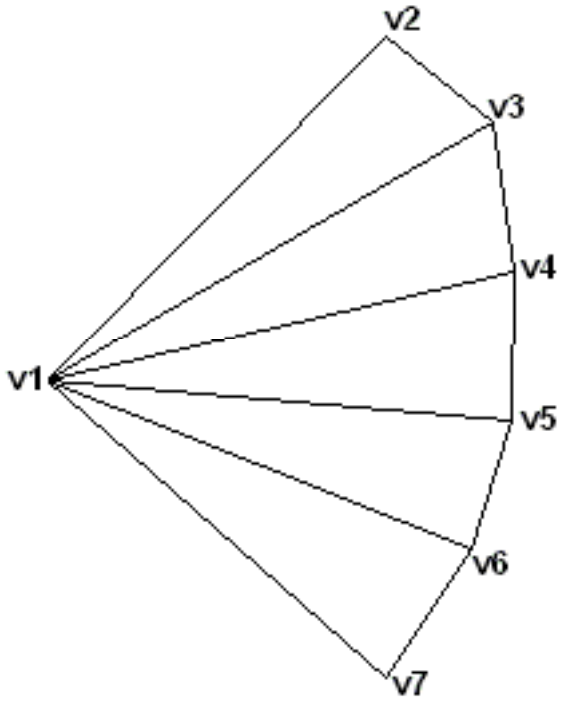

Figura 15 - Leque de Triângulos

Para desenhar esta primitiva, configura-se *device* conforme a Listagem 7:

```
device.DrawPrimitives(PrimitiveType.TriangleFan, 0, 1);
```

Listagem 7- Desenho de triângulos usando leque de triângulos

Assim, informa-se o *device* para desenhar um leque de triângulos (*TriangleFan*), iniciando pelo elemento 0 do buffer de vértices. O último elemento indica que será desenhada somente 1 leque de triângulos.

Os agrupamentos gráficos de triângulos são bastante utilizados pois, a partir das formas geradas e conjuntos de formas, é possível desenhar elementos 3D bastante complexos.

Há, porém, alguns casos em que, para desenhar determinadas formas, é necessário redefinir vértices. Isso, ocorre, por exemplo, quando se deseja desenhar um cubo, como o caso da Figura 16. Para desenhar suas faces Frontal, Lateral Esquerda, Traseira e Lateral Direita pode-se perceber que se poderia utilizar uma Faixa de Triângulos. Porém, seria necessário repetir os dois últimos vértices do *buffer*, uma vez que eles se encontram. Este problema se agrava quando são desenhadas as faces Superior e Inferior, pois é necessário redefinir todos os oito vértices.

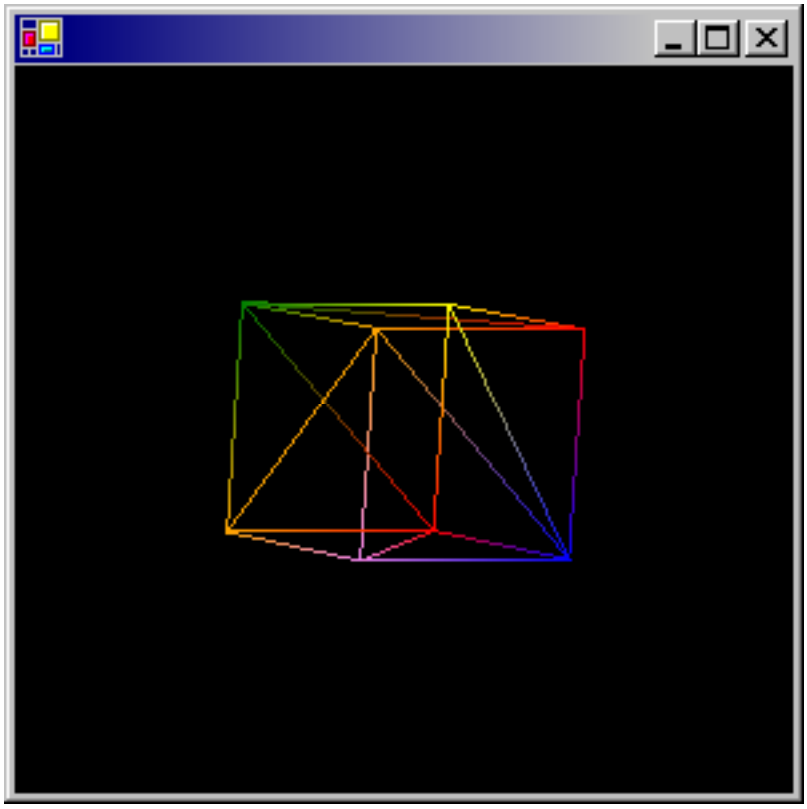

Figura 16 - Visualização dos Vértices na Renderização de um Cubo

Para resolver este problema, utilizam-se os chamados *buffers* de índices. Os *buffers* de índices são seqüências de referências aos vértices disponíveis em um

buffer de vértices. Cada tripla de um *buffer* de índices, com cada índice referenciando a um elemento do *buffer* de vértices, define um triângulo, de modo que um vértice possa ser referenciado mais de uma vez, evitando que haja replicação de vértices. A relação entre um *buffer* de índices e um *buffer* de vértices pode ser vista através da Figura 17.

Dessa forma, o que o *buffer* de índices faz é gerar uma relação normalizada entre as formas a serem renderizadas e os vértices relacionados a eles. Com isso, economiza-se memória do dispositivo móvel, um fator bastante relevante no desenvolvimento.

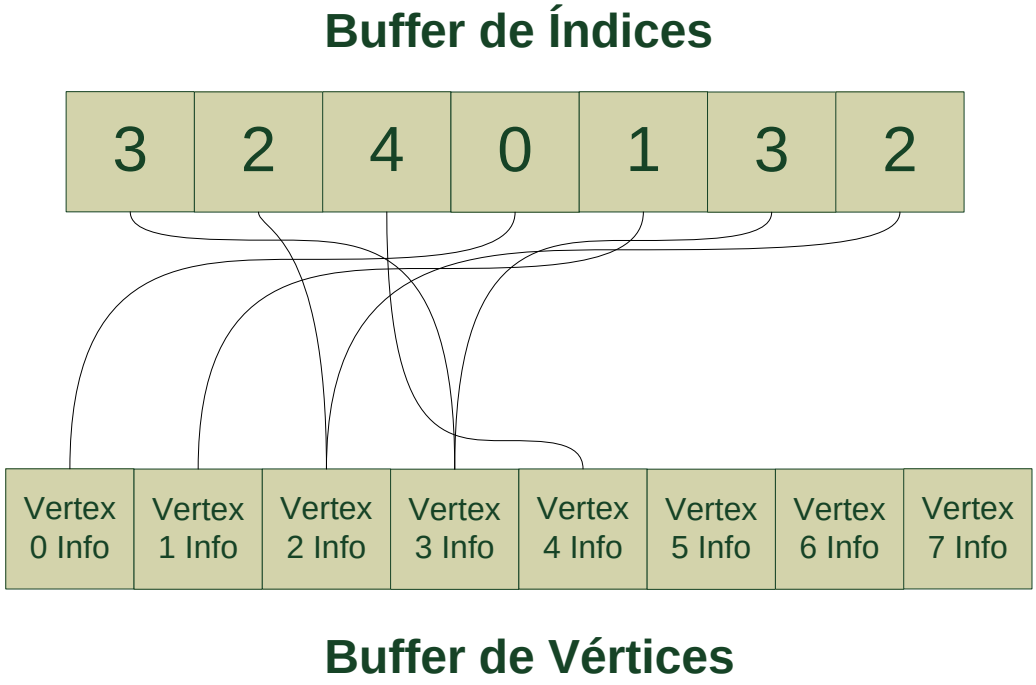

Figura 17 - Relação entre um Buffer de Índices e um Buffer de Vértices

2.4.3. MESHES

Um *meshe* é uma classe que permite agrupar, em um único objeto, um elemento a ser renderizado na tela, através de seu *buffer* de vértices e *buffer* de índices.

O *Direct3D* fornece um grupo de *meshes* para serem utilizados, de modo que elementos básicos possam ser gerados sem a necessidade de escrever a definição de vértices e de índices.

Um *mesh* que representa um paralelepípedo reto-retângulo pode ser criado conforme a Listagem 8:

```
Mesh mesh = Mesh.Box(device, largura, altura, profundidade);
```

Listagem 8 - Criação de *meshe* para paralelepípedo

Como pode ser observado, basta fornecer o *device* utilizado e as dimensões para criar o elemento.

O desenvolvimentos de aplicações 3D é comumente integrado com software de modelamento de objetos tridimensionais, como o software comercial *3D Studio Max*, da *Autodesk*. Estes modelos gerados podem ser carregados na forma de *meshes*, eliminando as etapas de definição de vértices e índices.

Porém, o suporte para carregar os objetos de modelamento no *Direct3D Mobile* é limitado em relação ao *Direct3D*, para aplicações *desktop*, em que isso pode ser feito através de uma simples linha, como mostra a Listagem 9:

```
Mesh mesh = Mesh.FromFile(path, MeshFlags.SystemMemory,
device, out exMaterials);
```

Listagem 9 - Criação de *meshe* a partir de modelos

Na versão *Mobile*, devido às limitações nas classes fornecidas, é necessário implementar métodos para realizar a leitura dos modelos 3D aos *meshes*, estando este desenvolvimento fora do escopo deste trabalho.

2.4.4. SPRITES

Sprites são pequenas imagens que representam objetos bidimensionais em uma cena, contendo atributos, como posição, velocidade e direção, e funções, como mudar sua própria cor. Normalmente os *sprites* são utilizados em jogos bidimensionais e representam, por exemplo, um personagem principal ou um inimigo.

Há dois tipos de *sprites*: estáticos e dinâmicos. Os *sprites* estáticos não são animados, mas são móveis, enquanto os *sprites* dinâmicos são animados e móveis, onde cada um de seus *frames* é obtido a partir de um *bitmap*.

2.4.5. SISTEMAS DE COORDENADAS

Um sistema de coordenadas é uma forma de representar espaços multi-dimensionais, de modo que cada ponto do espaço possa ser representado unicamente. No *Direct3D Mobile*, os sistemas de coordenadas são tridimensionais.

O *Direct3D Mobile* conta com uma forma de representar diversos objetos dentro de uma mesma cena, o que é feito utilizando-se os sistemas de coordenadas locais e globais.

O sistema de coordenadas local é utilizado para representar cada um dos vértices de um dado objeto, relativos a ele próprio. Neste sistema, é comum que um dos vértices do objeto seja a origem do sistema de coordenadas local e, assim, os outros vértices serão dados pelos pontos no plano relativos àquele vértices da origem. Desta forma, cada vértices do objeto estará representado em um sistema de coordenadas próprio.

O sistema de coordenadas global é utilizado para representar a relação entre todos os objetos em uma mesma cena. Desta forma, mapeia-se, no sistema de coordenadas global, a posição da origem de cada um dos objetos, dentro do seu próprio sistema, resultando em sua organização dentro da mesma cena.

No *Direct3D Mobile*, os sistema de coordenadas local e global podem ser implementados utilizando sistemas cartesianos, esféricos, cilíndricos, entre outros.

Além dos sistemas de coordenadas citados tem-se também o sistema de coordenadas de texturas, bidimensional, utilizado para mapear imagens de texturas aos elementos tridimensionais, informando que ponto de uma imagem está associado a cada vértice de um elemento. Este sistema de coordenadas recebe mais detalhes no item 2.4.7, quando Texturas são apresentadas.

2.4.6. TRANSFORMAÇÕES

Transformações são operações matemáticas utilizadas para realizar a conversão de um objeto de um espaço vetorial para outro, realizando mudança de bases para atender as necessidades do contexto em que a operação está envolvida.

Quando se trabalha com programação tridimensional, freqüentemente manipulam-se matrizes para simbolizar os elementos tridimensionais e o ambiente, em relação a um dado sistema de coordenadas global.

Dessa forma, utilizam-se transformações sempre que se desejar realizar qualquer tipo alteração no ambiente ou nos modelos de uma dada cena.

Como exemplos de transformações utilizadas, podem-se citar a Translação, a Escala e a Rotação, detalhadas a seguir.

Translação – Supondo que se quer transladar um ponto (x, y, z) para uma posição cuja movimentação em x seja Tx, em y seja Ty e em z seja Tz, pode-se realizar a seguinte transformação, dada pela Figura 18.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Figura 18 - Transformação de Translação

Escala – Supondo que se quer mudar a escala de um ponto (x, y, z) de modo que x receba escala Sx, y receba escala Sy e z receba escala Sz pode-se realizar a seguinte transformação, dada pela Figura 19.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 19 - Transformação de Escala

Rotação - Supondo que se quer rotacionar um ponto (x, y, z) em torno dos eixos X, Y e Z podem-se realizar as seguintes transformações, dadas, respectivamente, pelas Figuras Figura 20, Figura 21 e Figura 22.

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 20 - Transformação de Rotação em Relação ao Eixo X

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 21 - Transformação de Rotação em Relação ao Eixo Y

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 22 - Transformação de Rotação em Relação ao Eixo Z

Uma característica importante da utilização de matrizes é a possibilidade da combinação de efeitos, que pode ser realizada através da concatenação de matrizes de transformações. Dessa forma, caso se deseje realizar translação e rotação simultâneas, basta multiplicar as matrizes de transformação de translação e rotação, obtendo uma nova transformação a ser aplicada aos pontos do modelo.

2.4.7. TEXTURAS

As texturas são imagens aplicadas aos elementos tridimensionais para que recebam uma superfície que traz a impressão visual de como são vistos no mundo real. Com bastante freqüência as texturas utilizadas nos elementos são extraídas de imagens do próprio mundo real.

As texturas são aplicadas aos modelos a partir de tipos especiais de vértices que podem receber, além de informações de cores e de posição, sua posição dentro do sistema de coordenadas de textura.

As coordenadas de texturas, expressas por u e v, são coordenadas bidimensionais e expressam que ponto de uma imagem associar a cada vértice específico de um elemento. Para que o mapeamento seja feito, a imagem utilizada recebe uma escala onde sua coordenada u cresce horizontalmente, para a direita, e a coordenada v cresce verticalmente, para baixo. Além disso, a imagem base da textura tem como (0, 0) seu canto superior esquerdo e (1, 1) seu canto inferior direito, como mostrado na Figura 23.

Nesta figura tem-se um exemplo em que se quer aplicar uma textura a um triângulo. Como mostrado, é necessário especificar qual ponto da imagem será utilizado em cada vértice do triângulo, em coordenadas (u, v). Assim, o triângulo recebe o resultado mostrado.

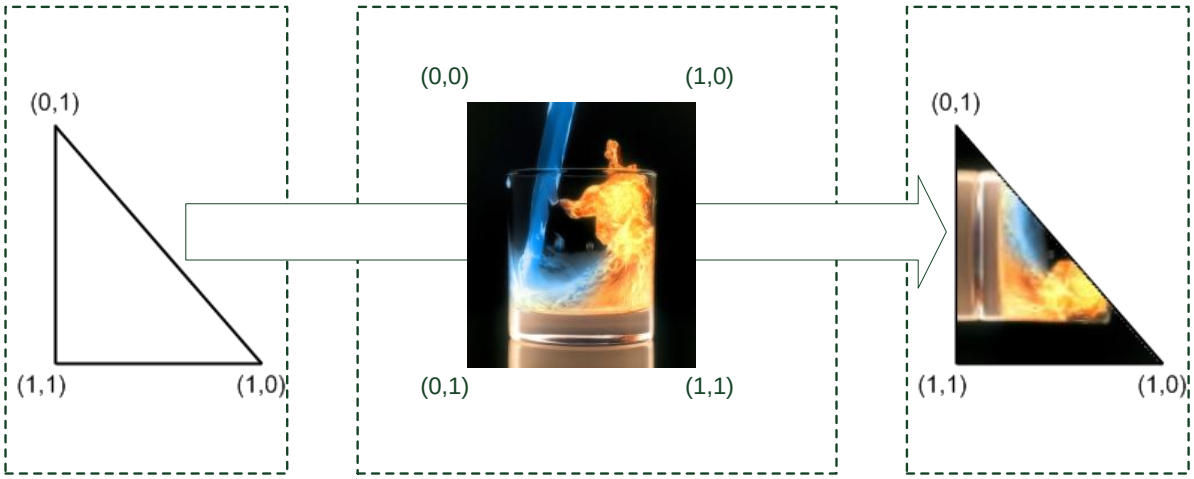

Figura 23 - Aplicação de Textura a um Triângulo

2.5. Considerações Finais

Este capítulo descreveu os principais conceitos teóricos e tecnologias envolvidas neste projeto de formatura A compreensão do conteúdo dos itens é essencial para o desenvolvimento do trabalho realizado, sendo a obtenção deste conhecimento um dos principais desafios encontrados.

3. PROTÓTIPO

O projeto tem como objetivo o desenvolvimento de um protótipo para a aplicação do conceitos analisados no projeto de formatura. O protótipo consiste de uma aplicação em que serão desenvolvidos componentes 3D para a integração com componentes nativos do *.NET Compact Framework.*

O protótipo desenvolvido terá como base o uso das ferramentas proporcionadas pelo Direct3D 10.0. O Direct3D é um pacote da Microsoft que fornece ferramentas para facilitar a criação de objetos tridimensionais, que serão a base da criação dos componentes do projeto.

Para a utilização do protótipo será necessário um computador com sistema operacional *Microsoft Windows XP* e ter instalado o software *Microsoft Visual Studio 2005*, necessário para o desenvolvimento de aplicativos da linha *Microsoft*.

3.1. ESPECIFICAÇÃO DE REQUISITOS

As funcionalidades e componentes previstos no protótipo são:

- *Criar Projeto Utilizando Componentes 3D* – Função responsável pela criação de um novo sistema utilizando os Componentes 3D.
- *Adicionar Placa 3D* – Função responsável pela adição do componente Placa com Duas Faces.
- *Adicionar Tela Cúbica* – Função responsável pela adição do componente Tela Cúbica.

Os principais requisitos não funcionais que o sistema deve atender estão listados abaixo

- *Usabilidade* – O protótipo deve ser desenvolvido de modo que a facilitar o aprendizado. A configuração dos componentes no projeto deve ser feita tentando facilitar a interação do usuário com o projeto.

- *Confiabilidade* – A confiabilidade do sistema deve-se principalmente na confiabilidade dos componentes disponibilizados, de modo que esses

componentes tenham, pelo menos, a mesma confiabilidade garantida pelos componentes que foram utilizados como base. Além disso, deve-se garantir que as funcionalidade de outros componentes nativos do .NET Compact Framework 2.0 não sejam afetadas pelos componentes 3D.

- *Desempenho* – O desempenho do protótipo também é critico nos componentes desenvolvidos. Por se tratar de componentes que serão utilizados em dispositivos móveis, deve-se garantir um desempenho em que o usuário não se sinta desconfortável com o sistema.

A especificação de requisitos completa se encontra no Apêndice A.

3.2. ARQUITETURA DE SOFTWARE

A arquitetura de software do protótipo tem como objetivo nortear a estrutura a ser utilizada para o desenvolvimento dos componentes. Os componentes por sua vez têm como característica principal serem utilizados em conjuntos com componentes nativos presentes no *.NET Compact Framework*, e portanto, é esperado que os novos componentes desenvolvidos provam grande parte das funcionalidades presentes nos componentes nativos.

Os componentes desenvolvidos no protótipo podem ainda ser comparados a outros componentes nativos semelhantes quanto a sua funcionalidade, se diferenciando pela apresentação visual tridimensional. Um exemplo é o componente Cubo, que funcionalmente pode ser comparado com o componente nativo de abas (*tabs*).

Levando em consideração estes pontos, foi definida uma arquitetura de software que foca o reuso das classes provenientes do *.NET Compact Framework*, por meio de extensão de classes, conceito abordado na programação orientada a objetos.

Usando como *superclasse* a classe Control do pacote System.Windows.Forms, mesmo não havendo acesso ao código fonte dessa classe, os componentes desenvolvidos no protótipo herdam propriedades que provêm características semelhantes ao dos componentes nativos, como atributos de

identificação, posicionamento e acesso a eventos (clique, foco, entre outros). Ao utilizar a classe Control como superclasse de todos os componentes também foi possível manter a compatibilidade dos componentes quanto a listas de tipos específicos, que comportam objetos de tipo Control. Isso foi possível, pois todos os controles nativos também herdam a classe Control.

Além da classe Control, a classe Form, do mesmo pacote System.Windows.Forms, também é utilizada no protótipo como superclasse da classe de formulário customizada que fornece suporte à utilização dos componentes tridimensionais. Essa classe customizada também aproveita das vantagens de herdar a classe nativa do *framework*, como os componentes em relação à classe Control. Um ponto importante é que essa classe de formulário mantém o atributos Controls, uma coleção que comporta objetos do tipo Control, ao mesmo tempo em que provê atributos importantes para o funcionamento dos componentes tridimensionais, como por exemplo, o acesso a uma instância única da classe Device.

Como boa prática definida na arquitetura, foi definido ainda que classes de apoio fossem separadas dos módulos referentes à construção de componentes. Essa prática auxilia na reutilização de métodos comuns a mais de um componente desenvolvido nesse protótipo, tendo como exemplo a classe Capture, abordada com detalhes em no item a seguir.

Os componentes criados no projeto, juntamente com as classes de apoio, foram agrupados em um projeto separado, de nome D3D.Windows.Forms. Dessa maneira, para a utilização dos novos componentes, é necessário apenas referenciar um arquivo, que contém a forma compilada do projeto.

A Figura 24 mostra a representação arquitetura utilizada.

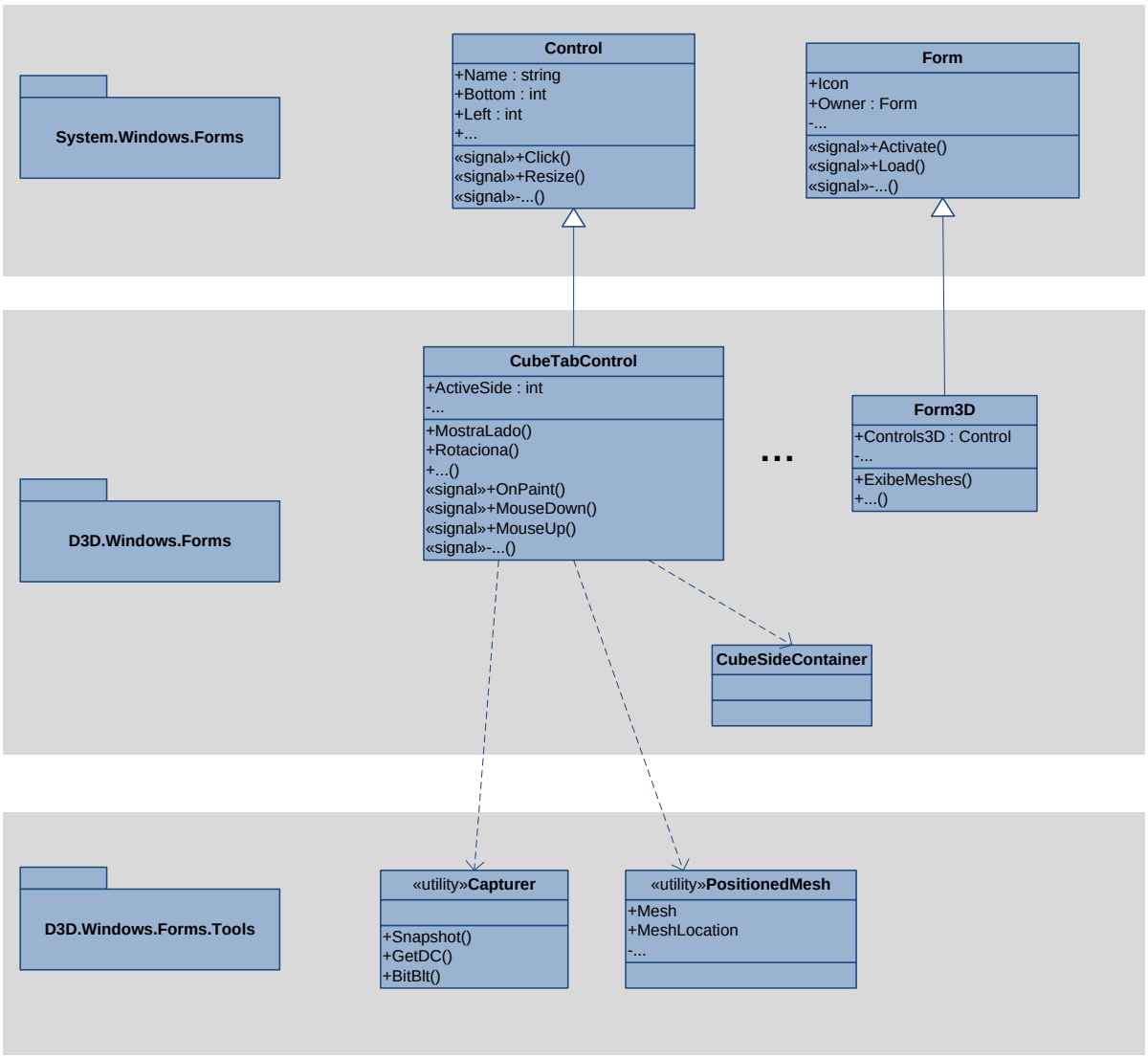

Figura 24 - Arquitetura do Sistema

3.3. SELEÇÃO DE TECNOLOGIA

A primeira fase realizada no projeto de formatura foi o desenvolvimento de provas de conceito (*POCs*) com o intuito de selecionar a tecnologia a ser utilizada para a análise de viabilidade e, conseqüentemente, para o desenvolvimento do protótipo.

O *POC* foi feito tendo como alternativas duas ferramentas: *Microsoft XNA* e o *Direct3D Mobile*, que são, atualmente, as tecnologias mais citadas no mercado quando se trata de desenvolvimento 3D para dispositivos móveis.

Como resultado, esta análise mostrou qual das duas ferramentas possui maior compatibilidade com o escopo desejado e, assim, tem maior chance de alcançar o sucesso da análise de viabilidade de integração.

3.3.1. MICROSOFT XNA

O *Microsoft XNA* é uma nova tecnologia da Microsoft que tem por objetivo substituir o *Managed DirectX* para o desenvolvimento de jogos. Uma vez que a plataforma tem sua versão compatível com o *.NET Compact Framework*, resolvemos realizar esta prospecção tecnológica, buscando verificar a possibilidade de se utilizar esta tecnologia para da análise de viabilidade deste projeto.

O objetivo desta análise foi verificar a adaptação do *Microsoft XNA* na versão *Personal Digital Assistant* (*PDA*) do *.NET Compact Framework*, de modo que fosse possível diagnosticar o resultado na criação de componentes baseados nas ferramentas tridimensionais oferecidas pela plataforma.

Uma vez que se trata de uma nova tecnologia, considerou-se interessante esta análise, de modo a adequar o projeto às novidades disponíveis no mercado.

3.3.1.1. RESULTADOS DA ANÁLISE DO MICROSOFT XNA

Após analisar tecnologia e tentar integrá-la aos dispositivos móveis (*PDA's*) não se obteve um resultado satisfatório. De acordo com pesquisas realizadas, citadas logo adiante, embora o *Microsoft XNA* utilize o *.NET Compact Framework*, a versão utilizada é específica para o desenvolvimento de jogos para o console de videogame *Xbox*, também da *Microsoft* (*.NET Compact Framework for Xbox*). Sendo assim, não é possível utilizar o *Microsoft XNA* no projeto, pois a versão do *.NET Compact Framework* deveria ser para *PDA*'s.

Esta análise pôde ser confirmada após a leitura do *XNA Frequently Asked Questions (FAQ)*, no *site* da Microsoft:

"(...)

Q: Does the XNA Framework support Windows and Pocket PC devices? *A: The XNA Framework currently does not support Windows Mobile or Pocket PC*

devices, but based on customer feedback this may be a direction we expand the XNA Framework in the future. We know that developing mobile games is a hot area of growth and one we would like to support in the future.
(…)".

Apesar disso, como citado no *FAQ*, há uma intenção por parte da *Microsoft* em tornar disponível a possibilidade de utilização do *Microsoft XNA* em dispositivos *PDA*, o que pode tornar esta análise interessante em uma futura versão.

Desta forma, esta tecnologia foi descartada como uma alternativa para o projeto, uma vez que o foco é o estudo da integração de componentes 3D que deverá ser utilizado em *PDA's* e esta tecnologia não se adapta a este tipo de dispositivo.

3.3.2. DIRECT3D MOBILE

O *Direct3D Mobile*, detalhado na seção 2.4, foi testado de modo que o grupo pudesse adquirir conhecimento com a ferramenta através do desenvolvimento de POCs que mostrem a eficiência da ferramenta nos PDA's.

O grupo sabia da possibilidade de desenvolvimento de jogos utilizando esta tecnologia para os *PDA's*. Sendo assim, as provas de conceito realizadas seriam feitas com o intuito de conhecer algumas dificuldades provenientes deste uso, de modo a poder ter diretivas que indicassem que esta era uma boa ferramenta ser utilizada no projeto.

Além disso, os testes também foram interessante, no sentido de avaliar o desempenho da ferramenta, o qual também poderia ser um fator restritivo para a escolha.

3.3.2.1. RESULTADOS DA ANÁLISE DO DIRECT3D MOBILE

A partir das pesquisas realizadas foi aferido que o *Direct3D Mobile* pode ser utilizado para realizar a análise de viabilidade de criação de componentes no *.NET Compact Framework*, como desejado. Tal conclusão foi possível visto que a versão

móvel do *Direct3D* foi desenvolvida justamente para dispositivos móveis que possuem pouca memória e hardware limitado.

O ambiente de desenvolvimento escolhido foi o *Visual Studio 2005* visto que esse já prevê o uso do *framework* de desenvolvimento *.NET Compact Framework* que é necessário para o uso do *Direct3D*.

Além disso, o *.NET Compact Framework* também provém o uso da *DLL* (*Dynamically Linked Library*) *Microsoft.WindowsMobile.DirextX.dll* que contém todas as funcionalidades do *Direct3D Mobile*.

Para a distribuição e testes, o *Visual Studio 2005* é integrado com o *Windows Mobile*, sendo assim mais fácil de implementar os componentes.

Foram feitos alguns exemplos utilizando o *Direct3D* no *.NET Compact Framework*, ilustrados nas Figuras Figura 25, Figura 26Figura 27 e Figura 28, para mostrar a utilização do *Direct3D Mobile* sobre o *.NET Compact Framework.*

Figura 25 - *ImageButton* via Direct3D

Figura 26 - *ImageButton* pressionado

Os *ImageButtons* das Figuras 25 e 26 foram gerados a partir das bibliotecas do *Direct3D* para .NET Compact Framework. Embora este componente criado não tenha características visuais tridimensionais ele foi desenvolvido com a mesma tecnologia.

Figura 27 - Paisagem via Direct3D

Figura 28 - Paisagem via Direct3D

As paisagens das Figuras Figura 27 e Figura 28 foram geradas a partir do *Direct3D* para *.NET Compact Framework*, mostrando parte da potencialidade da tecnologia: apresentar modelos 3D satisfatoriamente na interface gráfica do *PDA*, utilização de texturas, aplicadas no solo e no céu, e utilização de diversos modelos simultaneamente, que no caso foi feito gerando-se um número aleatório de árvores de tamanho variado no solo. Através deste exemplo, pode-se obter a indicação do desempenho necessário para o desenvolvimento da análise de viabilidade.

Sendo assim, foi escolhida a tecnologia do *Direct3D Mobile*, que apresenta as características necessárias para o desenvolvimento do projeto. Também foi levada em consideração a possibilidade de utilização do ambiente de desenvolvimento provido pelo *Visual Studio 2005*, onde o nível de conhecimento do grupo é alto.

3.4. DESENVOLVIMENTO DO CUBO 3D

O Cubo 3D é um componente que tem por objetivo substituir os chamados *TabContainers*, que são componentes de *PDA's* que permitem agrupamentos de componentes em abas de um formulário. Dessa forma, o Cubo 3D forneceria o mesmo agrupamento de componentes de uma forma diferente, que seria dando a possibilidade do usuário girar o cubo arrastando o *stylus* (nome dado a popular "caneta" dos PDA's) sobre a tela do *PDA*, alterando assim a face do cubo visualizada, a qual possuiria um conjunto de componentes. A Figura 29 mostra um

exemplo de uso de um TabContainer clássico em um sistema de automação de força de venda, onde há três abas: Pedidos, Notas Fiscais e Produtos.

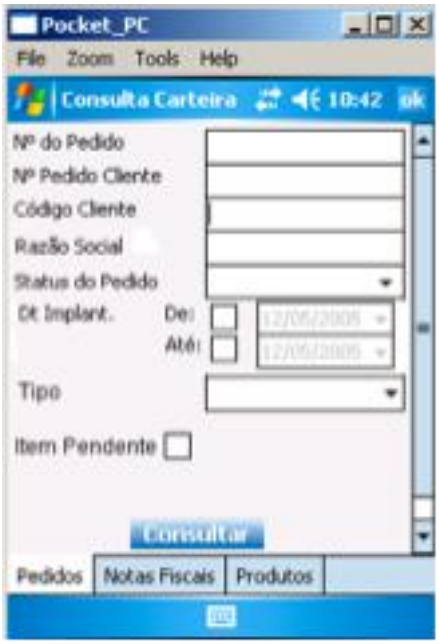

Figura 29 - Exemplo de Uso de TabContainer

3.4.1. CONSIDERAÇÕES INICIAIS

A decisão de iniciar por este componente surgiu do fato de que todos os outros a serem desenvolvidos deveriam ser integrados com o Cubo 3D, ou seja, o usuário poderia adicionar a ele qualquer componente, até mesmo os tridimensionais desenvolvidos neste protótipo. Assim, era importante se este componente estivesse pronto.

3.4.2. DIFICULDADES INICIAIS

As dificuldades do projeto iniciaram logo no momento da criação do primeiro componente, uma vez que o grupo não tinha conhecimento de como criar uma classe que pudesse representar o cubo e ao mesmo tempo ser adicionada como componente de um formulário do *.NET Compact Framework*.

De modo geral, quando uma aplicação utilizando o *Direct3D Mobile* é desenvolvida, é de costume fazer com que ela rode sempre dentro de um único formulário. Dessa forma, sempre que há a troca dos elementos 3D exibidos na tela

não há mudança de formulário, mas sim do grupo de *meshes* que são desenhados e das texturas que são aplicadas. No caso de um jogo, também há movimentação de *sprites* mas, de qualquer forma, o formulário utilizado não se altera.

Dentro deste contexto, não se pode fazer com que o *Cubo 3D* fosse uma classe herdada de um formulário, pois ele deveria ser um controle a ser adicionado em qualquer formulário de uma aplicação, possibilitando que o fluxo entre formulário aconteça da forma que as aplicações são implementadas hoje em dia. Caso se alterasse isso, o desenvolvimento de aplicações ficaria limitado, o que não era a intenção do projeto.

Este problema ocorre devido ao fato de que toda aplicação baseada no *Direct3D Mobile* necessita de uma instância de um *Device* (detalhado no item 2.4.1) para que possa renderizar gráficos. O *Device*, por sua vez, deve ser criado tendo como um de seus parâmetros um formulário.

Neste ponto, tomou-se uma decisão muito importante para o projeto, que impactou bastante nas atividades, mas que trouxe os resultados desejados: foi criada uma classe de formulário, própria do projeto, chamada de *Form3D*.

3.4.3. O DESENVOLVIMENTO DO FORMULÁRIO 3D

A idéia do Formulário 3D é implementar toda a infra-estrutura necessária para que os novos componentes possam ser adicionados a ele. Dessa forma, a classe *Form3D* possui internamente uma instância da classe *Device*, utiliando como parâmetro o próprio formulário. Dessa forma, todo componente 3D adicionado ao formulário poderá fazer uso do *Device* de sua classe pai, que no caso será o formulário.

Um detalhe importante é que o Formulário 3D criado também deveria ser compatível com os componentes nativos do *.NET Compact Framework*. Porém, este detalhe não trouxe dificuldades, uma vez que herdar a classe *Form3D* da classe *Form* padrão não causou problemas.

A versão inicial do Formulário 3D contou simplesmente com uma forma para proporcionar um Device aos componentes. Assim, a classe Form3D foi desenvolvida tendo a seguinte variável membro, disponibilizada através de propriedades aos componentes, conforme a Listagem 10:

```
private Device _device;

public Device Device
{
	get { return _device; }
	set { _device = value; }
}
```

Listagem 10 - Variável membro Device

Além disso, foi desenvolvido um método de inicialização do Device, o qual é chamado dentro do construtor do Form3D, conforme a Listagem 11:

```
protected void InitializeDevice()
{
	PresentParameters presentParams = new PresentParameters();
	presentParams.Windowed = true;
	presentParams.SwapEffect = SwapEffect.Discard;

	try
	{
		_device = new
		Microsoft.WindowsMobile.DirectX.Direct3D.Device(0,
		DeviceType.Default, this, CreateFlags.None,
		presentParams);
	}
	catch
	{
		MessageBox.Show("Erro de inicialização do Device");
	}
	_device.RenderState.CullMode = Cull.None;
	_device.RenderState.Lighting = false;

	}
```

Listagem 11 - Inicialização Form3D

Com isso, obteve-se o formulário 3D desejado, que deu a possibilidade de prosseguir com o projeto.

3.4.4. DESENHO DO CUBO 3D

O próximo passo do projeto foi o desenvolvimento do desenho do cubo, ou seja, foi feito o seu mapeamento de vértices e índices de modo que pudessem ser renderizados em forma de cubo.

A idéia do projeto seria desenhar o cubo e associar a ele dois tipos de entidades: outros componentes 3D, como a Placa 3D, descrita no item 3.5; ou então

um componente nativo. Dessa forma, quando o usuário arrasta o *stylus* sobre a tela, o conjunto de componentes exibido é alterado.

Além disso, uma importante consideração feita pelo grupo é que no momento em que o cubo está girando seja possível perceber os componentes girando juntamente com a face do cubo. Assim, a forma que as faces seriam desenhadas seria importante, pois poderia facilitar neste procedimento.

A primeira versão do desenho do cubo contou com somente o mapeamento de vértices e índices, sendo estes elementos adicionados diretamente no *device* do formulário. Um exemplo de código de geração de vértices e índices pode ser visto na Listagem 12.

```
// Criação de vértices
CustomVertex.PositionTextured[] arrayVertices =
new CustomVertex.PositionTextured[4];

arrayVertices[i++] = new CustomVertex.PositionTextured(
      -1.6F, 1.6F, -1.6F, 1, 0);
arrayVertices[i++] = new CustomVertex.PositionTextured(
      1.6F, 1.6F, -1.6F, 0, 0);
arrayVertices[i++] = new CustomVertex.PositionTextured(
      -1.6F, -1.6F, -1.6F, 1, 1);
arrayVertices[i++] = new CustomVertex.PositionTextured(
      1.6F, -1.6F, -1.6F, 0, 1);

//Criação de índices
short[] arrayIndices = new short[6];

arrayIndices[i++] = 0; arrayIndices[i++] = 3; arrayIndices[i++] =
1;
arrayIndices[i++] = 0; arrayIndices[i++] = 2; arrayIndices[i++] =
3;


//Adiciona os vértices e índices no device e apresenta a cena.
device.BeginScene();
device.SetStreamSource(0, vertices, 0);
device.Indices = indices;
device.EndScene();
device.Present();
```

Listagem 12 - Criação de vértices e índices para o Cubo3D

Com o passar do tempo, foi verificado que é uma boa pratica na programação 3D a utilização de *meshes* para armazenar conjuntos de informações deste tipo, uma vez que uma entidade pode conter partes com diferentes texturas, luzes e outros efeitos, o que é o caso do cubo 3D. O Cubo, ao ser rotacionado, deve somente alterar a textura de uma das faces do cubo (a face a ser escondida), não

alterando a textura das demais faces. Dessa forma, cada instância do cubo 3D passou a ter um conjunto de *meshes*, cada um deles representando uma das faces.

O código da Listagem 13 mostra como foi populado o *meshe* da face de um cubo.

```
mesh = new Mesh(arrayIndices.Length / 3, arrayVertices.Length,
MeshFlags.SystemMemory,
      CustomVertex.PositionTextured.Format, _device);


mesh.VertexBuffer.SetData(arrayVertices, 0, LockFlags.None);
mesh.IndexBuffer.SetData(arrayIndices, 0, LockFlags.None);
```

Listagem 13 - *Meshe* da face de um cubo.

Como resultado deste procedimento, obteve-se um cubo que tem a possibilidade de ter alterada a imagem de suas faces, o que facilitará em seus efeitos de rotação.

3.4.5. ROTAÇÃO DO CUBO 3D

Para realizar a rotação do cubo 3D foi feito uso de eventos do formulário que manipulam o momento em que o usuário clica com o *stylus* no *PDA* e também o momento que o usuário solta o *stylus*. Dessa forma, bastou identificar se houve uma variação horizontal entre os dois pontos identificados pelos eventos. Além disso, este mecanismo também permitiu identificar para qual lado o cubo foi girado, sendo possível fazer a animação 3D de acordo com o exato ato de arrastar o stylus.

Porém, como dito, estes eventos são captados pelos formulários e não pelo componente Cubo 3D. Dessa forma, o formulário deve identificar qual o componente que foi clicado (no caso o cubo) e, identificado isso, deve chamar o evento correspondente do cubo, que o faz girar. A dificuldade envolvida, então, é envolver o componente clicado, uma vez que a classe *Form3D*, que herda da classe *System.Windows.Forms* do *.NET Compact Framework*, identifica somente os pontos do formulário que foram clicado e solto.

A solução encontrada para identificar o componente foi armazenar no formulário uma lista de *meshes*, e suas respectivas posições, em uma classe criada chamada de *PositionedMeshe*. Assim, o formulário contém uma lista de

PositionedMeshe. Dessa forma, ao ocorrer o evento, o código realiza, para cada *PositionedMeshe*, uma interseção entre um plano criado pelo *meshe* e uma reta perpendicular ao plano que passa pelo ponto clicado. Caso haja a interseção, o código identifica que houve o clique, podendo então repassar o evento ao componente que contém o *meshe*. O código que realiza este procedimento para o evento *OnMouseUp* (que ocorre ao soltar o *stylus*) é mostrado na Listagem 14.

```
protected override void OnMouseUp(MouseEventArgs e)
{
      Vector3 nearVector = new Vector3(e.X, e.Y, 0);
      Vector3 farVector = new Vector3(e.X, e.Y, 1);

      for (int i = 0; i < PositionedMeshes.Count; i++)
      {
            nearVector.Unproject(_device.Viewport,
                  _device.Transform.Projection,
                  _device.Transform.View,
                  Matrix.Translation(PositionedMeshes[i].MeshLocation));

            farVector.Unproject(_device.Viewport,
                  _device.Transform.Projection,
                  _device.Transform.View,
                  Matrix.Translation(PositionedMeshes[i].MeshLocation));

            farVector.Subtract(nearVector);

            // Realiza o teste de interseção entre o plano e a reta
            if ( Geometry.BoxBoundProbe(
            PositionedMeshes[i].MeshBoundingBoxMinValue,
            PositionedMeshes[i].MeshBoundingBoxMaxValue, nearVector,
            farVector))
            {
                  if (i < this.Controls3D.Count)
                  {
                        ((CubeTabControl)this.Controls3D[i]).MouseUp(e);
                  }
                  break;
            }
      }
}
```

Listagem 14 - Tratamento do evento *OnMouseUp*

Uma vez identificado que o cubo deve ser rotacionado, iniciou-se o desenvolvimento do código que faz o seguinte procedimento: translada o cubo para trás, realiza a rotação de 90 graus no sentido cujo *stylus* foi arrastado e, por fim, faz o translado para frente.

Este procedimento realiza esta tarefa alterando as coordenadas globais do observador, ou seja, do usuário, através de funções do *Direct3D Mobile* de

manipulação de matrizes. Assim, basta utilizar os métodos *RotationY* e *Translation* da classe *Matrix* com os parâmetros que identifiquem o resultado esperado.

Para combinar os efeitos de translação e de rotação foi utilizado o conceito de que para realizar duas transformações lineares seguidas, basta multiplicar suas matrizes de transformação. Assim, as matrizes de rotação e de translação geradas foram multiplicadas.

Ao alterar a matriz de coordenadas globais, foi necessário apresentar o resultado para que o usuário veja o efeito. Porém, como a rotação e translação têm de serem feitas gradualmente, de modo que o efeito seja observado, a rotação e translação foram realizadas em passos, de modo a apresentar também situações intermediarias ao usuário.

3.4.6. CAPTURA DE TELA

Cada face do cubo deve ter associada a ela um conjunto de componentes que são exibidos quando a face está na posição frontal do cubo. Assim, ao rotacionar o cubo os componentes atuais devem ficar ocultos (isto será tratado no item 3.4.7), porém uma imagem da tela deve ser associada como textura do cubo para que haja a impressão de que os componentes estão girando junto com a face.

Para realizar isto, foi criada uma classe chama de *Capturer* que tem a função de fazer uma captura da tela e salvar em um arquivo temporário um *bitmap* que representa a tela. Esta imagem é, então, carregada e seus *bytes* são convertidos em textura para que possa ser aplicada a face que deixará de ser ativa.

O *.NET Compact Framework* fornece classes que facilitam o processo de captura, sendo necessário, porém, utilizar DLLs de interoperabilidade do PDA, ou seja, de manipulação em funções que manipulam, por exemplo, a memória utilizada.

Uma dificuldade enfrentada pelo grupo foi quanto à obtenção de uma imagem que representasse a próxima face que será exibida, uma vez que esta imagem deveria ser obtida em memória, e não por captura de tela, já que a face não foi visualizada ainda. Esta dificuldade ocorre uma vez que, como a tela pode conter tanto componentes 3D quanto os nativos, ocorre uma dificuldade de obter uma imagem que represente os dois tipos de componentes juntos. O grupo conseguiu obter uma imagem que representa os componentes 3D que serão mostrados, mas

não uma imagem que os unifica com os componentes nativos que serão mostrados. Dessa forma, não houve sucesso em encontrar uma solução e este requisito foi retirado da rotação do cubo 3D, restando simplesmente a imagem de captura da tela que deixará de ser exibida.

3.4.7. TROCA DOS COMPONENTES EXIBIDOS

Quando o usuário arrasta o *stylus* sobre a tela, além de haver a rotação do cubo também devem ser alterados quais os componentes que serão mostrados.

Dentro deste contexto existe ainda um agravante, que é o fato de não se pode alterar a visibilidade dos componentes, uma vez que esta é uma propriedade definida pelo usuário, ou seja, ele pode querer que um componente fique escondido no formulário, ao invés de ser exibido.

Assim, a primeira solução adotada foi guardar todos os componentes de cada face em uma estrutura separada que pudesse conter também a visibilidade dos componentes, ou seja, só se tornariam visíveis os componentes da face que realmente deveriam estar visíveis.

Porém, esta solução trouxe um problema: a utilização de uma estrutura separada para armazenar cada componentes juntamente com sua visibilidade não tornaria possível a criação do *design time* do componente, já que quando se arrasta um componente para um formulário ele é automaticamente adicionado à propriedade *Controls* do formulário e não à estrutura específica do Cubo.

Assim, foi descoberta uma forma de resolver isto sem afetar o *design time*: o cubo 3D teria quatro *CuboSideContainers*, classe criada para representar uma face do cubo, dentro da propriedade *Controls* do cubo. Cada face, por sua vez, teria uma série de componentes dentro de sua propriedade *Controls*. Para resolver o problema de visibilidade, todas as faces devem ficar sempre invisíveis, ou seja, nenhum componente é exibido. Quando uma face tiver de ser exibida, todos os componentes deixam de ter como classe pai o *CubeTabContainer* e passam a ter como pai o formulário (*Form3D*), passando a utilizar a sua própria visibilidade dentro do formulário. Ao trocar a face a ser exibida, os componentes retornam a sua classe pai, ou seja, para a sua face, e deixam o formulário. A seguir há uma representação desta organização de classes:

Form3D

Cube3D

CubeTabContainer1 (sempre invisível)

Control1 (sua própria visibilidade)

Control2 (sua própria visibilidade)

...

CubeTabContainer2 (sempre invisível)

...

No momento de exibição de uma face, passa-se a ter o trecho a seguir:

Form3D

Control1 (sua própria visibilidade)

Control2 (sua própria visibilidade)

...

Cube3D

CubeTabContainer1 (sempre invisível)

(controles migraram para o formulário)

CubeTabContainer2 (sempre invisível)

...

3.5. PLACA 3D

Com o início do desenvolvimento do Cubo 3D, descrito na seção anterior, grande parte dos conceitos necessários para a criação da placa 3D já tinham sido analisados.

Porém, alguns ajustes se mostraram necessários, pois as propriedades utilizadas no Cubo 3D não se ajustavam com o componente da placa 3D.

O desenvolvimento se iniciou com o aproveitamento de funções criadas para o desenvolvimento do cubo, ou seja, a criação de camadas dos *meshes* e sua integração com o *Form* 3D foram utilizados para iniciar o desenvolvimento da placa, visto que tais propriedades seriam comuns a todos os componentes.

Em seguida, tornou-se necessária a criação do desenho da placa, que deveria ser um retângulo com duas faces. A intenção desde o início era a utilização de um *meshe*, o que pouparia tempo na criação dos vértices do componente.

Neste ponto, descobriu-se a existência da criação de *meshes* em formatos comuns, ou seja, bastante utilizados no desenvolvimento de qualquer entidade, que são em forma de cubo, triângulo, entre outros. Na Listagem 15 é apresentado um exemplo da criação de um *meshe* em forma de caixa, e na Figura 30 a sua renderização.

```
      _mesh    =    Mesh.Box(_form,    _floatWidth,    _floatHeight,
_floatDepth);
```

Listagem 15 - *Meshe* em forma de caixa

Vale ressaltar que esta técnica não foi utilizada no componente Cubo 3D uma vez que, embora seja uma forma muito mais fácil de se criar um cubo, somente uma textura pode ser aplicada a um *meshe* e, desta forma, todas as faces do cubo teria a mesma textura. Por isso, optou-se que cada face fosse representada por um *meshe*.

Para o retângulo da placa esta técnica foi bastante útil, uma vez que um *meshe* no formato de cubo, porém com profundidade zero, representou bem a placa.

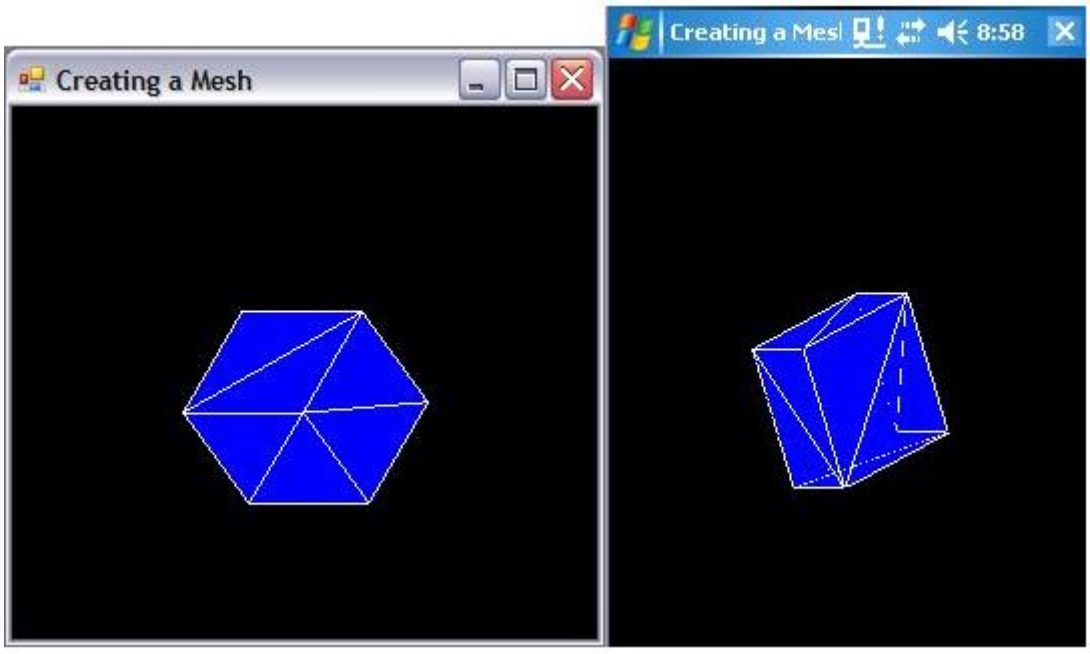

Figura 30 - Renderização de *Meshe* criado pelo método Mesh.Box

Uma alternativa para o desenho da placa seria o mapeamento de seus vértices, porém essa solução não seria viável, uma vez que para cada tamanho diferente de lados seria necessário o mapeamento de todos os vértices novamente.

Após a decisão de utilizar o *meshe* com profundidade nula, o segundo desafio era conseguir colocar algum texto sobre o *meshe*. Após pesquisas e estudos, foi decidido utilizar *sprites* para escrever textos sobre o componente, como apresentado na Figura 31.

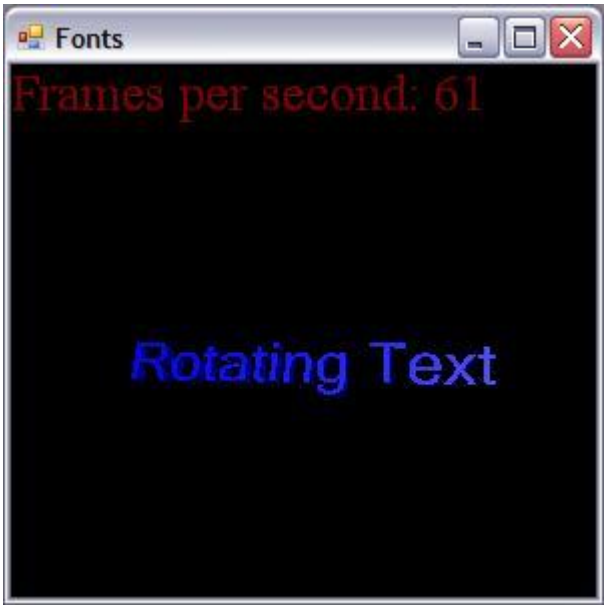

Figura 31 - Textos em Formas de *Sprites*

O *sprite*, como visto na seção 2.4.4, é uma classe que representa um elemento móvel (animado ou não). Através desta classe foi possível criar textos com propriedades de rotação, translação, etc.

Portanto, foi decidido colocar um *sprite* sobre o *meshe* para a rotação do texto juntamente com o componente 3D.

Porém, com tal decisão, outro problema surgiu no desenvolvimento. O sistema de coordenadas utilizado pelos componentes do *sprite* é 2D, enquanto que o *meshe* trabalha no sistema de coordenadas 3D, além da orientação de seus eixos ser diferente, como pode ser visto na Figura 32.

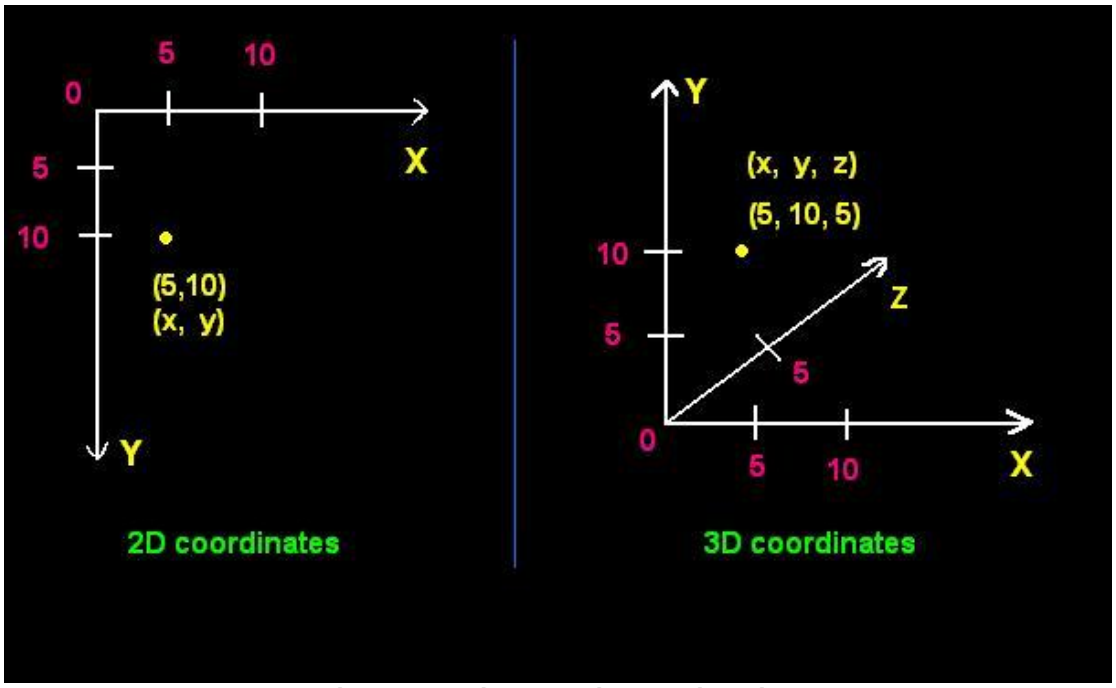

Figura 32 - Sistemas de Coordenadas

A solução, então, foi a conversão do sistema de coordenadas 3D para coordenadas 2D. Após conseguir tal conversão, percebeu-se que a métrica do sistema 2D com o 3D também era diferente, gerando diferenças na percepção de rotação, devido à diferença de perspectiva entre o *sprite* e o *meshe*. Para a correção desse problema foi necessário novamente realizar conversão e ajustes na métrica dos dois sistemas de coordenadas.

Com estes problemas resolvidos a próxima etapa foi mapear todos os pontos da placa e criar um algoritmo que divide a placa em duas áreas úteis, em que o usuário utiliza para rotacionar a placa, mudando a face selecionada, utilizando para isso eventos de clique.

O último passo do desenvolvimento foi a parametrização das propriedades listadas a seguir:

Tamanho da placa:

- Largura
- Comprimento

Posição da placa (2D):

- Coordenada X
- Coordenada Y

Textos da placa (2 posições):

- Texto 1
- Texto 2

O principal método que o componente contém é a ação de rodar a placa e mudar a opção do componente.

Após o término do desenvolvimento, testes unitários foram executados e por fim, testes integrados com outros componentes 3D e componentes nativos foram feitos para garantir a integração.

Quanto à utilização do componente, a grande vantagem da placa 3D é o espaço livre obtido quando comparado com a utilização de outro componente semelhante como um *checkbox* ou um *combobox*.

A extensão prevista para o componente é a possibilidade de se utilizar N alternativas para o componente, em vez de duas.

3.6. DESIGN TIME CUSTOMIZADO

Uma das vantagens ao desenvolver aplicações móveis utilizando o *.NET Compact Framework* é a possibilidade de utilização do Visual Studio 2005. Como apresentado anteriormente, há o conceito de *Design Time*, que, em tempo de criação, permite ao desenvolvedor ter uma prévia do comportamento visual de sua aplicação, facilitando o seu trabalho.

No caso geral, para gerar a prévia do visual da sua aplicação, o *Visual Studio* utiliza o mecanismo de renderização padrão, contido no *event handler* OnPaint de cada componente utilizado. Por esse motivo, para componentes nativos, a visualização é muito semelhante ao que será visto pelo usuário final.

Quando se utilizam componentes tridimensionais, com o uso especifico do *Direct3D Mobile*, a visualização em *Design Time* fica comprometida, devido à alteração do mecanismo de desenho descrito pelo OnPaint. A renderização dos componentes tridimensionais criados no protótipo utilizam métodos diferenciados, fazendo com que o Visual Studio não consiga apresentar em tempo de desenvolvimento uma prévia visual dos novos componentes, gerando um erro na visualização.

Foi utilizado no projeto, então, a diretiva **NETCFDESIGNTIME**, usado da forma apresentada pela Listagem 16, dentro da implementação do método OnPaint.

```
protected override void OnPaint(PaintEventArgs e)
{

        #if NETCFDESIGNTIME
           // Código de Renderização para tempo de desenvolvimento
        #else
           // Código de Renderização para tempo de execução
        #endif

}
```

Listagem 16 - Diretiva NETCFDESIGNTIME

Usando a diretiva dessa forma, o ambiente de desenvolvimento pode utilizar um código de renderização diferente do código para execução, de tal maneira que não haja erros na visualização dos componentes.

Utilizando-se esse método, no entanto, a visualização não é totalmente fiel ao visual apresentado ao usuário final. No caso deste protótipo, apenas uma representação simples do componente foi utilizada para ser apresentada em tempo de desenvolvimento, sendo possível sua utilização com componentes nativos mesmo em *design time*.

Um ponto importante é que a utilização da diretiva **NETCFDESIGNTIME** é funcional apenas quando a classe que a utiliza se encontra dentro de um projeto separado, de tipo *Class Library*. Os projetos deste tipo geram como saída uma DLL, e por isso o *Visual Studio* consegue utilizar as declarações condicionais da diretiva. Ao utilizar a diretiva fora desse tipo específico de projeto, a diretiva não é levada em consideração, o que gerou certa dificuldade na sua utilização durante o desenvolvimento do projeto.

A diretiva **NETCFDESIGNTIME** resolveu o problema de separação de códigos de visualização em desenvolvimento e de execução. Porém, ainda havia a dificuldade de se criar uma interface que pudesse ter uma boa interação com o usuário em momento de desenvolvimento.

3.6.1. UTILIZAÇÃO DE USERCONTROLS

O *.NET Compact Framework* permite que os controles sejam herdados da classe *UserControl*, em vez de *Control*. Herdando um controle de *UserControls*, pode-se gerar um controle que seja composto de outros controles. Um exemplo de controle utilizando UserControls pode ser visto na Figura 33, na qual se pode notar que um único controle contém cinco controles: dois rótulos, duas caixas de texto e um botão.

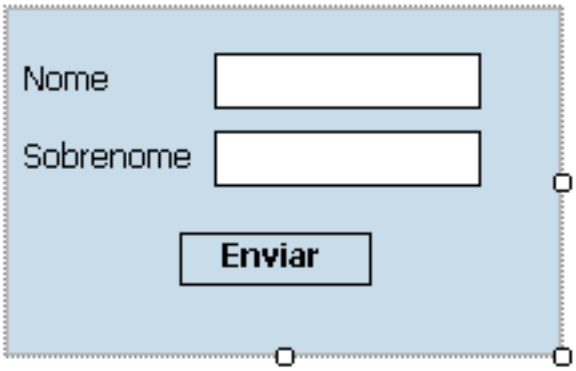

Figura 33 - UserControl

Assim, para criar a interface dos controles 3D em *Design Time* criou-se um controle utilizando rótulos, cada um simbolizando uma das faces do cubo (desconsiderando as faces superior e inferior). A idéia é que, ao clicar em cada um dos rótulos, é apresentada uma face diferente ao usuário, a qual pode receber um grupo de controles.

Como no caso do projeto o controle é desenvolvido com um conteúdo que utiliza o Direct3D Mobile, ao tentar renderizá-lo em *Design Time* não se conseguiu visualizá-lo, pois mesmo que o conteúdo 3D esteja dentro da diretiva **NETCFDESIGNTIME** o Visual Studio considera isso como não confiável. Assim, para estes casos pode-se forçar a renderização do controle através da criação de um arquivo ***DesignTimeAttributes.xmta*** no mesmo projeto dos controles.

3.6.2. O ARQUIVO DESIGNTIMEATTRIBUTES.XMTA

O arquivo ***DesignTimeAttributes.xmta*** é utilizado para editar os atributos dos controles em *Design Time*. Assim, o arquivo possibilita a criação de novas propriedades e eventos, adicionar valores padrão para cada propriedade, limitar a visualização delas, entre outras alternativas. Através deste arquivo pode-se forçar que um controle seja exibido no *Design Time*, mesmo que o seu conteúdo não seja considerado como confiável pelo Visual Studio. A Listagem 17 mostra o código que torna possível a visualização da interface criada para o Cubo 3D utilizando UserControls.

```
<?xml version="1.0" encoding="utf-16"?>
<Classes xmlns="http://schemas.microsoft.com/VisualStudio/2004/
03/SmartDevices/XMTA.xsd">
      <Class Name="D3D.Windows.Forms.CubeTabControl">
            <DesktopCompatible>true</DesktopCompatible>
      </Class>
</Classes>
```

Listagem 17 - Exemplo de arquivo DesignTimeAttributes.xmta

Analisando a listagem pode-se observar a *tag* DesktopCompatible, com conteúdo *true*. Com esta *tag*, o controle definido em D3D.Windows.Forms.CubeTabControl pode ser visualizado em *Design Time*. Este arquivo também possibilitou que algumas propriedades de edição, como o tamanho do cubo, fossem limitadas. Com este procedimento, foi possível criar em *Design Time* a interface do Cubo 3D. A Figura 34 mostra o resultado da interface.

Figura 34 - Interface em Design Time utilizando DesignTimeAttributes.xmta

O uso de UserControls facilita a criação de controles, mas ainda não resolveu todos os problemas, pois ainda não houve a possibilidade de arrastar controles para dentro do cubo. Ao arrastar um controle para dentro dele, o controle é adicionado ao formulário, em vez de dentro do cubo 3D, ou seja, os UserControls não possibilitam a criação de controles que possam receber outros controles, mesmo que os controles adicionados ao UserControl sejam do tipo que possam receber outros controles.

Uma alternativa testada foi o uso de eventos como OnMouseUp e OnMouseDown, que poderiam mostrar pontos clicados sobre o controle. Porém, estes eventos não funcionam em *Design Time*, mas somente em *runtime*.

Após muito estudo foi descoberto o conceito de classe Designer associada a um controle. A classe Designer fornece uma série de eventos para serem utilizados em *Design Time*, como eventos que detectam se um controle foi arrastado ou clicado.

3.6.3. UTILIZAÇÃO DE UM DESIGNER ASSOCIADO AO CONTROLE 3D

Para utilizar uma classe Designer, que representem o controle em *design time* para o usuário, é necessário criar um projeto a parte do projeto dos componentes. Essa necessidade é decorrente do fato que o projeto dos componentes é específico para componentes móveis, e para que a classe Designer funcione corretamente no Visual Studio, é necessário que o projeto em que essa classe seja *Desktop*.

Ao criar uma classe que implementa a interface IDesigner têm-se duas opções mais comuns de classes para herdar: ControlDesigner e ParentControlDesigner, sendo que a última permite que o controle criado possa agir como um container, ou seja, possa conter outros controles.

Para mapear o Designer ao controle desejado utiliza-se, novamente, o arquivo ***DesignTimeAttributes.xmta***, conforme mostrado na Listagem 18.

```
<?xml version="1.0" encoding="utf-16"?>
<Classes xmlns="http://schemas.microsoft.com/VisualStudio/2004/03
/SmartDevices/XMTA.xsd">
  <Class Name="D3D.Windows.Forms.UserControl1">
      <Class Name="D3D.Windows.Forms.CubeTabControl">
            <DesktopCompatible>true</DesktopCompatible>
            <Designer>

      <Type>D3D.Windows.Forms.Designer.CubeTabControlDesigner,
D3D.Windows.Forms.Designer</Type>
            </Designer>
      </Class>
</Classes>
```

Listagem 18 - Associação de Designer em Controle

Ao implementar esta classe foi resolvida boa parte dos problemas, uma vez que se temo agora um classe que recebe os eventos em *Design Time* e também possibilita que o Cubo 3D receba controles.

3.7. PROJETO CUSTOMIZADO

Outro conceito utilizado para a integração com o Visual Studio foi o de projeto customizado. O Visual Studio, por padrão, já apresenta um tipo de projeto para aplicações móveis.

A partir desse tipo de projeto padrão, é possível utilizar os componentes tridimensionais descritos anteriormente. Porém, foi desenvolvido um tipo de projeto customizado, facilitando a utilização dos novos componentes devido à série de customizações necessárias para que os componentes se comportassem de maneira adequada (como a utilização de classes para formulários adequadas).

Dado que todos os componentes tridimensionais criados estão contidos no mesmo projeto, quando compilados geram apenas um arquivo de saída. Esse arquivo de saída, uma *DLL* de nome D3D.Windows.Forms e uma classe de *bootstrap* simples, responsável pela instanciação das classes de formulário, foram encapsulados na forma de *template* de projeto. Esse *template* serve como base de projeto para todos os outros que venham a utilizar os componentes desenvolvidos.

```
namespace Poli3DMobile
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [MTAThread]
        static void Main()
        {
            Form3D form = new Form3D();
            Application.Run(form);
        }
    }
}
```

Listagem 19 - Classe de *Bootstrap*

Para criar o *template* de projeto customizado, foi utilizada a ferramenta de exportação presente no *Visual Studio*. Ao se escolher a opção "***Export Template***", é apresentado um modelo passo-a-passo para auxílio na criação de *templates* de projeto. São apresentadas opções para modificar o ícone de identificação, o nome e a descrição do *template*.

Com o final das configurações, a própria IDE encapsula os arquivos referentes ao projeto de referência para a criação do *template* em um arquivo compactado de extensão **zip**, e salva este arquivo em uma pasta padrão para que o

novo tipo de projeto seja reconhecido e disponibilizado para criação de projetos que levem como base esse *template*, como mostrado na Figura 35.

A ferramenta acessada ao escolher a opção de exportação de *templates* é de fácil uso, porém a configuração do Visual Studio para seu acesso pode ser confusa. Essa opção é exibida ao usuário de forma dinâmica dentro do menu principal *File*. Sendo assim, a exibição da opção é intermitente, se configurada a *IDE* em modo padrão, onde em alguns momentos a opção não fica disponível, sendo omitida. Esse sistema é interessante, pois funcionando corretamente, mostra apenas as ações realmente disponíveis ao desenvolvedor.

Em alguns momentos, no entanto, o *Visual Studio* não disponibiliza essa opção mesmo quando ela é válida. Para o uso da funcionalidade de exportação durante o projeto, foi alterada a configuração do ambiente de desenvolvimento da equipe, configurando-a para sempre mostrar a opção. Dessa maneira, pode-se gerar com facilidade os *templates* de projeto, a ser utilizado como apresentado na Figura 36.

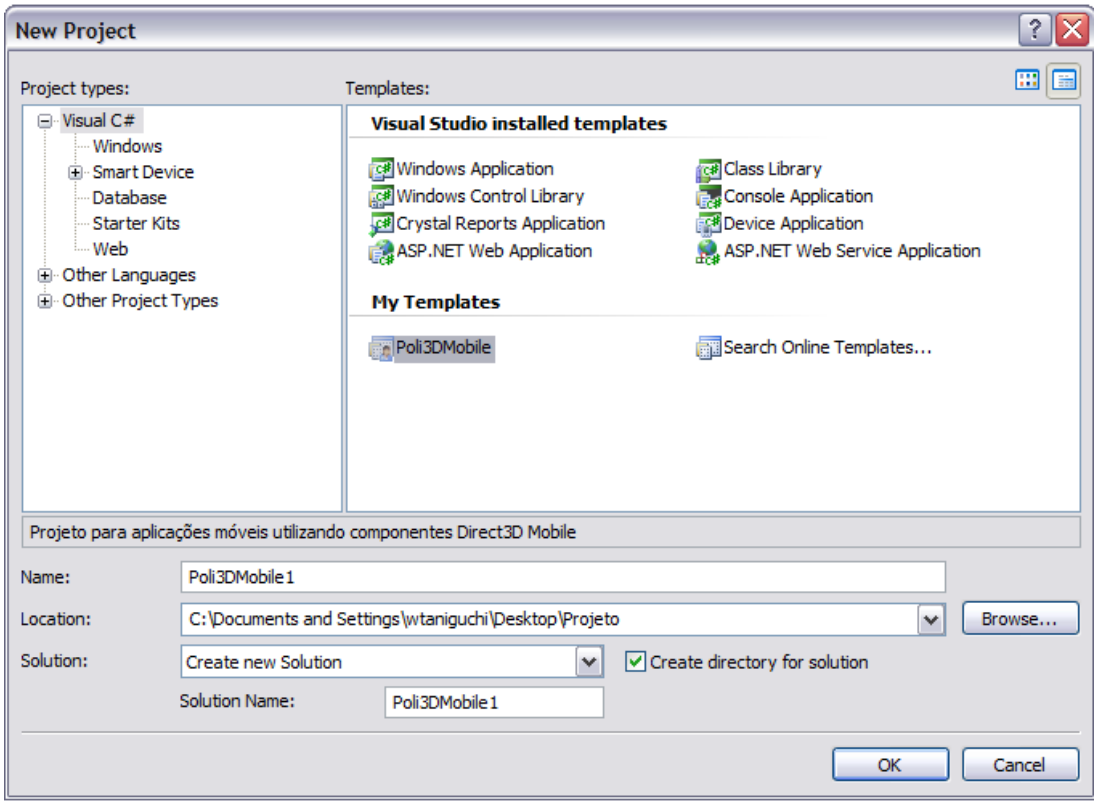

Figura 35 - Template de projeto disponivel para utilização

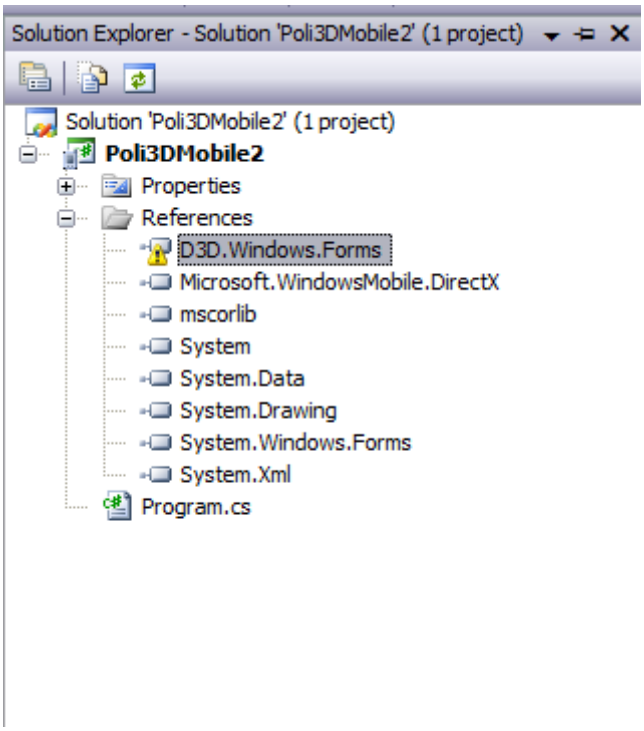

Figura 36 - Árvore de arquivos com referência aos componentes Direct3D

3.8. CONSIDERAÇÕES FINAIS

O desenvolvimento do protótipo, desde sua concepção, com a realização de provas de conceito, se mostrou um processo trabalhoso. A aplicação de conceitos estudados nas fases iniciais de projeto foi feita com sucesso, apesar de problemas decorrentes pelo aumento da complexidade geral dos componentes.

Ao final do processo de desenvolvimento, foram criados, para fins de protótipo, dois componentes. Esse componentes, Cubo3D e Placa3D, são alternativas tridimensionais aos componentes nativos TabControl e RadioButton, respectivamente. Estes componentes aproveitam as capacidades providas pelo Direct3D Mobile, como rotações, translações, provendo um visual diferenciado com relação aos componentes nativos; além disso, ofereceu um ganho de espaço de tela, característica importante para sistemas de dispositivos móveis como os PDA's.

A integração destes novos componentes com componentes nativos em uma mesma aplicação se mostrou uma tarefa complicada. Um dos fatores que aumentou a complexidade dessa tarefa foi a diferença do sistema de coordenadas utilizado pelos componentes nativos (2 dimensões) com o sistema utilizado pelo *Direct3D*,

além da necessidade da utilização de uma classe de formulário especialmente desenhada para esse fim.

Foi também levada em consideração no protótipo a utilização destes componentes por desenvolvedores que, por ventura, se interessem. A maneira com que o desenvolvedor trabalha com os novos componentes é importante, levando ao estudo e desenvolvimento do comportamento dos componentes em design time e também ao desenvolvimento de um pacote de instalação dos componentes.

4. CONSIDERAÇÕES FINAIS

4.1. CONCLUSÕES

O estudo de viabilidade proposto neste trabalho foi baseado no desenvolvimento dos componentes tridimensionais em forma de protótipo, e pode-se afirmar que, no escopo definido, a implementação foi concluída satisfatoriamente.

Após testes envolvendo algumas das tecnologias disponíveis, selecionou-se, o *Direct3D Mobile* como mecanismo de apresentação do espaço tridimensional. Há diversas fontes de apoio para o uso dessa tecnologia porém, como descrito com detalhes em seções anteriores, foi necessário um conhecimento específico em relação à plataforma móvel, fazendo com que o esforço para a implementação correta fosse grande.

A utilização freqüente de técnicas e métodos avançados e não triviais, como a de diretivas de compilação, criação de projetos customizados e mudança de sistemas de coordenadas entre componentes serviu como grande aprendizado, complementando a formação teórica recebida, com experiência prática na resolução de problemas complexos.

Para efeito de protótipo, foi necessário também a preocupação com a utilização dos componentes por parte do desenvolvedor de aplicações móveis, pois o uso dos novos componentes é ligado à sua integração com os componentes nativos, tanto em tempo de desenvolvimento quanto em execução.

Um ponto importante é que apesar desse trabalho ter preterido tecnologias mais recentes como o *Microsoft XNA*, estas podem se mostrar alternativas interessantes a partir do momento em que estiverem maduras e disponíveis abertamente para a comunidade desenvolvedora, melhorando a qualidade dos componentes tridimensionais.

4.2. CONTRIBUIÇÕES

A monografia e a parte prática desenvolvida pretendem deixar uma contribuição futura para os projetos destinados aos *PDA*'s utilizando o *.NET Compact Framework*.

O estudo serve de base para o desenvolvimento de componentes 3D para esta plataforma, integrando com componentes nativos e com o ambiente de desenvolvimento *Visual Studio 2005* da *Microsoft*.

Não se conhece muitos componentes ou plataformas de desenvolvimento para aplicações comerciais utilizando o *.NETCF*. Este estudo visa estimular tal desenvolvimento, abrindo a possibilidade de desenvolvedores começarem a criar componentes para o melhor aproveitamento de espaço e recursos do PDA.

Quanto à contribuição para os integrantes do grupo, este estudo possibilitou principalmente o aprendizado de diversas linguagens de programação, além de aplicação de metodologias aprendidos no decorrer do curso de Engenharia de Computação.

Além disso, foi necessário um grande esforço na parte de pesquisa para o estudo efetuado, capacitando os membros do grupo para esta área tão importante na carreira de engenheiro da computação.

O conhecimento adquirido durante o processo de pesquisa e desenvolvimento do projeto foram principalmente:

- Estudo sobre o *.NET Compact Framework* – O estudo sobre a plataforma da Microsoft foi necessário, pois foi a linguagem escolhida para o estudo da integração. Foi necessário descobrir como os componentes eram executados nesta plataforma, para que assim fosse possível criar a arquitetura necessária para o funcionamento dos componentes 3D.

- Estudo sobre o *Direct3D Mobile* – A ferramenta foi escolhida para o modelagem e criação de componentes 3D, integrado com o *.NET Compact Framework*. Estes componentes serviram para testes durante o desenvolvimento e para a criação do protótipo do projeto.

- Estudo sobre o *Visual Studio 2005* – O objetivo do estudo era que, além de ser possível construir componentes que seriam compatíveis com os componentes nativos, fosse possível também a utilização da ferramenta *Visual Studio* da *Microsoft* para a criação de projetos pelo

desenvolvedor. Para isso foi necessário o estudo sobre como os componentes eram utilizados, ou seja, como eram adicionados no projeto, como suas propriedades eram definidas, etc.

- Estudo sobre o *Windows Mobile 5.0* – Devido ao fato de que o *.NETCF* roda em cima da plataforma *Windows Mobile*, foi necessário um estudo para determinar as funcionalidades e as limitações do desenvolvimento de componentes para este sistema operacional.

- Utilização de técnicas de Engenharia de Software – Utilização de técnicas vistas na aula de Engenharia de Software, tanto em documentação, como no modelagem da arquitetura escolhida.

- Utilização de técnicas de Gerenciamento de Configuração – Para o desenvolvimento de componentes e dos documentos, foram utilizadas técnicas de gerenciamento de configuração vistas na disciplina de Gerência e Qualidade de Software, como criação de versões, cronograma, repositório, etc.

4.3. TRABALHOS FUTUROS

Tendo em vista o estudo da integração, que foi o principal objetivo do trabalho, os próximos passos são todos relacionados com a melhor integração com os componentes nativos e com a *IDE* de desenvolvimento *Visual Studio 2005*, como criação de novos componentes, integração com o *3D Studio Max* listados a seguir:

- Novos componentes – Para uma melhor análise, novos componentes 3D são necessários para a total abrangência das funcionalidades do *.NETCF*. Os componentes seriam desenhados e especificados de tal maneira que pudesse ser testados juntamente com todos os componentes nativos, para a total integração e garantia de compatibilidade.

- Integração com o *3D Studio Max* – Muito utilizado no mercado, o *3D Studio Max* é um programa de modelagem 3D que permite a criação de modelos mais detalhados, de modo que possam ser importados pelas aplicações no *PDA*. A integração com este programa seria muito interessante para permitir o desenho dos novos componentes 3D.

- Pesquisa sobre captura de tela em memória – Uma vez que as tecnologias apresentadas são diferentes (componentes nativos e 3D), o acesso em memória a uma imagem que unifica os dois tipos de componentes fica comprometido. Desta forma, realizar este procedimento de alguma forma seria útil como um trabalho futuro, já que isto poderia ser necessário em outras situações geradas ao criar novos componentes.

- Pesquisa sobre compatibilidade com *.NET Compact Framework 3.5* – Com o lançamento da nova versão da plataforma de desenvolvimento da *Microsoft*, é necessário a pesquisa sobre as novas funcionalidades e arquiteturas para garantir que todos os estudos efetuados neste trabalho estejam de acordo com a nova versão.

- Pesquisa sobre a compatibilidade com o *Visual Studio 2008* – Juntamente com a nova versão do *.NETCF*, a *IDE* de desenvolvimento também foi modificada. Deve-se, então, fazer um estudo sobre o funcionamento da nova versão para que todas as funcionalidades estudadas, tais como projeto customizado, *design time*, sejam válidas.

- Testes integrados – O escopo desse estudo era somente a possibilidade de integração com os componentes nativos e com o *VS 2005*. Para a continuidade deste trabalho, é necessário efetuar novos testes integrados com todos os componentes existentes para garantir a integração.

- Integração com *Microsoft XNA* – Como citado na seção 3.3.1, apesar de ainda não existir suporte para PDA's, este nova plataforma de modelagem 3D da Microsoft é mais completa e, portanto a sua integração pode ser interessante no futuro.

- Testes de carga – Os componentes desenvolvidos como protótipo foram testados visando principalmente a integração. Para a garantia de total integração e funcionamento aceitável em sistemas móveis utilizando a linguagem .NET CF, é necessário testes de carga nos componentes desenvolvidos. Isto seria muito importante para testar o desempenho dos componentes em sistemas reais, pois como descrito no item 2.1, os PDA's geralmente possuem recursos limitados e portanto o desempenho é um fator importante.

5. REFERÊNCIAS BIBLIOGRÁFICAS

Amr Elsehemy. **Custom Controls Design Time Support Part 9: Introducing the Designer**. 2008. Disponível em http://amrelsehemy.net/post/2008/01/20/Custom-Controls-Design-Time-Support-Part-9-Introducing-the-Designer.aspx. Acesso em 21 Nov 2008.

BBC Brasil. **Brasil é 81º em uso de celular e 72º em internet**. Disponível em http://www.bbc.co.uk/portuguese/reporterbbc/story/2008/02/080206_unctadbrasil_fp.shtml. Acesso em 22 Set 2008.

ComputerWorld Brasil. **Brasil terá 800 mil smartphones em uso até 2008**. Disponível em http://computerworld.uol.com.br/telecomunicacoes/2007/07/24/idgnoticia.2007-07-24.0317567791/. Acesso em 22 Set 2008.

Critical Development. **Compact Framework Controls (Part 1): Creating Custom Controls and Designers**. 2008. Disponível em http://dvanderboom.wordpress.com/2008/03/14/compact-framework-creating-custom-controls-and-designers-part-1/. Acesso 21 Nov 2008.

CrunchGear. **T-Mobile officially announces Android press conference on September 23rd. 2008**. Disponível em http://www.crunchgear.com/2008/09/16/t-mobile-officially-announces-android-press-conference-on-september-23rd/. Acesso em 22 Set 2008.

Direct3D Tutorial. **Getting Started with Direct3D**. Disponível em http://www.directxtutorial.com/Tutorial9/B-Direct3DBasics/dx9B1.aspx. Acesso em 21 Set 2008.

GRIFFITHS, I. et al. **.NET Windows Forms In A Nutshell**. 2003. ISBN: 978-0-596-00338-8.

Grundgeiger, D.. **Programming Visual Basic .NET**. 2001. ISBN: 978-0-596-00093-6.

HARBOUR, Jonathas S.. **Pocket PC Game Programing – Using the Windows CE Game API**. 2001. ISBN: 0-7615-3057-6.

LowEndMac. **Sculley's Dream: The Story Behind the Newton**. 2006. Disponível em http://lowendmac.com/orchard/06/john-sculley-newton-origin.html. Acesso em 15 Nov 2008.

Microsoft. **Press Release: Microsoft Releases Windows Mobile 5.0**. 2005. Disponível em http://www.microsoft.com/presspass/press/2005/may05/05-10WindowsMobile5PR.asp. Acesso em 15 Nov 2008.

Microsoft. **XNA Frequently Asked Questions**. 2007. Disponível em http://msdn.microsoft.com/en-us/xna/aa937793.aspx. Acesso em 15 Nov 2008.

MSDN. **Adding Designer Support to the .NET Compact Framework DateTimePicker Control**. 2006. Disponível em http://msdn.microsoft.com/en-us/library/aa446505.aspx. Acesso em 21 Nov 2008.

MSDN. **ControlDesigner Class**. 2006. Disponível em http://msdn.microsoft.com/pt-br/library/system.web.ui.design.controldesigner.aspx. Acesso em 21 Nov 2008.

MSDN. **ControlDesigner.OnClick Method**. 2006. Disponível em http://msdn.microsoft.com/pt-br/library/system.web.ui.design.controldesigner.onclick.aspx. Acesso em 21 Nov 2008.

MSDN. **Direct3D Mobile**. Disponível em http://msdn.microsoft.com/en-us/library/aa452478.aspx. Acesso em 21 Set 2008.

MSDN. **Direct3D Mobile Resources**. Disponível em http://msdn.microsoft.com/en-us/library/aa451329.aspx. Acesso em 05 Out 2008.

MSDN. **.NET Compact Framework Version 2.0**. Disponível em http://msdn.microsoft.com/en-us/library/ws1c3xeh(VS.80).aspx. Acesso em 11 Out 2008.

MSDN. **.NET Framework Programming**. Disponível em http://msdn.microsoft.com/en-us/library/ms229284(VS.80).aspx. Acesso em 11 Out 2008.

MSDN. **Transforms (Direct3D 9)**. Disponível em http://msdn.microsoft.com/en-us/library/bb206269(VS.85).aspx. Acesso em 05 Out 2008.

MSDN. **Documentação do Visual Studio**. Disponível em http://msdn.microsoft.com/pt-br/library/ms269115(vs.80).aspx. Acesso em 11 Out 2008.

.NET Developer's Journal. **Creating Custom Controls for Microsoft .NET Compact Framework in Visual Studio 2005.** 2005. Disponível em http://dotnet.sys-con.com/node/113332?page=2. Acesso em 21 Nov 2008.

Pekus. **Microsoft .NET Compact Framework**. Disponível em http://www.pekus.com.br/CompactFramework.htm. Acesso em 11 Out 2008.

Pluralsight Wiki. **Direct3D Tutorial**. Disponível em http://alt.pluralsight.com/wiki/default.aspx/Craig.DirectX/Direct3DTutorialIndex.html. Acesso em 05 Out 2008.

United Nations Conference on Trade and Development. **Information Economy Report 2007-2008**. 2007. ISBN: 978-92-1-112724-9.

Vivo. **iPhone 3G Vivo**. Disponível em http://www.vivo.com.br/iphone/home.php?tipo=pf. Acesso em 23 Set 2008.

Voidware. **Casio PF-3000**. 2004. Disponível em http://www.voidware.com/calcs/pf3000.htm. Acesso em 15 Nov 2008.

APÊNDICES

5.1. APÊNDICE A – ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE

ANÁLISE DE VIABILIDADE DA INTEGRAÇÃO DE COMPONENTES 3D (DIRECT3D) COM COMPONENTES NATIVOS COM .NET COMPACT FRAMEWORK

Apêndice A: Especificação de Requisitos de Software

Autores: *Raphael Petegrosso* *Renato Hiroaki Tano* *William Shinji Taniguchi*	Data de emissão: 21/11/2008
Revisor:	Data de revisão

Protótipo para o Estudo sobre componentes 3D
Especificação de Requisitos de Software

Objetivo do Documento

O objetivo do documento de Especificação de Requisitos do Software é identificar de forma completa e clara todos os requisitos a serem atendidos pelo software a ser desenvolvido.

Objetivo do Sistema

Nome do Sistema

Protótipo para o estudo sobre componentes 3D.

Escopo

O projeto apresentado tem como objetivo o desenvolvimento de um protótipo para a aplicação do conceitos analisados no projeto de formatura. O protótipo consiste de uma aplicação em que serão desenvolvidos componentes 3D para a integração com componentes nativos do *.NET Compact Framework*.

O princípio adotado no estudo, e conseqüentemente no protótipo, se refere à criação de ambientes mais interativos ao usuário, uma vez que os controles serão projetados para oferecer uma melhor visualização do elemento exibido, além de lhe proporcionar meios mais interessantes de interagir com o sistema.

Por fim, o estudo visa a integração com o ambiente de desenvolvimento Visual Studio 2005 da Microsoft, proporcionando aos elementos 3D desenvolvidos propriedades que possam ser parametrizadas pelo programador, que os redimensionará e customizará de acordo com as opções disponibilizadas e o resultado esperado para o projeto.

Definições, Siglas e Abreviaturas

PDA – Personal Digital Assistants.

Framework – Conjunto de ferramentas e práticas que têm como objetivo facilitar e acelerar o desenvolvimento de um determinado tipo de aplicação, provendo na forma de bibliotecas os códigos-fonte gerais aos projetos que utilizam o *framework*, permitindo ao desenvolvedor focar o seu tempo em problemas específicos a sua aplicação.

Visual Studio 2005 – Ambiente de desenvolvimento da Microsoft para a criação de projetos utilizando o Framework de desenvolvimento *.NET Framework* e *.NET Compact Framework*, tanto para sistemas *WEB*, *Desktops* e *Mobiles*.

Descrição Geral

Perspectivas do Protótipo

O protótipo será desenvolvido de modo a fornecer ferramentas para a criação de sistemas utilizando componentes 3D para a plataforma *.NET Compact Framework 2.0* da Microsoft. Dessa forma, o protótipo conta com o desenvolvimento de uma plataforma integrada com o *framework* da Microsoft, de modo a fornecer ao desenvolvedor ferramentas adicionais para a criação, sem que seja necessário utilizar outro ambiente.

Interface no Sistema

A interface do sistema do protótipo a ser desenvolvido será baseado no *.NET Compact Framework 2.0*, de modo que possa se adaptar às ferramentas já disponíveis por essa plataforma.

Interfaces de Usuário

Uma vez que os usuários do projeto serão desenvolvedores de sistemas móveis, a interface será feita de forma a proporcionar componentes cuja forma de implantação em um projeto seja similar às dos componentes já existentes. Dessa

forma, o usuário terá que simplesmente selecionar um componente que deseja utilizar, movê-lo para uma posição destino em um formulário e, por fim, preencher parâmetros relativos a esse item adicionado.

Interfaces de Hardware

Não se aplica.

Interfaces com Software

O protótipo a ser desenvolvido terá como base o uso das ferramentas proporcionadas pelo *Direct3D 10.0*. O *Direct3D* é um pacote da Microsoft que fornece ferramentas para facilitar a criação de objetos tridimensionais, que serão a base da criação dos componentes do projeto.

Para a utilização do protótipo será necessário um computador com sistema operacional Microsoft *Windows XP* e ter instalado o software Microsoft Visual Studio 2005, necessário para o desenvolvimento de aplicativos da linha Microsoft.

Além disso, será necessário ter instalado o *Microsoft Compact Framework 2.0*, de modo que se possa desenvolver aplicativos para dispositivos móveis utilizando o *Visual Studio*.

Por fim, para realizar a distribuição do sistema criado em um *PDA*, o usuário deverá ter instalado o software *Microsoft ActiveSync*, que realiza sincronizações entre aplicativos móveis e um computador.

Interfaces de Comunicação

Não se aplica.

Operação

Não se aplica.

Funções do Software

Criar Projeto Utilizando Componentes 3D – Função responsável pela criação de um novo sistema utilizando os Componentes *3D*.

Adicionar Placa 3D – Função responsável pela adição do componente Placa com Duas Faces.

Adicionar Tela Cúbica – Função responsável pela adição do componente Tela Cúbica.

Características dos Usuários

Os usuários do protótipo serão desenvolvedores de sistemas para *PDA's* com bom conhecimento técnico com relação ao *.NET Compact Framework*, necessário para o desenvolvimento de sistemas desse tipo.

Restrições

Por se tratar de componentes 3D que será utilizado em aplicativos que rodarão em um dispositivo móvel (*PDA*), o protótipo deverá ser desenvolvido de forma a se adaptar às limitações de desempenho do *PDA*, tais como pouca memória, processamento limitado, etc.

Usualmente dispositivos móveis tendem a ter um difícil modo de entrada de dados (teclado). Por esse motivo os componentes desenvolvidos devem ser simples e focar na usabilidade.

Hipóteses e Dependências

A hipótese adotada para o correto funcionamento do protótipo é a presença do .NET Compact Framework 2.0.

Quanto aos componentes, o sistema operacional do *PDA* deverá ser o Windows Mobile 5.0 para que os componentes funcionem corretamente.

Versões futuras

Não se aplica.

Requisitos Específicos

Modelo de Casos de Uso

O Modelo de Casos de Uso está descrito no Apêndice B.

Modelo de Classes

Não há.

Modelo de Interação

Não há.

Modelo de Estados

Não se aplica.

Requisitos de Bancos de Dados

Não se aplica.

Descrição das Interfaces Externas

Não se aplica.

Restrições de Projeto

A restrição essencial do protótipo é o uso do .NET Compact Framework 2.0, preferencialmente com o Visual Studio 2005.

Requisitos Não Funcionais

Usabilidade – O protótipo deve ser desenvolvido de modo que a facilitar o aprendizado. A configuração dos componentes no projeto deve ser feita tentando facilitar a interação do usuário com o projeto.

Confiabilidade – A confiabilidade do sistema deve-se principalmente na confiabilidade dos componentes disponibilizados, de modo que esses componentes tenham, pelo menos, a mesma confiabilidade garantida pelos componentes que foram utilizados como base. Além disso, deve-se garantir que as funcionalidade de outros componentes nativos do .NET Compact Framework 2.0 não sejam afetadas pelos componentes 3D.

Desempenho – O desempenho do protótipo também é critico nos componentes desenvolvidos. Por se tratar de componentes que serão utilizados em dispositivos móveis, deve-se garantir um desempenho em que o usuário não se sinta desconfortável com o sistema.

Disponibilidade – *Não se aplica*.

Segurança de acesso – *Não se aplica*.

Segurança – *Não se aplica*.

Critérios de Aceitação

Não se aplica.

Referências

Não há.

5.2. APÊNDICE B – MODELO DE CASOS DE USO

ANÁLISE DE VIABILIDADE DA INTEGRAÇÃO DE COMPONENTES 3D (DIRECT3D) COM COMPONENTES NATIVOS COM .NET COMPACT FRAMEWORK

Apêndice B: Modelo de Casos de Uso

Autores: *Raphael Petegrosso* *Renato Hiroaki Tano* *William Shinji Taniguchi*	Data de emissão: 21/11/2008
Revisor:	Data de revisão

Protótipo para o Estudo sobre componentes 3D

Modelo de Casos de Uso

Objetivo do Documento

O objetivo do documento é descrever os casos de uso do protótipo desenvolvido para aplicar conceitos vistos no estudo do projeto de formatura: “Análise de viabilidade da integração de componentes 3D (Direct 3D) com componentes nativos com .NET Compact Framework” .

Descrição de Atores

Desenvolvedor – Usuário que desenvolve um projeto para *PDA* utilizando controles tridimensionais. O usuário tem bom conhecimento técnico com relação ao *.NET Compact Framework*, necessário para o desenvolvimento de sistemas desse tipo. Dessa forma, os usuários terão capacidade de se adaptarem ao protótipo, uma vez que sua interface será baseada no *.NET Compact Framework*.

Diagrama de Casos de Uso

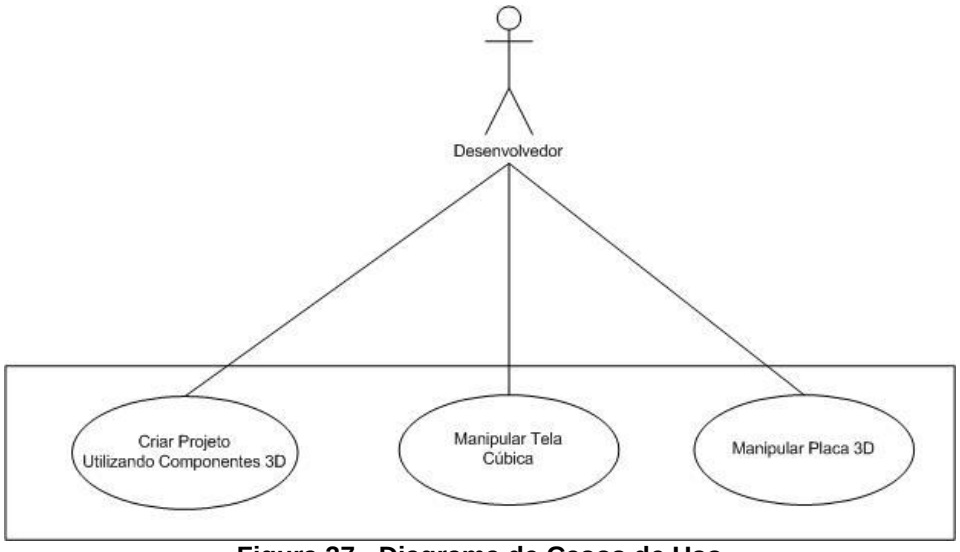

Figura 37 - Diagrama de Casos de Uso

Casos de uso

Caso de uso 1 – Manipular Placa 3D

Caso de uso 1.1 – Adicionar placa

Descrição: Esse caso de uso descreve a adição do Checkbox 3D.
Evento iniciador: Caixa de ferramentas aberta.
Atores: Desenvolvedor.
Pré-condição: Projeto criado utilizando protótipo.
Seqüência de eventos:

1. Desenvolvedor seleciona o controle Placa 3D.
2. Desenvolvedor posiciona o placa no formulário.
3. Sistema exibe o placa no formulário.
4. Desenvolvedor define as propriedades da placa.
5. Desenvolvedor escreve os eventos da placa.

Pós-condição: Placa adicionado no projeto e configurado.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 1.2 – Remover placa

Descrição: Esse caso de uso descreve a remoção da Placa 3D.
Evento iniciador: Seleção do formulário com o placa que se deseja remover.
Atores: Desenvolvedor.
Pré-condição: Desenvolvedor visualizando o formulário.
Seqüência de eventos:

1. Desenvolvedor seleciona a placa que deseja remover.
2. Desenvolvedor remove a placa do projeto.
3. Sistema deixa de exibir a placa do formulário.
4. Desenvolvedor remove os eventos do código.

Pós-condição: Placa removida no projeto.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 1.3 – Adicionar evento ao placa

Descrição: Esse caso de uso descreve a adição de um novo evento a Placa 3D.
Evento iniciador: Seleção do formulário com a placa para o qual se deseja adicionar o evento.
Atores: Desenvolvedor.
Pré-condição: Desenvolvedor visualizando o formulário.
Seqüência de eventos:

1. Desenvolvedor seleciona a placa no projeto.
2. Desenvolvedor solicita visualizar os eventos da placa.
3. Desenvolvedor seleciona o evento desejado e clica duas vezes sobre ele.
4. Desenvolvedor escreve o código referente a aquele evento.

Pós-condição: Evento adicionado a placa.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 1.3 – Remover evento do placa

Descrição: Esse caso de uso descreve a remoção de um evento de uma Placa 3D.
Evento iniciador: Seleção do formulário com a placa que se deseja remover.
Atores: Desenvolvedor.
Pré-condição: Desenvolvedor visualizando o formulário.
Seqüência de eventos:

1. Desenvolvedor seleciona a placa no projeto.
2. Desenvolvedor solicita visualizar os eventos da placa.
3. Desenvolvedor seleciona o evento que deseja remover.
4. Desenvolvedor remove o evento.
5. Desenvolvedor remove o código referente a aquele evento.

Pós-condição: Evento removido da placa.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 2 – Manipular Tela Cúbica

Caso de uso 2.1 – Adicionar tela cúbica

Descrição: Esse caso de uso descreve a adição de uma tela cúbica.
Evento iniciador: Caixa de ferramentas aberta.
Atores: Desenvolvedor.
Pré-condição: Projeto criado utilizando o protótipo.
Seqüência de eventos:

1. Desenvolvedor seleciona o controle tela cúbica.
2. Desenvolvedor posiciona a tela cúbica no formulário.
3. Sistema exibe a tela cúbica no formulário.
4. Desenvolvedor define as propriedades da tela cúbica.
5. Desenvolvedor escreve os eventos da tela cúbica.

Pós-condição: tela cúbica adicionada no projeto e configurada.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 2.2 – Remover tela cúbica

Descrição: Esse caso de uso descreve a remoção de uma tela cúbica.
Evento iniciador: Seleção do formulário com a tela cúbica que se deseja remover.
Atores: Desenvolvedor.
Pré-condição: Desenvolvedor visualizando o formulário.
Seqüência de eventos:

1. Desenvolvedor seleciona a tela cúbica que deseja remover.

2. Desenvolvedor remove a tela cúbica do projeto.
3. Sistema deixa de exibir a tela cúbica do formulário.
4. Desenvolvedor remove os eventos do código.

Pós-condição: Tela cúbica removido no projeto.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 2.3 – Adicionar evento à tela cúbica

Descrição: Esse caso de uso descreve a adição de um novo evento à tela cúbica.
Evento iniciador: Seleção do formulário com a tela cúbica para o qual se deseja adicionar o evento.
Atores: Desenvolvedor.
Pré-condição: Desenvolvedor visualizando o formulário.
Seqüência de eventos:

1. Desenvolvedor seleciona a tela cúbica no projeto.
2. Desenvolvedor solicita visualizar os eventos da tela cúbica.
3. Desenvolvedor seleciona o evento desejado e clica duas vezes sobre ele.
4. Desenvolvedor escreve o código referente a aquele evento.

Pós-condição: Evento adicionado à tela cúbica.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 2.4 – Remover evento da tela cúbica

Descrição: Esse caso de uso descreve a remoção de um evento de uma tela cúbica.
Evento iniciador: Seleção do formulário com a tela cúbica que se deseja remover.
Atores: Desenvolvedor.
Pré-condição: Desenvolvedor visualizando o formulário.
Seqüência de eventos:

1. Desenvolvedor seleciona a tela cúbica no projeto.
2. Desenvolvedor solicita visualizar os eventos da tela cúbica.
3. Desenvolvedor seleciona o evento que deseja remover.
4. Desenvolvedor remove o evento.
5. Desenvolvedor remove o código referente a aquele evento.

Pós-condição: Evento removido da tela cúbica.
Extensão: Não se aplica.
Inclusão: Não se aplica.

Caso de uso 3 – Criar projeto utilizando componentes 3D

Descrição: Esse caso de uso descreve a adição de um projeto juntamente com o componentes 3D.
Evento iniciador: Caixa de ferramentas aberta.
Atores: Desenvolvedor.
Pré-condição: Projeto criado utilizando o protótipo.
Seqüência de eventos:

1. Desenvolvedor escolhe a opção de novo projeto no menu.
2. Desenvolvedor escolhe a opção de projetos mobiles.
3. Desenvolvedor escolhe a opção de projeto utilizando o protótipo.
4. Desenvolvedor insere o nome do projeto.
5. Desenvolvedor indica o caminho em que será gravado o projeto.

Pós-condição: Projeto adicionado e componentes 3D carregados na caixa de ferramentas.
Extensão: Não se aplica.
Inclusão: Não se aplica.