UNIVERSIDADE FEDERAL DE PERNAMBUCO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

ODE4PPC
Um Porte do Engine Físico ODE para a Plataforma Pocket PC

Trabalho de Graduação

Aluno: Daliton da Silva
Orientadora: Prof. Judith Kelner
Co-Orientadora: Prof. Veronica Teichrieb

Recife, Janeiro de 2008

Resumo

Este Trabalho de Graduação tem como objetivo apresentar o ODE4PPC, um porte do *engine* gráfico ODE para a plataforma Pocket PC. Este porte foi realizado com o intuito de fornecer aos desenvolvedores de aplicações móveis para Pocket PC a possibilidade de integrar as suas aplicações 3D com uma biblioteca de simulação física gratuita, de código aberto e com ampla documentação e suporte.

O documento apresenta o levantamento realizado sobre os dispositivos móveis existentes no mercado, bem como as bibliotecas de física candidatas ao porte. É feita uma descrição dos passos adotados na realização do porte.

Posteriormente são mostrados os resultados obtidos com o porte, onde é feita uma análise do desempenho comparando a versão ponto fixo com a de ponto flutuante. Após busca exaustiva, não foi encontrada nenhuma outra biblioteca de física grátis e que desse suporte a desenvolvimento para Pocket PC. Dessa forma este trabalho de graduação se destaca pela sua característica inovadora. Na seção de conclusão são ressaltados os trabalhos futuros a serem implementados para complementar o desenvolvimento deste porte.

Agradecimentos

Sumário

1. Introdução 7
   1.1. Estrutura do documento 8
2. Contexto 9
   2.1. Dispositivos móveis 9
      2.1.1. PDA 9
      2.1.2. Celulares e *smartphones* 11
      2.1.3. Consoles portáteis de jogos 11
   2.2. Realidade virtual e realidade aumentada 12
   2.3. Representação de números reais 14
      2.3.1. Ponto flutuante 14
      2.3.2. Ponto fixo 22
   2.4. Trabalhos relacionados 25
3. *Engines* de física 28
   3.1. AGEIA PhysX 28
   3.2. Bullet 29
   3.3. Newton 30
   3.4. ODE 30
      3.4.1. Corpos rígidos 31
      3.4.2. Juntas 32
      3.4.3. Mundo 32
      3.4.4. Detecção de colisão 32
4. ODE4PPC: porte do ODE para Pocket PC 33
   4.1. Ambiente de desenvolvimento e dependências 33
   4.2. Implementação da matemática de ponto fixo 34
5. Resultados obtidos 36
   5.1. Sistema de renderização 36
   5.2. Testes preliminares 36
      5.2.1. Avaliação dos resultados 37
   5.3. Validação do ponto fixo 39
      5.3.1. Avaliação do desempenho 40
6. Conclusão 42
   6.1. Trabalhos futuros 42
7. Referências bibliográficas 43

Índice de Figuras

Figura 1. Ambiente de realidade aumentada. 7
Figura 2. PDA fabricado pela DELL. Computação na palma da mão. 10
Figura 3. Exemplos de *smartphones*: (a) PDA + telefonia; (b) celular + aplicações de PDA. 11
Figura 4. Triângulo de Realidade Virtual. 12
Figura 5. *Screenshot* de uma aplicação 3D desenvolvida com o porte do OGRE para Pocket PC. 13
Figura 6. Aplicação de RA em dispositivo móvel capaz de rastrear marcadores em tempo real. 14
Figura 7. Formato de ponto flutuante de 32 *bits*. 16
Figura 8. Formato de ponto flutuante de 64 *bits*. 17
Figura 9. Formato de ponto fixo (32 *bits*) adotado no desenvolvimento do ODE4PPC. 24
Figura 10. Jogo desenvolvido com o Mobiola 3D Engine. 25
Figura 11. Jogo criado com a PPL. 26
Figura 12. AGEIA 100M. PPU para *notebooks*. 27
Figura 13. Arquitetura de uma aplicação desenvolvida utilizando o PhysX. 28
Figura 14. Arquitetura da biblioteca Bullet. 29
Figura 15. Aplicações que utilizam o ODE: (a) Flight Simulator [24]; (b) Taxi3 [25]; (c) BaseGraph Pascal [26]; (d) Radish Works Cosmos Creator [27]. 31
Figura 16. Exemplos de juntas do ODE. 32
Figura 17. *Screenshots* da aplicação de teste do ODE4PPC. 37
Figura 18 Aplicação de demonstração utilizando o ODE4PPC + OpenGL ES: (a) cubos antes da colisão; (b) cubos após a colisão. 40

Índice de Gráficos

Gráfico 1. Taxas de renderização obtidas nos testes com retorno visual. 38
Gráfico 2. Tempos médios gastos nos passos da simulação. 39
Gráfico 3. Freqüência da simulação quando sem retorno gráfico..... 39
Gráfico 4. Comparação do desempenho do ponto fixo com o ponto flutuante. 41

1. Introdução

Um dos principais requisitos das aplicações que envolvem conceitos de Realidade Virtual (RV) e Realidade Aumentada (RA) é representar de forma fidedigna (comportamento similar ao que acontece na realidade) os elementos envolvidos nas simulações dos ambientes. Atualmente, existem diversos recursos à disposição dos desenvolvedores de aplicações 3D capazes de tornar a aplicação mais próxima do real. A Figura 1 ilustra um exemplo de ambiente de RA. Os elementos virtuais (o boneco) são vistos pelo usuário como se fizessem parte do ambiente real.

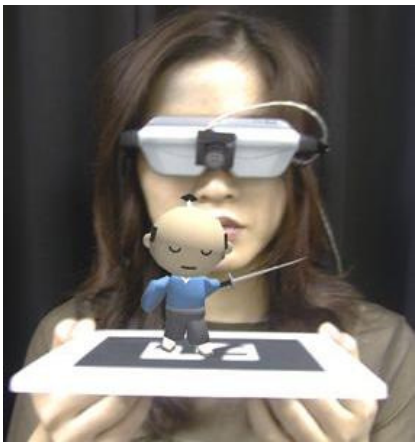

Figura 1. Ambiente de realidade aumentada.

Neste contexto, a renderização foto-realista vem sendo estudada e aplicada com o objetivo de refinar visualmente o conteúdo virtual tanto em aplicações de RV quanto de RA [1]. Existem diversas técnicas que podem ser empregadas para tornar as entidades do ambiente simulado mais realistas. Estas técnicas simulam, dentre outras coisas, o aparecimento de sombras, distorções das imagens dos objetos causadas por fatores ambientais ou até interferências acarretadas pelos cílios dos olhos do observador [2].

Recursos da área de Inteligência Artificial (IA) também vêm sendo amplamente utilizados. As pesquisas nas áreas de RV, RA e IA têm evoluído de forma independente. Apesar desse fato, nos últimos anos vários trabalhos que unem essas grandes áreas foram desenvolvidos [3]. Estes trabalhos defendem que o uso de IA no contexto de RV e/ou RA pode "automatizar" alguns pontos da aplicação de forma que se comportem tal qual entidades inteligentes do mundo real.

Outra tecnologia que pode ser integrada a uma aplicação de RV/RA são os *engines* de física. Estes funcionam como *Application Programming Interfaces* (APIs) capazes de simular o comportamento físico de entidades virtuais representativas de corpos, fluidos, partículas e outros elementos encontrados no mundo real.

Este Trabalho de Graduação descreve o desenvolvimento de um porte do *Open Dynamics Engine* (ODE) [4] para a plataforma Pocket PC, denominado ODE4PPC. Esse conhecido *engine* de física funciona como um simulador de corpos rígidos articulados. Tendo sido originalmente escrito em C++, também possui interface para programação em C. Foi feita uma revisão das tecnologias e conceitos envolvidos no desenvolvimento deste trabalho, e de qual a real contribuição oferecida à comunidade.

Uma das tendências mais recentes em aplicações 3D é o desenvolvimento para execução em dispositivos móveis. Desenvolver este tipo de aplicação para esses dispositivos representa um grande desafio, pois eles impõem sérias restrições de desempenho e espaço de armazenamento [5]. O ODE4PPC oferece à comunidade de desenvolvedores de aplicações para dispositivos móveis um *engine* de física gratuito e de código aberto para a plataforma Pocket PC. Até onde é do conhecimento dos autores este trabalho é inédito na literatura, até o atual momento.

1.1. Estrutura do documento

O capítulo 2 fala sobre os conceitos envolvidos no desenvolvimento deste trabalho, bem como mostra alguns trabalhos relacionados. O capítulo 3 faz uma breve descrição dos principais *engines* de física existentes na literatura, com um aprofundamento na biblioteca ODE, utilizada como base para este trabalho. O capítulo 4 descreve os passos realizados no desenvolvimento deste trabalho, com os resultados obtidos sendo descritos no capítulo 5. Por fim, a capítulo 6 ressalta as contribuições e propõe extensões futuras aplicáveis ao projeto.

2. Contexto

Este trabalho está inserido entre alguns conceitos como RV, dispositivos móveis e a representação dos números reais. Nas subseções seguintes são detalhados os conceitos envolvidos na concepção e desenvolvimento do ODE4PPC.

2.1. Dispositivos móveis

Com o avanço da tecnologia dos dispositivos de computação, têm-se construído nos últimos anos computadores cada vez menores e mais baratos [6], permitindo que aplicações para essa classe de dispositivos possam ser mais utilizadas na solução de problemas específicos, como organização pessoal, coleta de dados em ambientes externos, etc.

Graças à facilidade de integração com os mais diversos ambientes, os dispositivos embarcados encontram-se cada vez mais presentes em várias situações da vida cotidiana e profissional das pessoas. Celulares, sistemas de auxílio à navegação, salas de aula; todos esses cenários possuem pelo menos em quantidade mínima alguma tecnologia embarcada capaz de executar uma tarefa específica.

A presença massiva dos sistemas embarcados e sua contínua evolução representam um mercado em constante expansão. O desenvolvimento de *software* para esse tipo de dispositivo, por sua vez, tenta acompanhar toda a tecnologia de *hardware* que já está disponível, criando aplicações capazes de explorar o potencial da plataforma em que são executadas.

2.1.1. PDA

Os PDAs (*Personal Digital Assistants*) são computadores de tamanho reduzido, como mostrado na Figura 2, que têm como principal objetivo fornecer algumas ferramentas auxiliares na organização pessoal. Agenda de contatos, agendamento de compromissos, bloco de notas, calendário são exemplos de aplicações clássicas para estes dispositivos. O dispositivo mostrado na Figura 2 é um Pocket PC (classe de dispositivo utilizadas no ODE4PPC) que possui aceleração gráfica em hardware. Sua tela tem resolução de 640 x 480 *pixels*.

Com a evolução da capacidade de processamento desta classe de computadores, outros tipos de *softwares* puderam ser incorporados a gama de aplicações suportadas pelos PDAs. Atualmente, é possível utilizá-los para navegar na Internet através de redes sem fio, visualizar fotos, músicas, vídeos e até executar jogos e outros tipos de aplicações que exigem a renderização de gráficos em 3D. Existem duas famílias de PDAs hoje no mercado: Palm [7] e Pocket PC.

Palm é uma marca registrada da Palm, Inc. [7]. Trata-se de uma marca de PDAs que comercializa os dispositivos desde 1996, e desenvolveu o seu próprio SO (Sistema Operacional), o Palm OS. Apesar do grande número de Palms

comercializados, o Palm OS não foi estendido a outros fabricantes de PDAs. A API complexa e com pouco suporte do Palm OS fez com que a plataforma se tornasse impopular entre os desenvolvedores independentes, restringindo as aplicações do Palm às disponibilizadas pela fabricante do dispositivo.

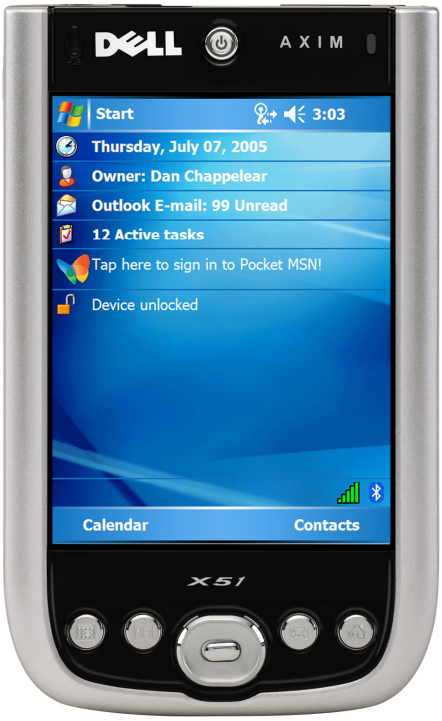

Figura 2. PDA fabricado pela DELL. Computação na palma da mão.

Pocket PC é uma especificação de PDAs não atrelada a um fabricante específico. Como o próprio nome diz (*Pocket* se traduz como bolso) a sua proposta é disponibilizar ao usuário um dispositivo que caiba no bolso e que realize as tarefas normalmente executadas em um computador de mesa comum. Para ser considerado um Pocket PC o dispositivo deve conter algumas características, listadas a seguir [8]:

- Possuir uma versão do SO da Microsoft [9] para dispositivos embarcados, no caso o Windows Mobile [10];
- Possuir um conjunto padrão de aplicativos pré-instalados na memória permanente (ROM);
- Incluir tela sensível ao toque;
- Incluir teclas direcionais ou um *touchpad*;
- Possuir algumas teclas de atalho padrão;
- Utilizar processador do tipo ARM, Intel XScale, MIPS ou SH3;

- As versões posteriores ao Pocket PC 2002 devem utilizar exclusivamente o processador ARM.

O SO do Pocket PC, o Windows Mobile, possui uma API de programação aberta e com amplo suporte. Além disto, possui diversas APIs auxiliares facilmente encontradas na Internet. Por esse motivo, o Windows Mobile vem se tornando o SO padrão para dispositivos móveis, ganhando adesão inclusive de alguns modelos de PDAs da Palm.

2.1.2. Celulares e *smartphones*

O celular, cada vez mais, está se tornando um objeto essencial e de uso cotidiano das pessoas. Progressivamente está agregando funções tradicionais de outros equipamentos, como envio de mensagens de texto e *e-mails*, acesso a Internet, fotografia, jogos eletrônicos, armazenamento e reprodução de conteúdo de mídia digital (vídeos, músicas, etc.). Dessa forma, o celular representa hoje o mercado mais promissor para o desenvolvimento de aplicações móveis.

Alguns fabricantes têm produzido PDAs que também trazem a funcionalidade de telefonia móvel e, por outro lado, fabricantes de celulares têm agregado aos seus aparelhos funcionalidades de PDAs. Estes aparelhos "híbridos" são chamados de *smartphones* (Figura 3).

Figura 3. Exemplos de *smartphones*: (a) PDA + telefonia; (b) celular + aplicações de PDA.

2.1.3. Consoles portáteis de jogos

Consoles são dispositivos móveis especificamente criados para rodar jogos. Os mais recentes possuem placa aceleradora 3D, como o Sony PSP [11] e o Nintendo DS [12]. Por serem desenhados especificamente para jogos, possuem qualidade de gráficos superior a qualquer PDA ou celular. Porém, trazem o grande inconveniente de não possuir API livre. Para ter direito de desenvolver para tais dispositivos precisa-se adquirir licenças de desenvolvimento muito caras. Dessa forma, o mercado de jogos para esses dispositivos portáteis é restrito a um pequeno número de empresas de grande porte, não dando qualquer oportunidade para desenvolvedores independentes.

2.2. Realidade virtual e realidade aumentada

RV é um tipo de interface avançada onde o usuário interage com elementos tridimensionais, utilizando-se de dispositivos de interação não convencionais. O principal objetivo dos sistemas de RV é trazer o usuário para um contexto mais próximo do que ele vivencia no seu dia a dia. Em vez de "janelas" e "botões" 2D, forma tradicional mas pouco natural das interfaces tradicionais de sistemas computacionais, o uso da RV permite a navegação por ambientes complexos onde a interação vai se dando através de objetos comuns posicionados no mundo virtual. A RV oferece dispositivos de interação que permitem ao usuário mais naturalidade na entrada dos dados. A RV pode ser ilustrada pela Figura 4, onde é mostrada a relação entre as três grandezas principais da RV: Imersão, Interação e Imaginação.

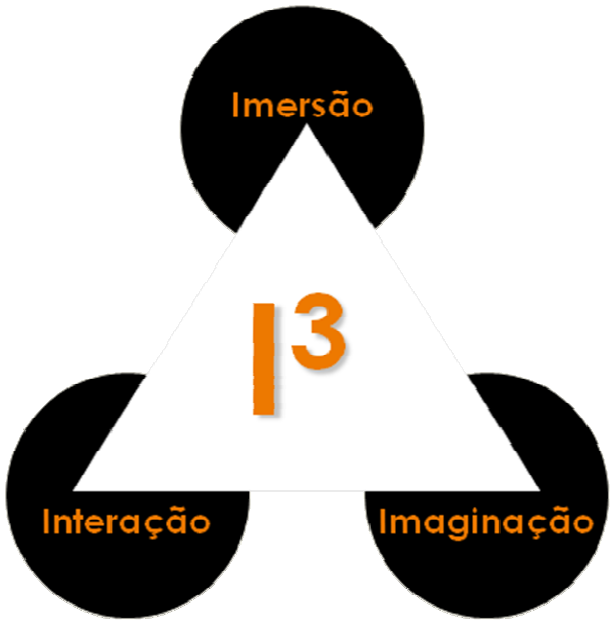

Figura 4. Triângulo de Realidade Virtual.

A RA trás a proposta de inserir elementos virtuais em tempo real nos ambientes reais. Para tanto, o ambiente real é capturado por câmeras e mostrado ao usuário,

já com a adição do elemento virtual. As entidades virtuais precisam estar registradas com o ambiente real, ou seja, o movimento da câmera e dos objetos reais devem influenciar na posição, orientação e escala dos elementos virtuais. Para fazer esse registro podem ser utilizados marcadores fiduciais [13] ou técnicas que identificam singularidades da cena, dispensando o uso de marcadores [14].

Várias pesquisas têm sido realizadas explorando o desenvolvimento de sistemas de RV e, principalmente de RA para sistemas embarcados. Dentre estas, destacam-se os sistemas de RA propondo soluções para tarefas que vão desde treinamento militar [15] até a simples leitura de *e-mails* [16].

Em [5] é apresentado um porte da biblioteca *Object-oriented Graphics Rendering Engine* (OGRE) para a plataforma Pocket PC. O OGRE é um *engine* gráfico de propósito geral que fornece uma API de alto nível, como forma de facilitar e agilizar o desenvolvimento de aplicações 3D. Este porte permite que aplicações de RA sejam desenvolvidas de forma transparente entre as plataformas PC e Pocket PC. A Figura 5 ilustra uma aplicação originalmente desenvolvida para PC e posteriormente portada para a plataforma Pocket PC, através do uso do porte do OGRE.

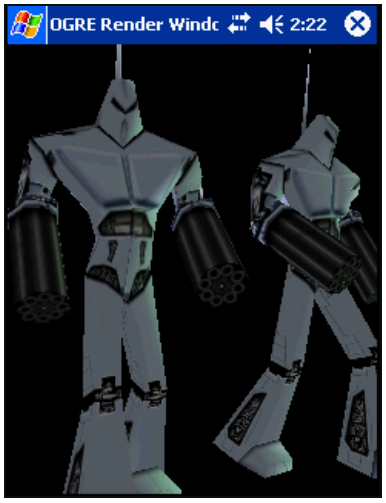

Figura 5. *Screenshot* de uma aplicação 3D desenvolvida com o porte do OGRE para Pocket PC.

Em [17] é descrito um sistema de RA embarcado em um PDA, utilizando uma câmera comercial para a detecção de marcadores. Como prova de conceito, os autores desenvolveram uma aplicação de navegação 3D (Figura 6) que guia o usuário a um determinado lugar dentro de um edifício desconhecido.

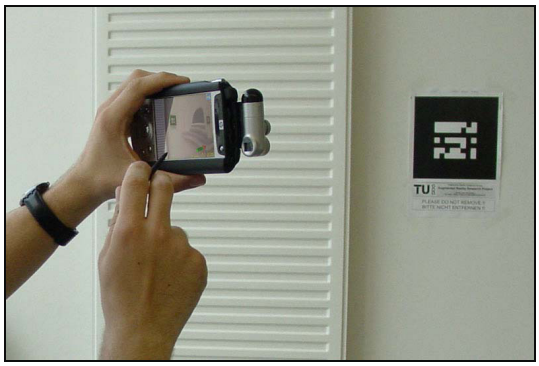

Figura 6. Aplicação de RA em dispositivo móvel capaz de rastrear marcadores em tempo real.

2.3. Representação de números reais

Os números reais podem ser armazenados através dos dois tipos de representação descritos nas subseções seguintes, a saber, ponto flutuante e ponto fixo. O estudo destas representações é importante porque o código do ODE precisará ser convertido, por questões de desempenho ligadas às limitações do *hardware*, de ponto flutuante para ponto fixo. Cabe ressaltar que o texto apresentado nas seções Ponto flutuante e Ponto fixo, abaixo, foi baseado no trabalho de graduação intitulado "Hardwire: um módulo em hardware para a visualização em wireframe de objetos tridimensionais" [18].

2.3.1. Ponto flutuante

Ponto flutuante é uma maneira de se representar números reais através de dígitos ou *bits* em um computador ou calculadora, da mesma maneira que a notação científica é utilizada para representar valores com exatidão. Um número na notação de ponto flutuante é geralmente armazenado como três partes:

- Um significante (indicando os dígitos que definem a magnitude do número);
- Um expoente ou escala (que indica a posição do ponto de separação entre parte inteira e parte fracionária);
- Um sinal (que indica quando o número é positivo ou negativo).

A computação de ponto flutuante possui um papel importante em uma variedade enorme de aplicações. A capacidade de realizar operações de ponto flutuante é uma importante medida de performance para computadores destinados a esses tipos de aplicações. O valor dessa capacidade é medido em "FLOPS" (*Floating point Operations Per Second*).

Números em ponto flutuante são destinados a representar o modelo matemático dos números reais. Todavia, enquanto os números reais formam uma continuidade que pode ser subdividida sem limites, números na notação de ponto flutuante apresentam uma resolução finita – eles apenas podem representar pontos

discretos de uma reta numérica real. Com a representação em precisão dupla (*double precision*), pontos consecutivos diferem de 1 a 10^{16}. Ou seja, eles só podem representar um subconjunto dos números reais. Por causa disso, um número na notação de ponto flutuante é às vezes considerado uma aproximação de um número real, ou a representação do número real com alguma tolerância, o que não é correto. Um número na notação de ponto flutuante (isto é, uma *string* de *bits* armazenada em um computador) representa exatamente um número real. O valor pode não ser o número real pretendido de acordo com a situação, caso ele não esteja no subconjunto representável pelos *bits* disponíveis.

Uma representação de ponto flutuante requer, antes de tudo, a escolha de uma base (*b*), assim como do número de *bits* (precisão *p*) que formarão o significante. O significante (característica + mantissa) é um número que consiste de *p* dígitos na base *b*, de forma que cada dígito varia entre 0 e *b* – 1. A base 2 (ou seja, representação binária) é utilizada pela grande totalidade dos computadores.

Uma das vantagens da notação científica é que números muito pequenos (ou muito grandes) podem ser representados sem a necessidade de *strings* gigantescas, com inúmeros zeros em seu começo ou fim, cuja contagem pode ser facilmente confundida. A notação científica determina uma posição específica para o ponto – logo depois do primeiro dígito diferente de zero, e o expoente é responsável por determinar o valor do número.

Alguns autores (e algumas representações de computador) optam por uma convenção diferente para a localização presumida do ponto, como na parte esquerda ou no *bit* mais à esquerda do número. Isto simplesmente adiciona um endereço constante ao expoente. Quando um número em ponto flutuante é normalizado, o seu dígito mais à esquerda deve ser não-nulo. Dessa forma, o valor zero não pode ser representado em uma notação normalizada de ponto flutuante, uma vez que não se teria nenhum dígito diferente de zero na parte mais à esquerda do número. Com base nisso, ou o número zero não seria representado (poderia possivelmente ser aproximado através da representação de um valor suficientemente pequeno), ou seria representado em violação às regras estabelecidas e tratado como um padrão que fosse reconhecido e tratado de forma especial. Por exemplo, a notação científica para o valor zero é simplesmente 0, enquanto que no IBM1620 (um computador decimal) ela aparece como *e* = -99 (expoente) e *s* = 0000000... (significante) – o significante não é normalizado, e o valor máximo do expoente negativo facilita a operação do processamento aritmético do *hardware*.

O valor matemático de um ponto flutuante é geralmente *s.sssssssssss...sss* x *be*. Quando se trabalha com a base binária, o significante é uma *string* de *bits* (1s e 0s) de tamanho *p*, dos quais o *bit* mais à esquerda possui valor 1. Quando o número em ponto flutuante não corresponde exatamente ao valor do número real

desejado, realiza-se uma aproximação de forma a encontrar o valor mais próximo do número real em questão.

Com o objetivo de representar um ponto flutuante como um dado armazenável em computador, o expoente deve ser codificado em um campo de *bits*. Uma vez que o expoente pode ser negativo, poderia ser usada a representação de complemento a dois. Ao invés disso, uma constante fixa é adicionada ao expoente, com a intenção de tornar o resultado um número positivo capaz de ser compactado em um campo de *bits* de tamanho fixo. Para a representação comum de 32 *bits* (*single precision*), ou formato *float* definido como um dos padrões da IEEE, essa constante possui o valor 127, de forma que o expoente é dito como representado no formato de "127 em excesso". O resultado dessa adição é armazenado num espaço de 8 *bits*. Como o significante mais à esquerda de um número em ponto flutuante (normalizado) é sempre 1, esse *bit* não precisa ser armazenado. O *hardware* do computador funciona como se o valor 1 que foi suprimido tivesse sido fornecido. Esse é o "*bit* implícito" ou "*bit* escondido" definido no padrão da IEEE, que permite que um campo de 23 *bits* represente um significante de 24 *bits*, mas também implica que o número zero não pode ser representado com todos os 23 *bits* iguais a zero, já que seria interpretado como um significante iniciado por 1. Uma codificação especial é requerida para o valor zero. Por último, um *bit* de sinal é necessário. Este é definido com o valor 1 para indicar que todo o número em formato de ponto flutuante é negativo, ou 0 para indicar que o mesmo é positivo.

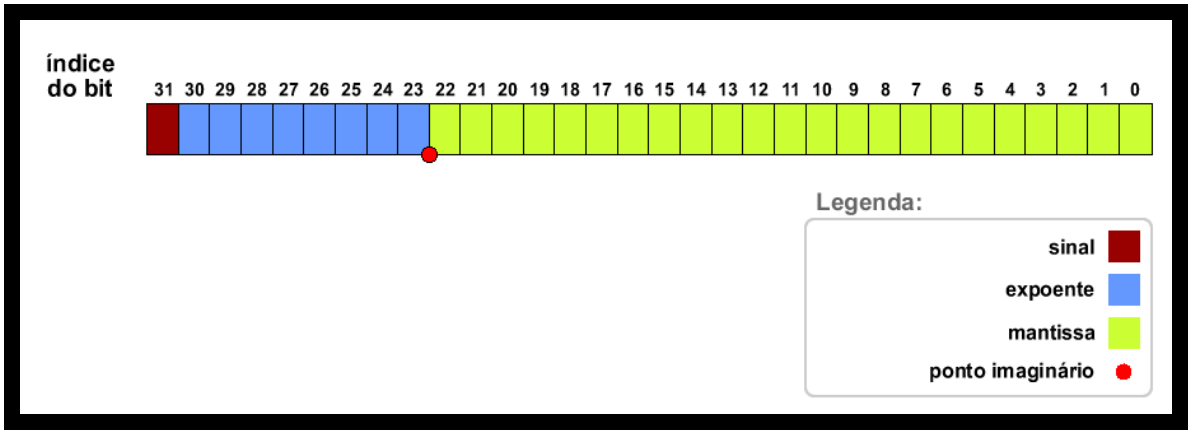

Figura 7. Formato de ponto flutuante de 32 *bits*.

No passado, alguns computadores utilizavam um tipo de codificação em complemento a dois para todo o número, ao invés do formato sinal/magnitude. O número em ponto flutuante é armazenado por completo em uma palavra de 32 *bits*, com o sinal localizado na parte mais à esquerda, seguido do expoente no formato de "127 em excesso" nos próximos 8 *bits*, e posteriormente o significante (sem o "*bit* implícito") nos 23 *bits* mais à direita. Para a representação comum de 64 *bits*

(*double precision*), o formato *double* foi definido como um dos padrões da IEEE. O deslocamento adicionado ao expoente possui valor 1023, e o resultado é posto em um campo de 11 *bits*. A precisão é composta por 53 *bits*. Após a remoção do "*bit* implícito", restam 52 *bits*. O resultado compreende *1 + 11 + 52 = 64 bits*. Ambos os formatos (32 e 64 *bits*) são mostrados nas Figura 7 e Figura 8.

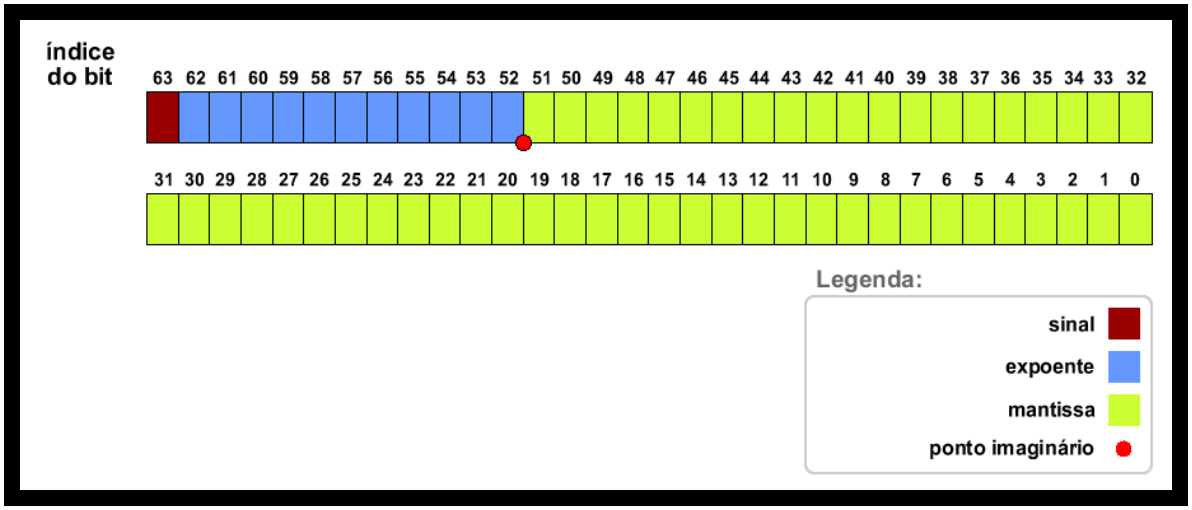

Figura 8. Formato de ponto flutuante de 64 *bits*.

A necessidade de compactar o expoente em um campo de *bits* de tamanho fixo impõe limites sobre o expoente. Para o padrão de 32 *bits*, e^{+127} deve ocupar um espaço de 8 *bits*, de forma que $-127 \leq e \leq 128$. Os valores -127 e 128 são reservados para representações especiais, de forma que o valor de *e* varia entre -126 e 127. Isso significa que o menor número positivo normalizado possui o valor de *e* = -126; *s* = 10000000000000000000000, com o valor aproximado de 1.18×10^{-38}, e é representado em hexadecimal por 00800000. O maior número representável possui o valor de *e* = 127; *s* = 11111111111111111111111, com o valor aproximado de 3.4×10^{38}, e é representado em hexadecimal por 7F7FFFFF. No caso da precisão dupla (64 *bits*), os valores variam de 2.2×10^{-308} até 1.8×10^{308}.

Quando há um processamento sobre um número no formato de ponto flutuante que origina um valor (após realizado o arredondamento necessário) acima do limite superior, diz-se que houve um caso de *overflow*. De acordo com o padrão da IEEE, o resultado é modificado para um valor especial que corresponde a "infinito", o qual possui um *bit* de sinal apropriado, o expoente reservado com valor de +128 e um padrão de *bits* na parte significante (tipicamente zero) indicando o "infinito". Esses números geralmente são impressos como "+INF" ou "-INF". *Hardwares* de manipulação numérica no formato de ponto flutuante são desenvolvidos para suportar operações com a representação de "infinito" de acordo com o esperado, como por exemplo, (+INF) + (+7) = (+INF) e (+INF) x (-2) = (-INF).

Quando há um processamento sobre um número no formato de ponto flutuante que origina um valor (após realizado o arredondamento necessário) não-nulo abaixo do limite inferior, diz-se que houve um caso de *underflow*. De acordo com o padrão definido pela IEEE, o expoente com o valor reservado de -127 é utilizado e o valor varia conforme descrito a seguir. Caso o valor do número seja zero, ele é representado por um expoente com valor -127 e todo o significante com valor 0. Isso significa que o valor 0 é representado em hexadecimal por 00000000. Caso o valor apresente expoente menor que -126, o significante é denormalizado, de forma a representar valores menores sem a notação de "*bit* implícito". Desta maneira, por exemplo, o valor aproximado de 7.3×10^{-40} seria representado por 0 00000000 00010000000000000000000, que corresponde a 00080000 em hexadecimal. A criação de números denormalizados é geralmente chamada de *underflow* gradual. À medida que os números ficam extremamente pequenos, os *bits* do significante são gradualmente sacrificados. Uma alternativa é o "*underflow* repentino", no qual qualquer número que não pode ser normalizado é simplesmente transformado em zero pelo *hardware*. Existe um alto grau de complexidade na manipulação de *underflow* gradual em *hardware*. Geralmente é usado *software* como suporte, através de interrupções. Isso acarreta uma perda de performance considerável, e nesses casos o "*underflow* repentino" deve ser implementado.

O comportamento padrão do *hardware* dos computadores é aproximar o resultado ideal (infinitamente preciso) de uma operação aritmética para o valor representável mais próximo, e fornecer essa representação como resultado. Na prática, existem outras opções. *Hardware* compatível com a especificação IEEE-754 permite a escolha de um dos seguintes tipos de arredondamento:

- Aproximar para o valor mais próximo (decisão padrão, utilizada na maior parte das vezes);
- Arredondar para cima (até o limite de máximo infinito; valores negativos vão para zero);
- Arredondar para baixo (até o limite de mínimo infinito);
- Arredondar para zero (funciona de forma similar às conversões de float para inteiro, que transformam, por exemplo, -3.9 para -3).

O padrão de arredondamento especificado pela IEEE 754 indica o uso da primeira opção (valor mais próximo) em todas as operações algébricas fundamentais, incluindo a operação de raiz quadrada. O uso não é obrigatório em funções de bibliotecas como seno e cosseno. Isto significa que o comportamento do *hardware* é completamente determinado para situações com 32 e 64 *bits*. O comportamento indicado para tratamento de *overflow* e *underflow* é definido de tal forma que o resultado apropriado seja computado, levando em consideração o tipo de arredondamento, como se o intervalo (*range*) do expoente fosse infinitamente

grande. Se o resultado do expoente não puder ser compactado no campo de *bits* corretamente, o *overflow*/*underflow* é tratado conforme descrito anteriormente.

A distância aritmética entre dois números representados em notação de ponto flutuante é chamada de "ULP" (*Unit in the Last Place*). Por exemplo, os números representados por 45670123 e 45670124 em hexadecimal estão distantes em 1 ULP. Essa medida corresponde a aproximadamente 10^{-7} em 32 *bits* (*single precision*), e 10^{-16} em 64 *bits* (*double precision*). A IEEE especifica que o resultado de qualquer operação esteja distante do resultado ideal em no máximo metade de uma ULP.

Em adição ao valor especial de "infinito" que é produzido na ocorrência de um *overflow*, também existe um outro valor especial chamado de "NaN" (*Not a Number*), que é produzido por uma operação de raiz quadrada de número negativo, por exemplo. O valor NaN é codificado com o expoente reservado de 128 (ou 1024), e um significante que o diferencie do valor especial de infinito. O objetivo das representações INF e NaN é que, em circunstâncias normais, esses valores podem ser propagados para as próximas operações (qualquer operação com um valor NaN retorna como resultado NaN), e o resultado só precisa ser tratado quando for conveniente.

Além da criação de valores especiais, existem "eventos" que podem acontecer:

- Um overflow ocorre, conforme descrito anteriormente, produzindo um infinito;
- Um underflow ocorre, conforme descrito anteriormente, produzindo uma denormalização;
- Uma divisão por zero ocorre quando o divisor da operação possui valor zero, produzindo um infinito com o sinal apropriado (apesar do sinal do zero não fazer diferença significativa). Destaca-se que um valor muito pequeno, porém diferente de zero, pode causar um overflow e resultar em infinito;
- Um erro de operando acontece quando um NaN é criado. Isso ocorre quando um dos operandos é um NaN, ou qualquer outro fator dá origem ao erro, como é o caso da radiciação ou log de números negativos;
- Um evento inexato acontece quando o arredondamento mudou o resultado do valor matemático verdadeiro. Todas essas exceções são postas em máscaras (não habilitadas). Algumas vezes overflow, divisão por zero e erro de operando estão habilitadas.

Uma vez que os números no formato de ponto flutuante não representam exatamente os números reais, e as operações entre eles não representam o comportamento exato das operações com os números reais, existem muitos problemas que surgem quando se escreve *software* que utiliza o formato de ponto flutuante. Enquanto a adição e a multiplicação são ambas operações comutativas (*a*

$+ b = b + a$ e $a \times b = b \times a$), elas não são associativas. De acordo com a ordem que as operações são realizadas, o arredondamento do ponto flutuante pode ocasionar diferenças nos resultados. As operações também não são distributivas, pelo mesmo problema de aproximação. O arredondamento é realizado ao fim de cada operação algébrica, o que leva a imprecisões que podem acarretar resultados inesperados.

Além da perda de precisão, a incapacidade de representar exatamente números como Π ou 0.1, e outras pequenas imprecisões, os seguintes fenômenos podem acontecer:

- Cancelamento: subtração de operandos com valores aproximadamente iguais podem causar uma falta de precisão grande. Esse é talvez o mais comum e sério problema de precisão;
- Conversões ineficazes para inteiros: a conversão de (63.0/9.0) para inteiro resulta em 7, enquanto a conversão de (0.63/0.09) resulta em 6, por exemplo. Isso acontece porque operações de conversão geralmente truncam o número, ao invés de arredondá-lo;
- Tamanho do expoente limitado: os resultados podem gerar overflow, tendendo para o infinito;
- O teste de divisão por zero é problemático: checar se o divisor é diferente de zero não garante que a divisão não irá gerar overflow ou infinito como resultado;
- Comparação de igualdade entre números: programadores geralmente realizam comparações com níveis de tolerância, mas isso não necessariamente resolve o problema.

Por causa dos problemas citados acima, o uso ingênuo de números no formato de ponto flutuante pode levar a diversos problemas. Um bom entendimento de análise numérica é essencial para a criação de *softwares* robustos de manipulação de ponto flutuante. Além do cuidado que se deve tomar na criação dos programas, também é necessário cuidado ao se projetar o compilador que será utilizado. Algumas otimizações realizadas pelos compiladores (por exemplo, reordenação de instruções) podem ir contra os objetivos do *software*, com comportamento esperado.

O maior potencial da aritmética de ponto flutuante é aproveitado quando ela é utilizada para medir quantidades do mundo real baseado em uma grande quantidade de escalas (como por exemplo, o período orbital de um planeta ou a massa de um próton), e seu pior funcionamento ocorre quando se pretende modelar interações de quantidades expressas como *strings* decimais que devem ser exatas. Como exemplo desse último caso, têm-se os cálculos financeiros. Por esse motivo, *softwares* financeiros tendem a não utilizar a representação numérica

binária de ponto flutuante. O tipo "*decimal*", de linguagens como C# e Java, assim como o padrão IEEE 854, foi criado como solução para os problemas que existem na notação de ponto flutuante binária e faz com que a aritmética sempre se comporte conforme esperado, com os números impressos em decimal.

O formato de 64 *bits* (*double precision*) é mais preciso do que qualquer medida física que pode ser realizada. Por exemplo, ele pode ser usado para indicar a distância da Terra até a Lua, com uma precisão de 50 nanômetros. O que torna a aritmética de ponto flutuante problemática é o fato de ser realizada uma enorme quantidade de operações, responsáveis por fazer os erros mínimos de precisão crescerem. Alguns exemplos são operações de inversão de matrizes, vetores característicos e ainda resolução de equações diferenciais. Esses algoritmos devem ser bem projetados para que funcionem corretamente.

Vantagens de algumas propriedades da notação de ponto flutuante:

- Qualquer número inteiro estritamente inferior a 224 pode ser exatamente representado na notação de 32 *bits* (*single precision*), assim como qualquer número estritamente inferior a 253 pode ser exatamente representado em 64 *bits* (*double precision*). Além do mais, o resultado da multiplicação de qualquer potência de 2 por um número com as características citadas acima pode também ser representado. Essa propriedade é muitas vezes utilizada em aplicações puramente inteiras, com o objetivo de conseguir inteiros de 53 *bits* em máquinas que possuem o formato de ponto flutuante de 64 *bits*, mas apenas inteiros com 32 *bits*;
- A representação dos *bits* é monotônica, contanto que os valores especiais sejam evitados e os sinais sejam manipulados cuidadosamente. Números no formato de ponto flutuante são considerados iguais se e somente se as suas representações inteiras são iguais. Comparações de maior e menor valor são feitas através de comparações inteiras no padrão de *bits*, desde que os sinais sejam coincidentes. Todavia, as comparações entre números no formato de ponto flutuante atuais apresentam tipicamente mais sofisticação no tratamento de valores especiais (como INF e NaN, por exemplo);
- Fazendo uma aproximação, tem-se que a representação em bits de um número da notação de ponto flutuante é proporcional ao seu logaritmo na base 2, com um erro médio de 3% (isso pelo fato do expoente estar na parte mais significante dos bits). Isso pode ser explorado em algumas aplicações, como em operações com volume em processamento de som digital.

A IEEE padronizou a representação binária de ponto flutuante para computadores com o IEEE 754. Esse padrão é seguido por praticamente todas as máquinas atuais. O padrão permite usar diferentes níveis de precisão, dos quais os

de 32 e 64 *bits* são os mais comuns, uma vez que são suportados por linguagens de programação convencionais. Alguns *hardwares* (por exemplo, da série Pentium e Motorola 68000) também fornecem um formato de precisão estendida com 80 *bits*, sendo 15 *bits* para expoente e 64 para significante, sem *bit* implícito.

2.3.2. Ponto fixo

Em Computação, o formato de ponto fixo representa um tipo de número real que possui um número fixo de casas decimais antes e depois do ponto decimal. Números na notação de ponto fixo são bastante úteis para representar valores fracionários nativamente em complemento a dois, caso o processador não possua uma unidade de processamento de ponto flutuante (FPU – *Floating-Point Unit*), ou se o formato de ponto fixo oferecer mais performance ou precisão. Grande parte dos processadores embarcados de baixo custo não possui uma FPU, que é o caso do Pocket PC.

Os *bits* à esquerda do ponto decimal representam a parte inteira (magnitude), enquanto os *bits* à direita do ponto representam a parte fracionária. Cada *bit* fracionário representa uma potência inversa de 2. Dessa forma, o primeiro *bit* possui o valor $\frac{1}{2}$, o segundo $\frac{1}{4}$, o terceiro $\frac{1}{8}$, e assim por diante. Para números no formato de ponto fixo na notação de complemento a dois, o limite superior é dado por: $2^{m-1} - \frac{1}{2^f}$, enquanto o limite inferior é dado por -2^{m-1}, onde *m* representa o número de *bits* da parte inteira (magnitude) e *f* representa o número de *bits* da parte fracionária. A assimetria entre limites superior e inferior ocorre devido à notação de complemento a dois. Por exemplo, um número binário de 16 *bits* com sinal no formato de ponto fixo, com 4 *bits* após o ponto decimal, apresenta 12 *bits* de magnitude e 4 *bits* fracionários. Ele pode representar números entre 2047.9375 e -2048. Um número binário de 16 *bits* sem sinal no formato de ponto fixo com 4 *bits* fracionários varia entre 4095.9375 e 0. Números em ponto fixo conseguem representar potências fracionais de 2 com exatidão, mas, assim como números em ponto flutuante, não conseguem representar exatamente potências fracionárias de 10. Se potências exatas de 10 forem desejadas, o formato BCD (*Binary-Coded Decimal*) deve ser utilizado. Todavia, BCD não torna eficiente o uso dos *bits* como faz a notação de complemento a dois, e nem é computacionalmente rápido.

Por exemplo, um décimo (0,1) e um centésimo (0,01) apenas podem ser representados aproximadamente na notação de ponto fixo em complemento a dois ou representação de ponto flutuante, enquanto a representação BCD consegue armazená-los com exatidão.

Valores em ponto fixo são sempre representados exatamente contanto que eles estejam na faixa determinada pelos *bits* de magnitude. Essa é uma das diferenças para o formato de representação de ponto flutuante, que possui campos de valor e

expoente separados, o que permite especificar valores inteiros maiores do que o número de *bits* no campo significante, causando ao valor representado perda de precisão.

Um uso comum para números no formato de ponto fixo BCD é o armazenamento de valores monetários, tendo em vista que nesse caso os valores inexatos dos números no formato de ponto flutuante são sempre um problema. Historicamente, representações de ponto fixo eram o padrão para tipos de dados decimais (por exemplo, em PL/I ou COBOL). A linguagem de programação ADA inclui suporte para ambas as representações em ponto fixo (ordinário e decimal) e ponto flutuante. A linguagem de programação JOVIAL também fornece tipos de dados em ambos os formatos.

Poucas linguagens de programação incluem suporte para valores em ponto fixo, porque na maior parte das aplicações as representações em ponto flutuante são rápidas e precisas o suficiente. A representação de ponto flutuante é considerada mais "fácil" para o programador do que a representação de ponto fixo, uma vez que ela pode lidar com uma variação dinâmica maior e não requer que os programadores especifiquem o número de dígitos existentes após o ponto decimal.

Todavia, caso haja necessidade, os números no formato de ponto fixo podem ser implementados até mesmo em linguagens como C e C++, que não incluem tal suporte. Com a publicação da ISO/IEC TR 18037:2004, tipos de dados no formato de ponto fixo foram especificados para a linguagem de programação C.

Existem várias notações usadas para representar o tamanho da palavra e o ponto decimal em um número no formato de ponto fixo. Uma notação comum é o prefixo "Q", onde o número que segue o Q especifica o número de *bits* fracionários à direita do ponto decimal. Por exemplo, Q15 representa um número com 15 *bits* fracionários. Essa notação é ambígua, uma vez que ela não especifica o tamanho da palavra, apesar de geralmente se assumir 16 ou 32 *bits*, dependendo do processador a ser utilizado. A forma não ambígua para a notação "Q" é representada por um Q seguido de um par de números, separados por um ponto, que indicam o número de *bits* de magnitude à esquerda do ponto decimal, seguido pelo número de *bits* fracionários. Por exemplo, Q1.15 descreve um número com 1 *bit* de magnitude e 15 *bits* fracionários em uma palavra de 16 *bits*. Outra notação, o prefixo "fx", funciona de forma similar e usa o comprimento da palavra como segundo item do par numérico. Por exemplo, fx1.16 descreve um número com 1 *bit* de magnitude e 15 *bits* fracionários, em uma palavra de 16 *bits*. O formato de ponto fixo adotado no desenvolvimento do ODE4PPC é mostrado na Figura 9.

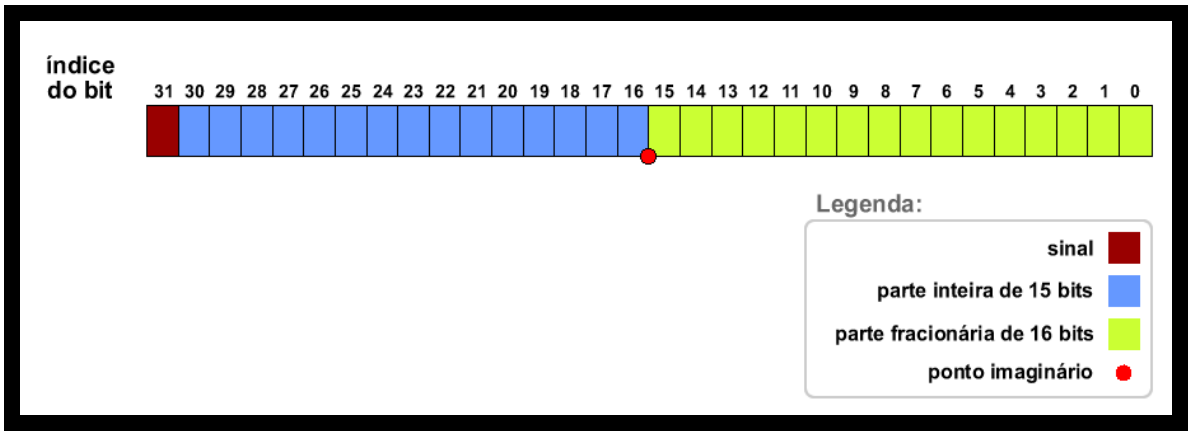

Figura 9. Formato de ponto fixo (32 *bits*) adotado no desenvolvimento do ODE4PPC.

Uma vez que operações com números no formato de ponto fixo podem produzir resultados que possuam mais *bits* do que os operandos envolvidos, existe a possibilidade de perda de informação. Como exemplo, o resultado de uma multiplicação de números no formato de ponto fixo pode potencialmente ter tantos *bits* quanto a soma do número de *bits* dos operandos. Com o objetivo de tornar o resultado representável com o mesmo número de *bits* dos operandos, o resultado da operação deve ser arredondado ou truncado. Se este for o caso, a escolha de quais *bits* devem ser mantidos é muito importante. Quando se multiplica dois números com o mesmo formato de ponto fixo, por exemplo com I *bits* inteiros e Q *bits* fracionários, a resposta pode possuir até 2*I *bits* inteiros, e 2*Q *bits* fracionários.

Por simplicidade, muitos programadores usam o mesmo formato dos operandos para o resultado. Como conseqüência, deve-se manter os *bits* intermediários; I *bits* para a parte inteira menos significativa, e Q *bits* para a parte fracionária mais significativa. Os *bits* fracionários que são perdidos representam uma perda de precisão que é comum em multiplicação de números fracionários. Se for perdido algum número diferente de zero na parte inteira, o valor estará radicalmente impreciso. Esse caso é considerado um *overflow*, e deve ser evitado em cálculos embarcados.

Recomenda-se utilizar um modelo baseado em simulação de operadores como o VisSim, para detectar e evitar tais *overflows* com o uso de uma configuração apropriada do tamanho da palavra do resultado e posição do ponto decimal, ganhos escalares adequados e limitação de magnitude de valores intermediários. Algumas operações, como a divisão, geralmente utilizam limitadores no resultado, fazendo com que qualquer *overflow* positivo resulte no maior número possível que pode ser representado no formato corrente. De forma similar, resultados com *overflow* negativo resultam na maior representação negativa suportada pelo formato

adotado. Esse limitante é geralmente chamado de saturação. Alguns processadores suportam um *flag* de *overflow* que pode gerar uma interrupção na ocorrência de um *overflow*, mas é geralmente tarde para recuperar os resultados apropriados a essa altura.

2.4. Trabalhos relacionados

Até a presente data, não foi encontrado nenhum *engine* de física com código aberto que oferecesse suporte para a plataforma Pocket PC. Apesar dos PDAs já estarem consolidados como uma opção de plataforma de execução de aplicações de RV e RA, não existem muitos trabalhos que visem o desenvolvimento de *engines* de física para essa classe de dispositivos. Alguns dos trabalhos mais relevantes são descritos abaixo.

O *engine* de jogos Mobiola 3D Engine [19] oferece suporte para renderização gráfica, processamento de som, lógica de jogo, física, entre outros. Todas essas funcionalidades são oferecidas mesmo em dispositivos com baixo poder de processamento, segundo os autores da biblioteca. Possui versões para Symbian OS, Windows Mobile (SO utilizado no Pocket PC), Linux e Brew. O *engine* de física interno ao Mobiola suporta simulação de corpos rígidos, diferentes métodos de detecção de colisão com precisão ajustável de acordo com a necessidade do usuário, simulação de atmosfera, simulação de aeronaves baseada em elementos aerodinâmicos, entre outros.

A desvantagem mais evidente do Mobiola é a inexistência de versões gratuitas, o que impossibilita sua comparação com o trabalho descrito neste artigo. A Figura 10 traz um *screenshot* de uma aplicação-exemplo integralmente desenvolvida sobre o Mobiola 3D Engine.

Figura 10. Jogo desenvolvido com o Mobiola 3D Engine.

A Pocket Programming Language (PPL) é uma linguagem de programação orientada a objetos, desenvolvida especificamente para dispositivos móveis [20]. De acordo com os autores, pode ser executada em *smartphones*, Pocket PCs e computadores *desktop*. A PPL provê um ambiente de desenvolvimento, além de garantir compatibilidade entre código das diferentes plataformas citadas anteriormente. Incorpora física 2D com suporte a detecção de colisão *pixel* a *pixel*. Apesar de possuir versões gratuitas, não pode ser utilizada como parâmetro de comparação com o ODE4PPC descrito neste Trabalho de Graduação, pois ainda não possui suporte a gráficos e física 3D. A Figura 11 exibe um jogo criado com a PPL. Os gráficos ilustrados na figura, assim como a física utilizada são em 2D.

Figura 11. Jogo criado com a PPL.

Dentre as diversas bibliotecas de simulação física existentes, uma tem se destacado entre os desenvolvedores: a AGEIA PhysX [21]. Essa biblioteca será melhor descrita na subseção 3.1. Além da biblioteca de física, a AGEIA desenvolveu um *hardware* co-processador de física, ou *Physics Processing Unit* (PPU).

Este novo tipo de *hardware* é responsável por executar a maior parte dos numerosos cálculos envolvidos na simulação física. É importante observar que o uso de um *hardware* específico para o processamento físico tem como principal conseqüência a possibilidade de simulações com maior riqueza de detalhes. Não necessariamente busca-se um incremento na velocidade da simulação.

Visando a demanda existente com os dispositivos móveis, a AGEIA lançou uma versão deste *hardware* específica para *notebooks*, a AGEIA PhysX 100M, que pode ser vista na Figura 12. Ela foi otimizada focando dois objetivos principais: redução

de tamanho e consumo de energia, o qual gira em torno de apenas 10W quando em pleno funcionamento.

Figura 12. AGEIA 100M*.* PPU para *notebooks*.

3. *Engines* de física

Existem, atualmente, muitos *engines* de física disponíveis para a comunidade de desenvolvimento de aplicações 3D para PC. Boa parte deles possui versões gratuitas e são de código aberto. Quando da seleção de uma destas bibliotecas para o desenvolvimento de uma nova aplicação, deve-se levar em conta uma série de elementos, que incluem a análise de quais características físicas são suportadas, o nível de integração com diferentes linguagens de programação e uma avaliação do desempenho com a utilização da biblioteca.

Neste capítulo é feito um levantamento das bibliotecas mais utilizadas atualmente, onde será justificada a escolha do ODE como a biblioteca a ser portada para a plataforma Pocket PC.

3.1. AGEIA PhysX

O AGEIA PhysX SDK (*Software Development Kit*) é um poderoso *middleware* de *engine* de física, baseado em *software*, para a criação de ambientes físicos dinâmicos [21]. O PhysX suporta as principais plataformas para jogos e aplicações gráficas, como mostra a Figura 13. A estruturação do PhysX é feita em camadas, a começar pela camada que abstrai o hardware onde o sistema está sendo executado. Em seguida vem a camada onde o PhysX é de fato implementado seguida pela camada que provê a API para o programador que usa o PhysX na sua aplicação. Por fim tem-se a camada opcional onde pode-se integrar o PhysX a outras bibliotecas, adicionando-se serviços como renderização processamento de dispositivos de entrada, etc.

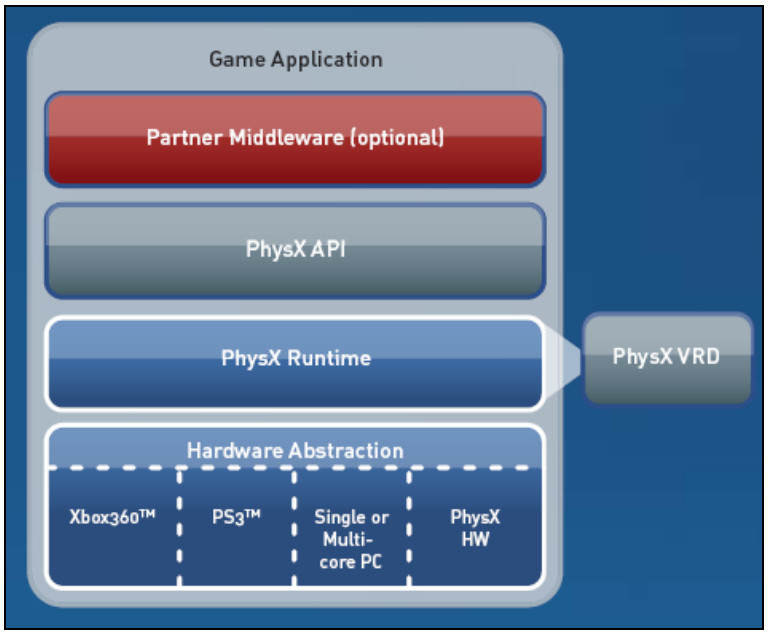

Figura 13. Arquitetura de uma aplicação desenvolvida utilizando o PhysX.

A API do PhysX possui uma avançada tecnologia de simulação para jogos e aplicações:

- Dinâmica de objetos com corpos rígidos complexos e detecção de colisão, para movimentação e interação realista de personagens;
- Juntas e molas, para modelagem de mecanismos complexos como a movimentação de personagens e a criação de colisões integradas a efeitos;
- Criação e simulação de fluidos volumétricos pode melhorar os efeitos visuais de cenas, e tornar possível o uso de armas de fluidos ou mangueiras de incêndio;
- O sistema de partículas inteligentes do PhysX FX torna possível simular de forma realista fogo, fumaça e neblina;
- Roupas aumentam o realismo de cenas contendo cortinas trêmulas e roupas que se moldam aos corpos.

Apesar de ser livre para utilização em qualquer aplicação, comercial ou não, o PhysX não possui código livre, fato que impossibilita a sua utilização neste trabalho.

3.2. Bullet

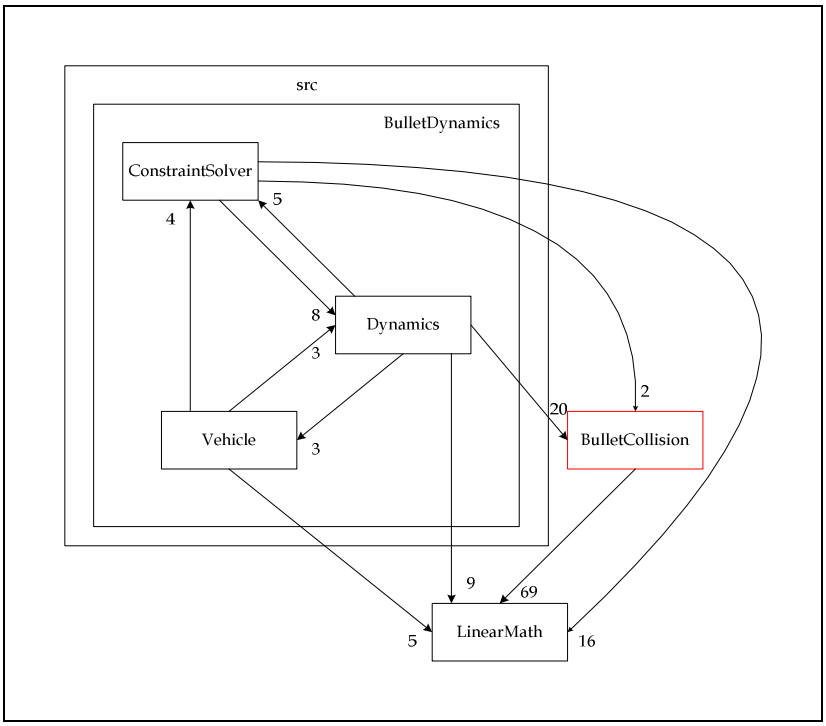

Figura 14. Arquitetura da biblioteca Bullet.

O Bullet [22] é uma biblioteca que realiza detecção de colisão e dinâmica de corpos rígidos para jogos. Possui código aberto e pode ser utilizada livremente em

aplicações comercias incluindo o Playstation 3. A biblioteca Bullet provê a detecção de colisão e a dinâmica de corpos rígidos, e é dividida em módulos com dependências claras. Esta divisão é refletida na estrutura de diretórios e subdiretórios para melhor representar os submódulos da biblioteca. Devido a esta divisão, o módulo de detecção de colisão pode ser usado sem o módulo responsável pela dinâmica (BulletDynamics). A Figura 14 mostra a arquitetura da biblioteca. Nela podemos ver a separação que existe entre a dinâmica dos corpos e a detecção de colisão. Também pode-se ver a interação entre todos os módulos e o núcleo de auxilio à matemática.

3.3. Newton

A biblioteca Newton [23] foi planejada para ter um impacto mínimo no desempenho da aplicação hospedeira, podendo controlar milhares de corpos sem delegar à aplicação o fardo de manter seus *arrays*. Quando a aplicação cria um objeto existe a opção de registrar um valor de dado do usuário ou alguns "*function pointers callbacks*". O Newton faz uso deste mecanismo para se comunicar com a aplicação sempre que algum tipo de informação é alterado. O Newton implementa um *solver* determinístico, que não é baseado em tradicionais LCPs (*Local Complex Potential*) ou métodos iterativos, porém possui a estabilidade e velocidade de ambos, respectivamente. Pode ser utilizada livremente, porém não possui o código fonte aberto.

3.4. ODE

O ODE foi desenvolvido para utilização em aplicações de tempo real que demandam o favorecimento da velocidade de processamento em prol da precisão da simulação física. Levando em consideração o poder de processamento relativamente baixo do Pocket PC quando comparado aos computadores de mesa, chegou-se a conclusão que o ODE seria a melhor escolha para o porte. Adicionalmente, o ODE já é amplamente utilizado pela comunidade (a Figura 15 mostra alguns exemplo do seu uso), podendo-se facilmente encontrar diversos fóruns especializados em solucionar dúvidas sobre o seu uso.

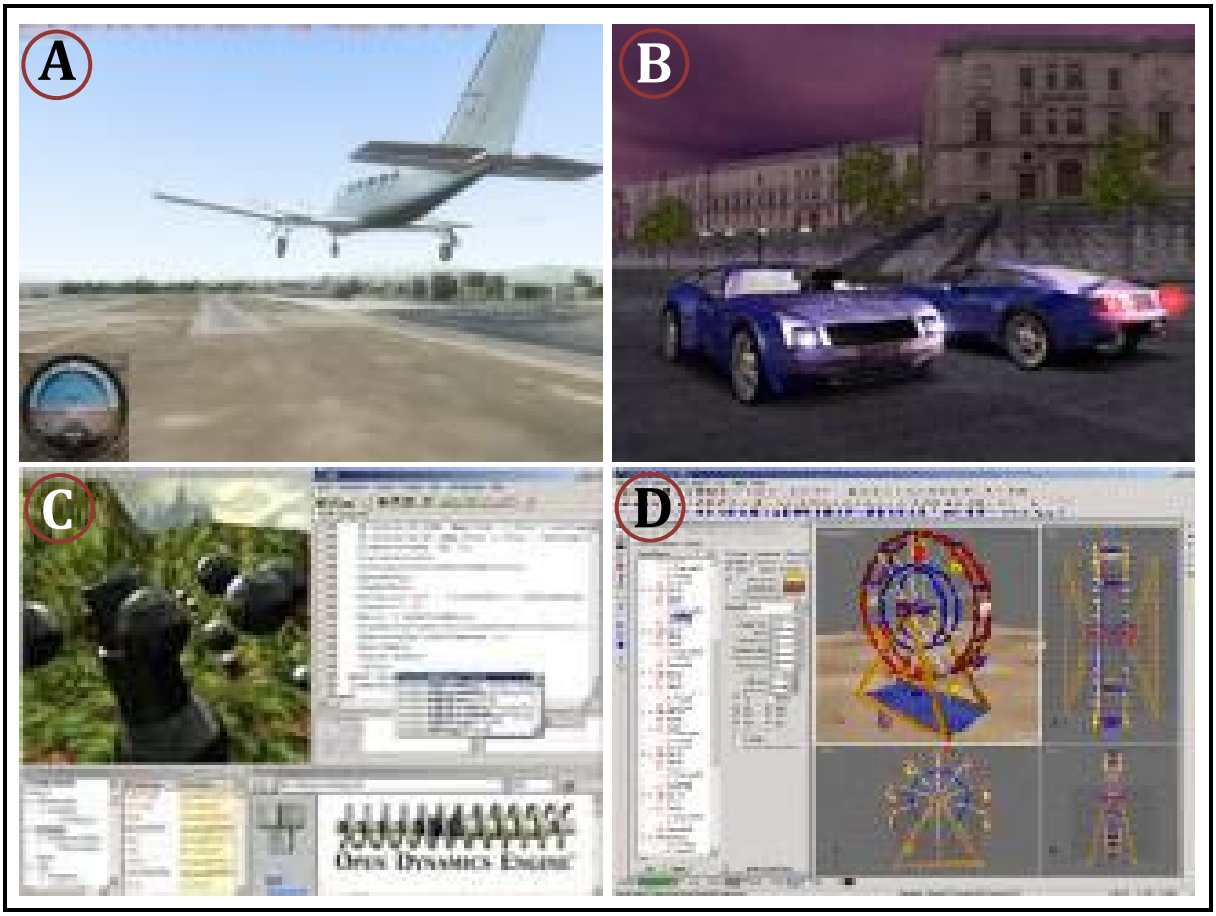

Figura 15. Aplicações que utilizam o ODE: (a) Flight Simulator [24]; (b) Taxi3 [25]; (c) BaseGraph Pascal [26]; (d) Radish Works Cosmos Creator [27].

As subseções seguintes trazem uma descrição resumida das principais características deste *engine*.

3.4.1. Corpos rígidos

O ODE suporta apenas a simulação da dinâmica de corpos rígidos, ou seja, não fornece suporte a objetos deformáveis, sistemas de partículas ou fluidos. No ODE, um corpo rígido reúne uma série de características que podem sofrer mudanças com o tempo (posição, orientação, velocidade, etc.) e outras que não são alteradas (massa e posição do centro de massa do objeto).

Existe no ODE o conceito de ilha, que é uma otimização de processamento responsável por agrupar objetos, representando assim a unidade básica de processamento. O uso de ilhas faz com que um objeto não seja processado individualmente, o que diminui a complexidade do processamento para $\Theta(n)$ (ordem de n), onde n é o número de ilhas. Adicionalmente, existe a possibilidade de desabilitar objetos, o que reduz ainda mais a carga do processamento. Essa característica é interessante para objetos que estão localizados em um ponto

distante da cena, quando seu processamento não influencia de forma significativa na cena em foco, em um dado instante. Este mesmo conceito pode ser aplicado às ilhas, onde também é possível desabilitar todo um grupo de objetos cujo processamento não é relevante em um determinado momento.

3.4.2. Juntas

No mundo real, objetos são constituídos por articulações que limitam o grau de movimentação de uns em relação aos outros. No ODE, essa característica pode ser simulada com o uso das juntas. Elas funcionam como relacionamentos que podem ser instanciados entre dois corpos, de modo a restringir as posições e rotações relativas entre eles. A Figura 16 ilustra três exemplos de juntas suportadas pelo ODE: *ball-and-socket* (i.e. ombro), *hinge* (i.e. dobradiça) e *slider* (i.e. pistão). Cada uma delas define um conjunto distinto de limitações entre os dois elementos envolvidos na ligação.

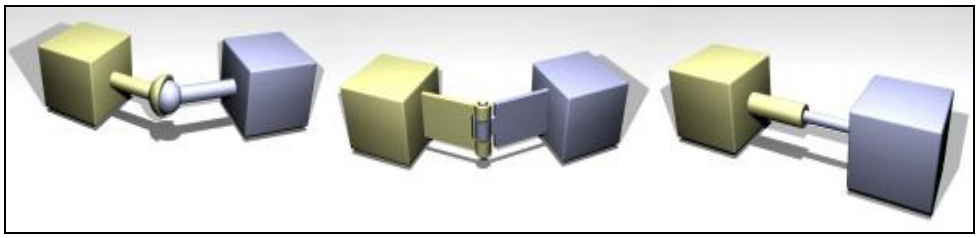

Figura 16. Exemplos de juntas do ODE.

3.4.3. Mundo

O mundo representa um *container* onde os corpos e as juntas são inseridos. Objetos de diferentes mundos não interagem entre si, o que significa que dois objetos de mundos diferentes não colidem um com o outro, por exemplo.

3.4.4. Detecção de colisão

A detecção de colisão é um módulo isolado do restante da biblioteca. É possível adicionar outro módulo de detecção de colisão que não seja o módulo padrão fornecido, bastando apenas que a interface predefinida pela biblioteca seja seguida.

Para detectar colisão entre os objetos, o desenvolvedor precisa atribuir formas a eles. Estas formas, chamadas de *shapes*, permitem que o ODE detecte se ocorreu colisão entre dois ou mais corpos.

4. ODE4PPC: porte do ODE para Pocket PC

O ODE4PPC, portado pelos autores, foi baseado na versão 0.8 do ODE, a qual encontra-se disponível no *site* oficial do *engine* de física [4]. Todas as funcionalidades da biblioteca foram integralmente portadas para a plataforma Pocket PC, possibilitando o aproveitamento de todos os seus recursos.

O código fonte do ODE é composto por três partes. A primeira delas é o núcleo principal, chamado ODE. Ele processa todos os objetos das cenas e realiza os cálculos de posição e rotação, além das chamadas aos outros núcleos do sistema. O ODE possui um arquivo *header*, chamado `config.h`, no qual é possível selecionar qual *trimesh engine* será utilizado na detecção de colisão, ou caso o desenvolvedor deseje, a não detecção de colisão.

As outras duas partes são os *trimesh engines*, responsáveis pela detecção de colisão. Desses dois *engines*, o que é utilizado por padrão pelo ODE é o *Optimized Collision Detection* (OPCODE), descrito em [28]. O OPCODE é uma biblioteca de detecção de colisão capaz de tratar malhas convexas ou não-convexas. Por ser utilizado em outras bibliotecas de física, o porte do OPCODE abre caminho para que este trabalho se estenda a outros *engines* de física.

A outra biblioteca de detecção de colisão fornecida é a GIMPACT [29]. Trata-se de uma biblioteca que contém ferramentas para o processamento de formas geométricas e a detecção de colisão. Da mesma forma que o OPCODE, também possui integração com outras bibliotecas de física [30][31]. A GIMPACT também teve o seu código portado para a plataforma Pocket PC.

Com o objetivo de preservar a portabilidade do código com o compilador para x86, todas as modificações realizadas na biblioteca foram incluídas através de diretivas de pré-processamento, permitindo que o compilador escolha qual versão do código deve ser compilada, dependendo de qual plataforma está sendo utilizada no momento.

Um dos primeiros problemas encontrados nas versões preliminares do ODE4PPC foi a impossibilidade da instanciação de muitos objetos em uma mesma cena, bem como o uso crescente de memória necessário ao uso da biblioteca. Após a investigação do problema, notou-se que ele estava sendo causado por uma "característica" do ODE: no arquivo `config.h` existe um `define` que seleciona como será feita a alocação de memória na biblioteca. Por padrão, está definido que as variáveis alocadas na pilha não serão desalocadas, com o objetivo de ganho de desempenho. Isto se tornou crítico no Pocket PC devido às restrições de memória impostas pelo dispositivo, as quais são bem mais rígidas que as do PC. A solução encontrada pelos autores foi modificar o `config.h` para que ele indique ao ODE quais as variáveis alocadas que precisam ser desalocadas posteriormente.

4.1. Ambiente de desenvolvimento e dependências

A ferramenta Microsoft Visual Studio 2005 foi a escolhida para compilar a versão do ODE para Pocket PC, uma vez que essa *Integrated Development Environment* (IDE) possui um utilitário de configuração que auxilia no processo de adaptação do ambiente de desenvolvimento, para geração de código nativo de Pocket PC. É possível escolher entre o SDK do Windows CE versão 2003 ou 2005.

Para suprir algumas necessidades não atendidas pela API padrão do Windows Mobile, foi incorporada ao porte do ODE a biblioteca libce [32]. Ela é uma biblioteca criada para preencher algumas lacunas presentes na biblioteca `libc` para Pocket PC, a qual não implementa muitas das funções suportadas na versão Win32 (*desktop*), como por exemplo, funções de temporização e manipulação de arquivos como `strftime()`, `gmtime()`, `localtime()`, `mktime()` e `time()`.

4.2. Implementação da matemática de ponto fixo

Para a representação de números reais, a API padrão do Windows Mobile oferece duas possibilidades: o tipo *float* e o tipo *double*. Ambos utilizam aritmética de ponto flutuante para a realização dos cálculos. O problema com estes dois tipos definidos na linguagem é que o *hardware* atualmente utilizado nos Pocket PCs não possui uma unidade de processamento de ponto flutuante (FPU). Dessa forma, todos os cálculos envolvendo números reais são emulados por *software*, o que compromete o desempenho global da biblioteca. A solução para esse problema é a conversão do código, passando a utilizar a aritmética de ponto fixo.

O formato de ponto fixo representa um tipo de número real que possui um número fixo de casas decimais antes e depois do ponto decimal [33]. O ponto fixo no ODE4PPC é representado através de uma classe chamada `FixedPoint`, que realiza a sobreposição dos operadores aritméticos padrões. Existem duas implementações disponíveis da classe `FixedPoint`; uma utiliza 32 *bits* para a representação do número, e outra utiliza 64 *bits*. Foi criada uma classe de suporte matemático para ponto fixo que implementa algumas funções (seno, cosseno, valor absoluto, função inversa, etc.) não suportadas diretamente pela classe `FixedPoint`.

Uma prática de programação comum para grandes projetos em C/C++ consiste na utilização de `defines`, para facilitar a generalização do código sem perda de desempenho. Baseado nisso, utilizou-se o tipo `real` como representante dos números reais. Este padrão é adotado em algumas classes do ODE, porém, devido à falta de homogeneidade do código, em muitas outras classes o tipo `float` é utilizado diretamente, sem a mediação destes `defines`. Este fato acarretou alguns problemas que impuseram dificuldades na implementação da aritmética de ponto fixo, pois todo o código da biblioteca teve que ser analisado para que todos os tipos reais pudessem ser substituídos pela generalização criada (ponto fixo ou flutuante).

Um exemplo onde este problema foi encontrado é o método `dMakeRandomVector` da classe `misc.cpp`. Em sua implementação, ele realiza divisão

de um número aleatório pelo maior valor real representável. O problema em questão é que este maior valor real representável foi digitado diretamente no código, correspondendo ao maior `float` representável. Dessa forma, após a conversão para a aritmética de ponto fixo, essa divisão passou a gerar valores incoerentes com o comportamento da simulação, tendo como resultado um decréscimo significativo do desempenho do método.

Ao se realizar comparações com zero quando se utiliza números reais, deve-se definir uma constante `epsilon`, a qual representa a tolerância máxima permitida na comparação, uma vez que a representação de números reais utilizada nos computadores nada mais é que uma discretização de números contínuos. Essa discretização acarreta em imprecisões maiores ou menores, dependendo do número de *bits* utilizados.

Essa prerrogativa é atendida no código fonte do ODE, exceto em um conjunto de comparações existentes em um método envolvido com a detecção de colisão. Esta falha originava erros de lógica durante a execução do método e, conseqüentemente, causava degradação do desempenho global da biblioteca. A falha só foi percebida quando o código foi convertido para ponto fixo, pois essa nova representação possui precisão inferior àquela em ponto flutuante. O problema foi resolvido com a introdução da constante de correção, e a utilização da representação de ponto fixo com maior precisão (64 *bits*).

Como no ODE4PPC o tipo `real` é representado por uma classe, o seguinte problema foi gerado. O ODE possui uma interface nativa para programação em C. Uma vez que a linguagem de programação C não suporta o uso de classes, houve uma incompatibilidade. A solução adotada pelos autores foi suprimir esta interface de C do ODE com o uso de diretivas de compilação. Futuramente, pode-se adaptar o código para utilizar uma representação de ponto fixo que não dependa do uso de classes C++. Nesse caso, a característica de interface nativa em C do ODE poderá ser facilmente recuperada pelo ODE4PPC, necessitando apenas a definição de uma constante.

5. Resultados obtidos

Para avaliar o correto funcionamento deste porte, foram implementadas duas aplicações de exemplo. Foi utilizado o Pocket PC HP iPAQ H5500 (dispositivo visto na Figura 17), que possui um processador Intel PXA255 XScale de 400MHz, 128MB de RAM, 48MB de ROM e um *display* LCD com 16 *bits* de *color depth* com resolução de 320x240 *pixels*. O seu SO é o Microsoft Windows Mobile 2003 (versão 4.20.1081, compilação 13100). Esse dispositivo não possui *hardware* de aceleração 3D.

5.1. Sistema de renderização

Devido à característica 3D da aplicação de teste, foi necessária a utilização de uma biblioteca gráfica para renderização. Dentre as disponíveis, o OpenGL *for Embedded Systems* (OpenGL ES) mostrou-se a melhor opção [34]. Essa biblioteca é uma versão simplificada de OpenGL, e foi concebida especificamente para dispositivos móveis como PDAs, telefones celulares e consoles de *videogames.*

Na concepção do OpenGL ES, muitas das funcionalidades do OpenGL foram removidas, e poucas foram acrescentadas. Pode-se citar duas destas mudanças como principais: a remoção das primitivas `glBegin-glEnd` e a introdução de tipos de dados de ponto fixo para as coordenadas de vértices [35]. O uso das primitivas `glBegin-glEnd` é uma das maneiras disponíveis em OpenGL para realizar a renderização de polígonos, e foi removido do OpenGL ES por questões de desempenho. Para renderizar, por exemplo, uma malha de triângulos no OpenGL ES, são necessários três passos: primeiro deve-se declarar um `array` com os vértices de todos os triângulos a serem renderizados, em seguida é preciso utilizar a instrução `glVertexPointer` para criar um ponteiro para os vértices definidos no `array`, e por último chamar a função `glDrawArrays(GL_TRIANGLES)` para renderizar simultaneamente todos os triângulos.

O suporte a ponto fixo foi adicionado no OpenGL ES para otimizar o processamento realizado nos processadores de dispositivos embarcados.

A implementação do OpenGL ES utilizada nos testes foi a Vincent, a qual não possui aceleração em *hardware* [36]. Não foi necessário realizar nenhuma modificação no núcleo do sistema de renderização para implementar a aplicação de demonstração.

5.2. Testes preliminares

Inicialmente foram feitos alguns testes para avaliar o correto funcionamento da biblioteca no dispositivo móvel. Estes testes foram realizados anteriormente a conversão do código para a utilização do ponto fixo.

A aplicação “test_chain1” utilizada para a validação inicial do porte foi adaptada a partir de uma das aplicações de teste disponíveis com o código fonte do ODE.

Chain (cadeia) é uma expressão em inglês que significa uma série de *links* conectados, tal qual uma corrente de bicicleta.

No exemplo, uma série de bolinhas estão conectadas através das juntas do ODE, fazendo com que a movimentação de uma bolinha cause impacto no deslocamento de todo o conjunto. O número de bolinhas da simulação foi parametrizado, podendo assumir qualquer valor a partir de dois. A cada passo da renderização uma força é aplicada no sentido positivo do eixo Z (para cima) em uma das bolinhas na extremidade da corrente. A força da gravidade atua no sentido contrário da força aplicada, fazendo com que as bolinhas fiquem em um ciclo de subidas e descidas sucessivas. Elas colidem entre si e também com o plano criado para representar o piso. Dois *screenshots* da execução da aplicação são mostrados na Figura 17.

Figura 17. *Screenshots* da aplicação de teste do ODE4PPC.

5.2.1. Avaliação dos resultados

Quando realizando cálculos de física em tempo real, uma característica importante que deve ser levada em consideração é a quantidade de passos de simulação por segundo que a aplicação é capaz de executar. Por isto, uma carga de testes foi executada para obter a taxa média de passos de simulação por segundo da aplicação utilizada nos testes do ODE4PPC.

Dois conjuntos de teste foram realizados para validação: no primeiro a aplicação foi executada de forma integral, incluindo a renderização da parte gráfica, tarefa executada pelo OpenGL ES; nos testes seguintes a parte gráfica foi

suprimida, de modo que a simulação física pudesse ser realizada sem o fornecimento do retorno visual.

Este procedimento foi necessário para avaliar qual o impacto causado pela renderização gráfica no desempenho final da aplicação. Esta avaliação é importante, visto que a versão do OpenGL ES adotada nos testes não possui aceleração gráfica e portanto mascara o desempenho real do porte implementado.

Testes com Retorno Visual

Os resultados dos testes realizados com retorno visual foram compilados no Gráfico 1. Nele pode-se observar que a taxa de atualização decresce de forma logarítmica à medida que se aumenta o número de objetos na simulação.

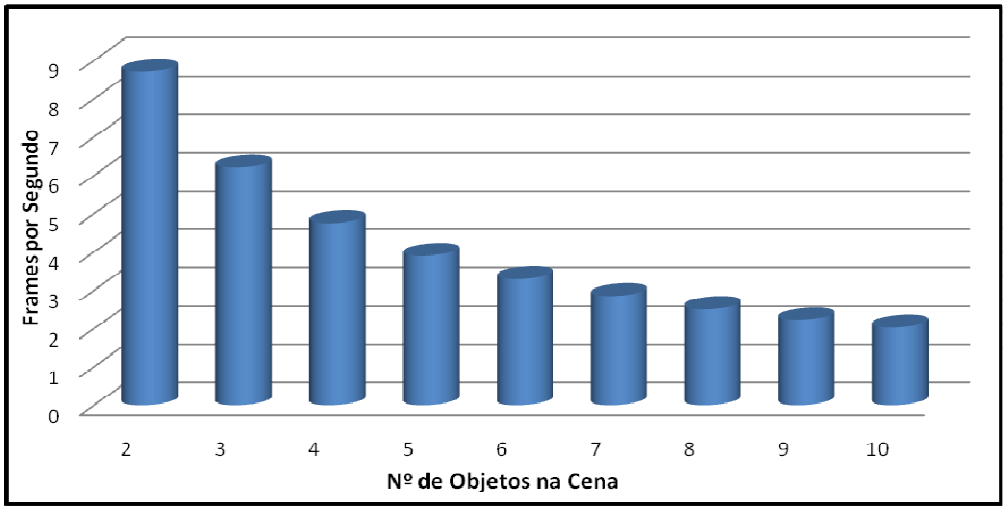

Gráfico 1. Taxas de renderização obtidas nos testes com retorno visual.

Testes sem Retorno Visual

Para realizar esse teste o código da aplicação foi modificado para suprimir as linhas de código responsáveis pela renderização das cenas. Um *loop* infinito ficou responsável por realizar a simulação física, chamando a cada iteração o método responsável por efetuar o passo de simulação.

Não se faz necessário calcular a quantidade de *frames* por segundo (fps), pois a noção de *frame* está ligada diretamente à renderização gráfica das cenas. Dessa forma, a grandeza medida durante a execução da aplicação foi o tempo decorrido entre o início e o fim de cada passo da simulação física, representado em microssegundos. Os resultados obtidos podem ser vistos no Gráfico 2.

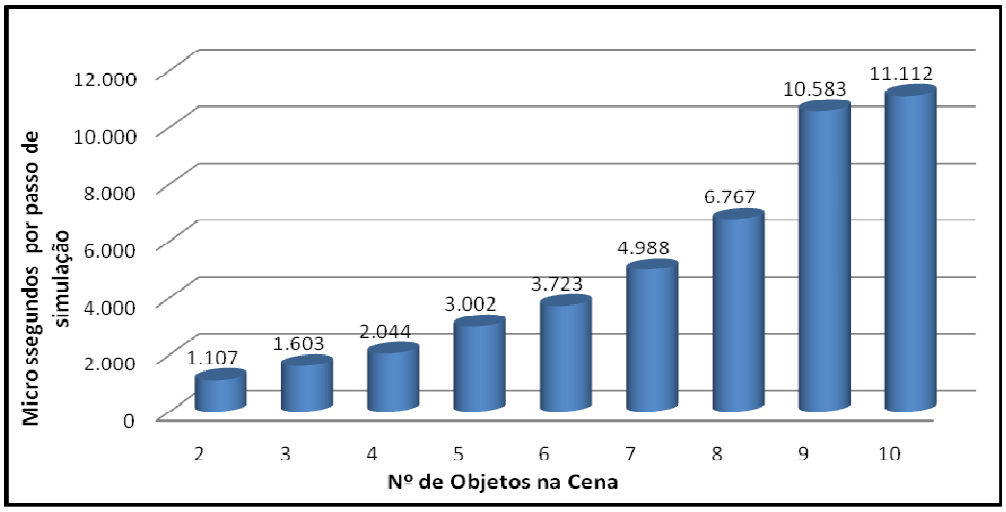

Gráfico 2. Tempos médios gastos nos passos da simulação.

Para efeito de comparação com os resultados obtidos nos testes realizados com o retorno visual, convertendo os valores do Gráfico 2 para fps, ter-se-ia taxas variando de 903 fps, no caso da simulação com dois objetos, até 89 fps, quando utiliza-se 10 objetos (Gráfico 3). Isto evidencia que a maior parte do tempo de execução é gasta com a renderização dos objetos e não com o cálculo da física. Este fato é uma conseqüência direta da utilização da versão do OpenGL ES sem aceleração por *hardware*.

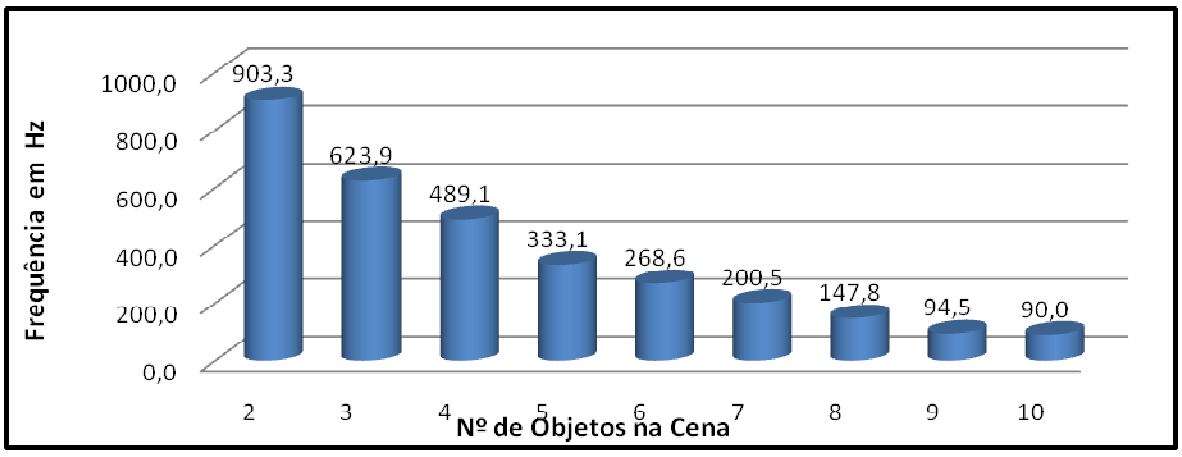

Gráfico 3. Freqüência da simulação quando sem retorno gráfico.

5.3. Validação do ponto fixo

Uma segunda aplicação de demonstração foi desenvolvida, composta por diversos cubos empilhados sobre o solo. Estes cubos são atingidos por uma esfera lançada a partir da origem da câmera, sendo a força desse lançamento determinada pela quantidade de toques dados pelo usuário na tela do dispositivo. A quantidade

de cubos empilhados foi parametrizada para possibilitar um estudo comparativo em relação ao desempenho da biblioteca. A Figura 18 mostra *screenshots* do dispositivo executando a aplicação.

Figura 18 Aplicação de demonstração utilizando o ODE4PPC + OpenGL ES: (a) cubos antes da colisão; (b) cubos após a colisão.

5.3.1. Avaliação do desempenho

Para avaliar o impacto da conversão do ODE4PPC para aritmética de ponto fixo foram realizadas várias execuções da aplicação de demonstração, utilizando as mesmas entradas, porém ora utilizando ponto fixo, ora utilizando ponto flutuante. Para quantificar o desempenho da aplicação foi medida a quantidade de passos de simulação por segundo que a aplicação foi capaz de executar. Esta taxa está expressa em fps.

Os resultados obtidos estão expressos no Gráfico 4. A análise do gráfico mostra que quando o número de objetos na cena é pequeno (base com um e dois cubos) os desempenhos das versões ponto fixo e ponto flutuante são semelhantes. Este resultado era esperado uma vez que a pequena quantidade de entidades físicas na simulação não é capaz de impactar significativamente no fps da aplicação, o qual fica limitado pela capacidade de renderização do dispositivo. No entanto, quando a quantidade de entidades cresce, a diferença entre o desempenho das duas versões aumenta de forma significativa. Com três cubos na base da pirâmide, o fps da versão em ponto fixo é 27,7% maior que a versão em ponto flutuante. Este número

cresce para 61,0% quando a base tem quatro cubos, 88,59% com cinco cubos na base e atinge 155,6% para seis cubos.

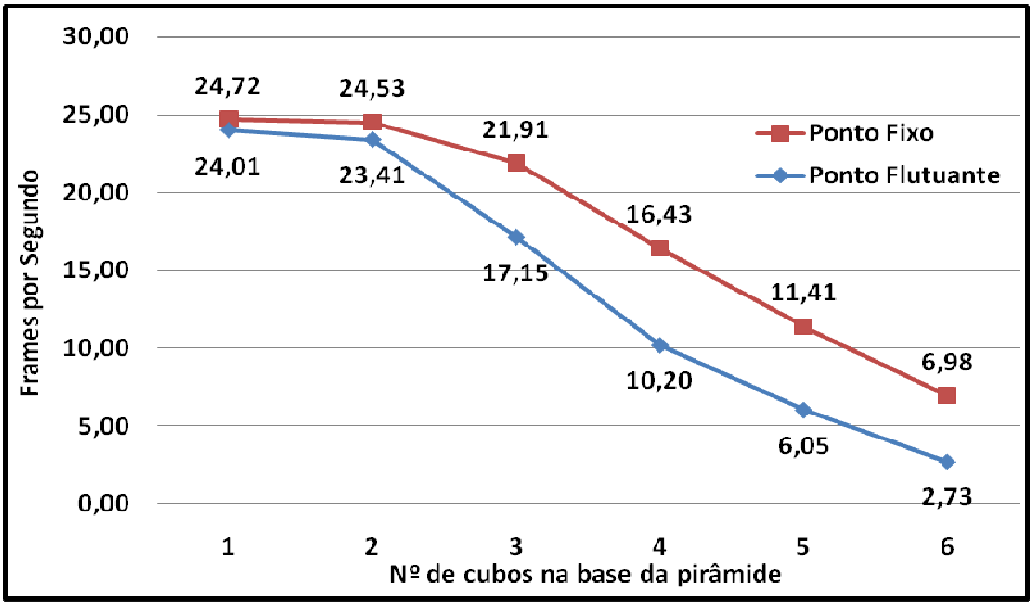

Gráfico 4. Comparação do desempenho do ponto fixo com o ponto flutuante.

6. Conclusão

O porte do ODE para a plataforma Pocket PC traz uma grande contribuição para a comunidade de desenvolvimento de jogos e aplicações 3D para dispositivos móveis. Ele atende à necessidade dos desenvolvedores por ser um *engine* de física gratuito e de código aberto, além de fornecer as funcionalidades necessárias para simulação de corpos rígidos. Este trabalho abre espaço para o porte de outras bibliotecas de física para Pocket PC, como também a extensão deste porte para outras plataformas.

Os testes preliminares realizados comprovam o ganho de desempenho significativo que o ODE4PPC obteve ao fazer uso da aritmética de ponto fixo, chegando a mais que o dobro de eficiência em comparação à versão em ponto flutuante, quando a cena envolveu um número suficientemente grande de entidades.

O porte foi publicado no IV Workshop de Realidade Virtual e Aumentada, **WRVA2007**. Também foi submetido para o X *Symposium on Virtual and Augmented Reality,* **SVR2008**, ainda sob avaliação.

Cabe ressaltar que o ODE4PPC será integrado a um *framework* de RA que suporta o desenvolvimento de aplicações tanto para a plataforma *desktop* quanto para Pocket PC [37]. Neste contexto, o ODE4PPC permitirá a utilização de simulação física nas aplicações desenvolvidas para Pocket PC.

6.1. Trabalhos futuros

Para complementar o desenvolvimento deste trabalho, serão disponibilizados o código fonte e o SDK do porte, devidamente documentados. Posteriormente, será realizada a integração deste porte com o *framework* de RA mencionado acima.

Uma possível implementação futura visando otimizar ainda mais a velocidade de execução é a utilização do compilador Intel C++ Compiler para Pocket PC, que é específico para o processador Intel XScale, provendo assim um aumento significativo no desempenho, de acordo com Wagner *et al.* [38].

7. Referências bibliográficas

[1] K. Agusanto, L. Li, Z. Chuangui e N.W.S. Nanyang, "Photorealistic Rendering for Augmented Reality Using Environment Illumination", *In proceedings of IEEE/ACM International Symposium on Augmented and Mixed Reality*, Tokyo, 2003, p. 208-216.

[2] G. S. Moura, J. M. X. N. Teixeira, V. Teichrieb e J. Kelner, "Efeitos de Iluminação Realistas em Realidade Aumentada Utilizando Imagens HDR", Workshop de Realidade Virtual e Aumentada, Itumbiara, 2007, p. 42-45.

[3] R. Aylett e M. Luck, "Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments", *In proceedings of Applied Artificial Intelligence*, Austria, 2000, p. 3-32.

[4] R. Smith, Open Dynamics Engine (ODE), disponível em: http://ode.org/, visitado em Setembro de 2007.

[5] J. P. S. M. Lima, T. S. M. C. Farias, V. Teichrieb e J. Kelner, "Port of the OGRE 3D Engine to the Pocket PC Platform", *In proceedings of Symposium on Virtual Reality*, Belém, 2006, pp. 65-76.

[6] T. Hollerer, S. Feiner, T. Terauchi, G. Rashid e D. Hallaway, "Exploring MARS: Developing Indoor and Outdoor User Interfaces to a Mobile Augmented Reality System", *In proceedings of Computer & Graphics*, 1999, p. 779-785.

[7] Palm, Inc., disponível em: http://www.palm.com, visitado em Janeiro de 2008.

[8] Wikipédia, Pocket PC, disponível em: http://pt.wikipedia.org/wiki/Pocket_PC, visitado em Janeiro de 2008.

[9] Microsoft Corporation, disponível em: http://www.microsoft.com, visitado em Janeiro de 2008.

[10] Microsoft Corporation, Windows Mobile Edition, disponível em: http://www.microsoft.com/windowsmobile, visitado em Dezembro de 2007.

[11] Sony Corporation, PSP, disponível em: http://www.us.playstation.com/PSP/About, visitado em Dezembro de 2007.

[12] Nintendo, Nintendo DS, disponível em: http://www.nintendo.com/ds, visitado em Dezembro de 2007.

[13] W. Piekarski e B. H Thomas, "Using ARToolKit for 3D Hand Position Tracking Mobile Outdoor Environments*", In proceedings of 1st Int'l Augmented Reality Toolkit Workshop,* Darmstadt, Germany, 2002.

[14] A. I. Comport, E. Marchand e F. Chaumette. "A Real-Time Tracker for Markerless Augmented Reality", *In proceedings of International Symposium on Mixed and Augmented* Reality, Tokyo, Japan, 2003.

[15] Land Warrior, disponível em: http://www.fas.org/man/dod-101/sys/land/land-warrior.htm, visitado em Dezembro 2007.

[16] J. M. X. N. TEIXEIRA, G. Moura, D. Silva, L. C. Henrique, N. Bastos, V. Teichrieb e J. Kelner, "A case study on the development of 3D user interfaces for mobile platforms", *In proceedings of Symposium on Virtual Reality,* Petrópolis, 2006, pp. 77-83.

[17] D. Wagner e D. Schmalstieg, "First Steps Towards Handheld Augmented Reality", *International Symposium on Wearable Computers*, White Plains, Institute of Electrical & Electronics Engineer, New York, 2003, pp. 127-137.

[18] J. M. X. N. Teixeira. Hardwire: um módulo em hardware para a visualização em wireframe de objetos tridimensionais. Conclusão 2006.1.

[19] WARELEX, Mobiola 3D engine, disponível em: http://www.warelex.com/games/engine.php, visitado em Setembro de 2007.

[20] ArianeSoft, Pocket Programming Language, disponível em: http://www.arianesoft.ca/page.php?1, visitado em Novembro de 2007.

[21] AGEIA, AGEIA PhysX, disponível em: http://www.ageia.com/physx/, visitado em Setembro de 2007.

[22] Bullet Physics Library, disponível em: http://www.bulletphysics.com, visitado em Janeiro de 2008.

[23] Newton, Newton Game Dynamics, disponível em: http://www.newtondynamics.com/, visitado em Dezembro de 2007.

[24] Microsoft Corporation, Fligth Simulator, disponível em: http://www.microsoft.com/games/flightsimulator/, visitado em Dezembro de 2007.

[25] Team 6 Game Studios, Taxi 3, disponível em: http://www.team6-games.com/, visitado em Dezembro de 2007.

[26] BaseGraph, BaseGraph Pascal, disponível em: http://www.basegraph.com/bg/bgp/bgpintro.html, visitado em Dezembro de 2007.

[27] Radish Works, Cosmos Creator, disponível em: http://www.radishworks.com/CosmosCreatorInfo.htm, visitado em Dezembro de 2007.

[28] OPCODE, Optimized Collision Detection, disponível em: http://www.codercorner.com/Opcode.htm, visitado em Novembro de 2007.

[29] GIMPACT, Geometric tools for VR, disponível em: http://gimpact.sourceforge.net/, visitado em Outubro de 2007.

[30] BULLET Physics Library, disponível em: http://www.continuousphysics.com/Bullet/, visitado em Outubro de 2007.

[31] Jiggle, Rigid Body Physics Engine, disponível em: http://www.rowlhouse.co.uk/jiglib/index.html, visitado em Outubro de 2007.

[32] Graz University of Technology, Handheld Augmented Reality, disponível em: http://studierstube.icg.tu-graz.ac.at/handheld_ar, visitado em Setembro de 2007.

[33] Fixed-Point arithmetic, disponível em: http://en.wikipedia.org/wiki/Fixed-point, visitado em Outubro de 2007.

[34] Khronos Group, OpenGL ES Overview, disponível em: http://www.khronos.org/opengles/index.html, visitado em Novembro de 2007.

[35] D. Nadalutti, L. Chittaro, e F. Buttussi, "Rendering of X3D Content on Mobile Devices with OpenGL ES", ACM, New York, *3D Technologies for the World Wide Web*, 2006, pp. 19-26.

[36] Vincent, the 3-D Rendering Library for Pocket PCs and Smartphones based on the Published OpenGL, disponível em http://ogl-es.sourceforge.net, visitado em Junho de 2007.

[37] J. P. S. M. Lima: "Um framework de realidade aumentada para o desenvolvimento de aplicações portáveis para a plataforma pocket PC", Trabalho de Graduação, Universidade Federal de Pernambuco - Centro de Informática, 2007.

[38] D. Wagner, e D. Schmalstieg, "ARToolKit on the PocketPC Platform", *Technical Report*, Vienna Univ. of Technol, Austria, 2003, pp. 1-2.

Judith Kelner

Veronica Teichrieb

Daliton da Silva