

Folhas ao Vento



Um simulador de física em tempo real

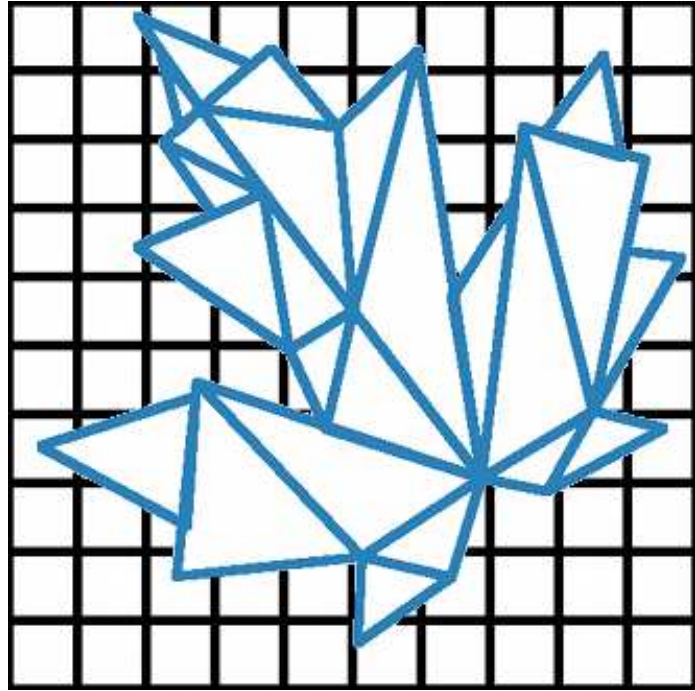
Aluno: J.P. Kerr Catunda
Orientador: Prof. José Coelho de Pina
DCC IME-USP



IME - Instituto de
Matemática e Estatística

[17]

1^o de dezembro de 2010



Sumário

1	Introdução	5
1.1	Motivação	5
1.2	Objetivos	5
1.2.1	Criar um cenário	5
1.2.2	Inserir objetos	5
1.2.3	Simular interação	5
1.2.4	Principais problemas	5
1.3	Por quê?	6
2	A física da interação entre corpos	6
2.1	A equação do movimento	6
2.2	A acrescentando novas forças	7
2.3	Cálculo e a evolução temporal	7
2.4	Arrasto	7
2.5	Gravitação	8
2.6	Vínculos	9
3	Estruturas de dados e algoritmos utilizados	9
3.1	Orientação a objeto	9
3.2	Tabelas dinâmicas	9
3.3	Interação de todos os objetos dois a dois	10
3.4	Máquina de estado, flags e máscaras	10
3.5	OpenGL	10
3.6	Sistemas lineares	10
3.7	Programação dinâmica de funções trigonométricas	11
3.8	Redimensionamento dinâmico de círculos	11
3.9	Modelos para desenho e interação	11
4	Linguagem e Ambiente	11
5	Atividades realizadas	13
5.1	Modelagem matemática	13
5.2	Vetores	13
5.3	O código	13
5.4	Modelos	14
5.5	Triangulação de polígonos	14
5.6	Leitura de cenas	14
5.7	Flags e teclas que alteram seus estados	15
6	Resultados	15
6.1	Animações	15
6.2	Arte em tempo real	15
6.3	Simulação industrial ou científica	16

6.4	Modularidade	16
7	Conclusão	17
8	Desafios passados e futuros	17
9	Disciplinas relevantes para o TCC	18
10	Considerações finais	18
11	Agradecimentos	19

1 Introdução

1.1 Motivação

De tempos em tempos me impressiono com a natureza. Paro o que estou fazendo e dedico atenção especial a beleza que me cerca. Quando tenho dúvidas encontro paz sentindo o vento no rosto e observando a natureza que me cerca.

Muito me agrada a idéia de simular tal beleza de forma realista e a transformar num cenário digital que possa ser utilizado tanto como arte quanto profissionalmente.

1.2 Objetivos

1.2.1 Criar um cenário

O primeiro objetivo do projeto foi criar um cenário com propriedades físicas. Ele é ser capaz de responder a perguntas como “Qual a temperatura na posição \vec{R} ?” ou “Qual a velocidade do meio na posição \vec{R} ?” de forma eficiente e concisa.

1.2.2 Inserir objetos

O segundo objetivo foi permitir a inserção de objetos no cenário. Estes objetos possuem características físicas e são capazes de responder perguntas como “Qual sua massa?” ou “Qual a sua velocidade?” de forma eficiente e concisa.

1.2.3 Simular interação

Inseridos objetos no cenário simulamos sua interação em tempo real atualizando propriedades tanto do cenário quanto dos objetos de forma realista. A isto chamamos de *cena*.

1.2.4 Principais problemas

As principais dificuldades deste trabalho foram a modelagem matemática do sistema físico e a utilização de estruturas de dados para o cenário e para os objetos de forma permitir a criação de diferentes cenas a tornar a interação física em cada uma delas possível, eficiente e realista.

1.3 Por quê?

Tais cenas tem como principal objetivo simular a interação do do vento e da gravidade do cenário com os objetos e com isso obter uma animação que possa ser usada como plano de fundo de outra aplicação.

Este projeto tem como objetivo além de criar um simulador físico fazê-lo de forma que possa ser usado para estimular o interesse dos alunos do BCC pela física que eles estudada no curso e, portanto, da ênfase ao conteúdo de Física I e II¹.

2 A física da interação entre corpos

Dois corpos podem interagir de diversas formas. Força gravitacional e resistência do ar são exemplos de nosso cotidiano que serão tratados neste documento.

2.1 A equação do movimento

Newton, o pai da mecânica, postulou em seu livro *Philosophiæ Naturalis Principia Mathematica*² suas três leis do movimento, hoje consideradas a base da mecânica clássica, a dizer:

1. Todo objeto mantém seu movimento até que uma força externa seja aplicada a ele
2. A relação entre a massa m de um objeto, sua aceleração \vec{a} e a força \vec{F} nele aplicada é dada por $\vec{F} = m\vec{a}$
3. Para toda ação existe uma reação igual e contrária

De cara nos deparamos com duas estruturas matemáticas importantes:

- Vetores: \vec{a} e \vec{F} são grandezas vetoriais
- Cálculo: \vec{a} é a variação da velocidade $\frac{d\vec{v}}{dt}$, que por sua vez é a variação da posição $\frac{d\vec{r}}{dt}$, ou seja: $\vec{a} = \frac{d^2\vec{r}}{dt^2}$

Para tratar vetores precisaremos criar uma estrutura vetorial capaz de fazer as operações necessárias. Já cálculo nos diz que devemos somar variações infinitesimais de uma grandeza[8, 9] relativas a um pequeno intervalo de tempo dt para evoluir nosso sistema.

A segunda lady Newton³ é muito importante pois nos dá uma forma de calcular o movimento em questão. Uma vez que temos tal equação do

¹FAP0126 e FAP0137 segundo a grade horária do BCC. Por ora apenas o conteúdo de Física I foi abordado

²Latin para "Princípios matemáticos de filosofia natural"[16]

³Não, ele nunca se casou

movimento podemos somar todas as forças que agem num objeto e obter a resultante. Com ela podemos calcular a aceleração do objeto.

2.2 Acrescentando novas forças

Cada força deve ser modelada matematicamente. Feito isso devemos somá-las e obter força resultante \vec{F} que usaremos para resolver a equação dada pela 2ª lei de Newton. Em particular, se nosso objeto está sujeito as forças de gravidade \vec{g} e arrasto \vec{D} , teremos como resultante

$$\vec{F} = \sum_{i=1}^n f_n = \vec{g} + \vec{D}$$

que nos dá a seguinte equação do movimento

$$\vec{F} = m\vec{a} \Rightarrow \vec{g} + \vec{D} = m\vec{a}$$

que, por sua vez, nos dá a aceleração \vec{a} do objeto.

2.3 Cálculo e a evolução temporal

A cada interação devemos calcular a aceleração de cada objeto e assim atualizar sua velocidade e posição. Entre duas iterações temos o intervalo Δt que usaremos para calcular o novo estado da cena. Como a aceleração é a variação da velocidade[3], trocamos o cálculo diferencial de nossa conta (ufa) por pequenas variações calculadas a cada paço da simulação. Vemos na Figura 1 a variação de uma função vetorial ao longo do tempo.

Podemos calcular a atualização da nova velocidade \vec{v} da seguinte forma

$$\vec{v} = \vec{v}_0 + \vec{a} \cdot \Delta t$$

e sua nova posição \vec{r} fica

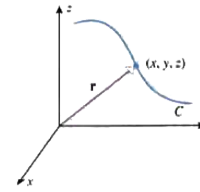
$$\vec{r} = \vec{r}_0 + \vec{v} \cdot \Delta t$$

Com isto em mãos basta agora modelar cada força em questão para obter a animação da cena.

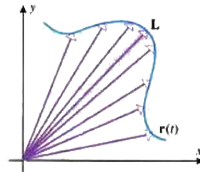
2.4 Arrasto

Uma vez que temos o objeto em movimento podemos levar em consideração a resistência do ar, também conhecida como arrasto.

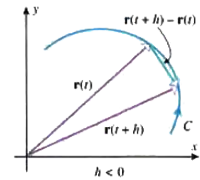
Arrasto é a força causada pelo ar ou qualquer meio viscoso no objeto quando existe uma velocidade relativa entre eles. Ela é uma força proporcional a velocidade[3, 20], e usaremos ela para modelar o vento. Ela pode ser proporcional a $|\vec{v}|$ ou $|\vec{v}|^2$, e aqui usaremos ela proporcional a $|\vec{v}|$. Podemos ver na Figura 2 a trajetória de um projétil sujeito a diferentes coeficientes de arrasto.



Ao variar t, a ponta do vetor \vec{r} percorre C.



se $\lim_{t \rightarrow a} \vec{r}(t) = \vec{L}$
se aproxima de \vec{L} em comp. e direção



$\vec{r}(t+h) - \vec{r}(t)$ nos dá a var. vetorial de $\vec{r}(t)$

Figura 1: Anton[9], pags 833, 838 e 839.

Mais precisamente o arrasto depende da densidade ρ do meio, da velocidade $|\vec{v}|$ do objeto, da seção transversal a do objeto e do coeficiente de arrasto d^4 da seguinte forma: $\vec{d} = -\frac{1}{2} \cdot \rho \cdot d \cdot a \cdot \vec{v}$.

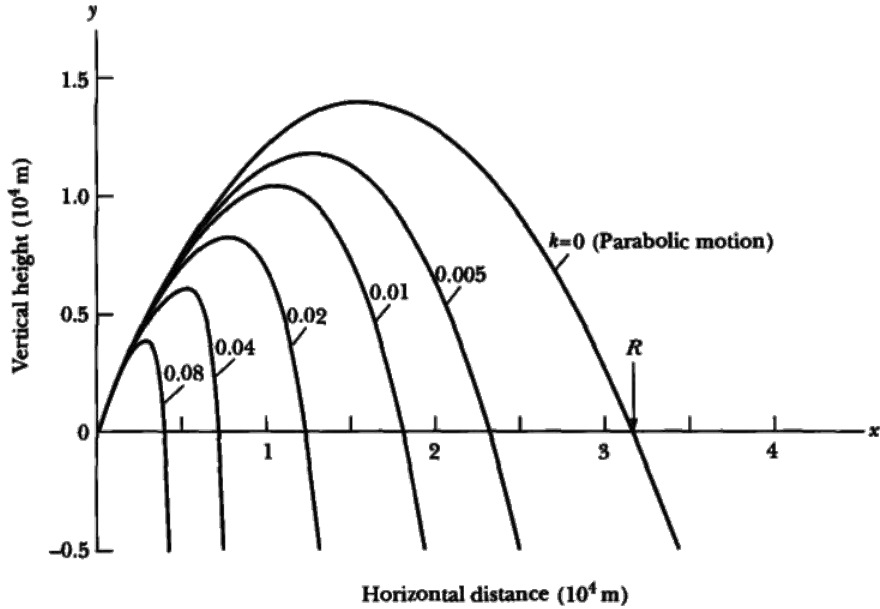


Figura 2: Cálculo de trajetórias com resistência do ar para diferentes valores de k em $F_{res} = -kmv$ com $\theta = 60^\circ$ e $v_0 = 600m/s$ retirado de Marion[20], pag 66.

Calcular o arrasto analiticamente é difícil e, quando tem solução, requer a conhecimento de equações diferenciais de primeira ordem. Entretanto, como dito antes, trocamos o cálculo pela evolução temporal do sistema.

2.5 Gravitação

De acordo com a gravitação[18, 3, 20], temos que massas se atraem da seguinte forma:

$$\vec{F} = G \cdot \frac{m_1 \cdot m_2}{|\vec{r}|^2} \cdot \hat{r} = G \cdot \frac{m_1 \cdot m_2}{|\vec{r}|^3} \cdot \vec{r}$$

O campo gravitacional, por se tratar de um campo conservativo, respeita o princípio da superposição[18, 20]. Com isto, o campo gravitacional em um ponto é dado pela soma da contribuição de todas as massas do sistema naquele ponto:

$$\vec{G}_j = \sum_{i \neq j}^n a_i = \sum_{i \neq j}^n k \cdot \frac{m_i \cdot m_j}{|\vec{r}_{ij}|^2} \cdot \hat{r}$$

⁴O coeficiente de arrasto, embora aproximado por uma constante, depende da geometria e da velocidade do objeto.

Precisaremos portanto interagir todos os objetos entre si dois a dois para conseguir tal simulação.

2.6 Vínculos

Objetos podem possuir vinculos entre si. Neste caso é preciso escrever as equações de movimento de cada um deles bem como suas equações de vínculo[18, 20]. Feito isto precisaremos resolver o sistema linear obtido para saber encontrar a força que atua em cada um dos corpos.

3 Estruturas de dados e algoritmos utilizados

Uma vez modelado nosso sistema, precisamos de estruturas de dados para acomodá-los.

3.1 Orientação a objeto

Para comportar tal sistema foi usado C++, uma linguagem rápida e com orientação a objeto. O diagrama UML das classes do projeto pode ser visto na Figura 3.

3.2 Tabelas dinâmicas

Para poder adicionar a uma cena um número arbitrário de objetos de forma eficiente e sem desperdício de memória foram utilizadas tabelas dinâmicas[4]. Tal algoritmo é $O(1)$ no sentido amortizado.

Cuidado especial foi tomado na remoção de elementos uma vez que elementos que não estavam no fim da tabela poderiam ser retirados. Para isso foi acrescida **linha 0** ao algoritmo de remoção[4] e ele ficou como abaixo:

```
Remove-da-Tabela (i)
0   troca A[i] com A[m]
1   se  $m \leq t/4$ 
2       então B  $\leftarrow$  Aloca-Tabela (t/2)
3       para k crescendo de 1 até m faça
4           B[k]  $\leftarrow$  A[k]
5       Libera-Tabela (A)
6       A  $\leftarrow$  B
7       t  $\leftarrow$  t/2
8   a  $\leftarrow$  A[m]
9   m  $\leftarrow$  m - 1
10  se m = 0
11      então Libera-Tabela (A)
12      t  $\leftarrow$  0
13  devolva a
```

Como desta forma elementos do vetor mudam de posição, não foram feitas referências diretas a nenhum deles.

3.3 Interação de todos os objetos dois a dois

Como todas as interações entre objetos são possíveis, temos que interagir todos com todos dois a dois. Supondo que possuímos m objetos no vetor A que precisam interagir entre si com a função `Interage (ObjA, ObjB)`, teremos o seguinte algoritmo:

```
Interage-Objetos (A, m)
1   para j crescendo de 1 até m faça
2       para k crescendo de j + 1 até m faça
3           Interage (A[j], A[k])
```

Tal algoritmo possui complexidade $\Theta(n^2)$.

3.4 Máquina de estado, flags e máscaras

O simulador possui flags que podem ser alteradas em tempo de execução para determinar o que desenhar ou quais interações levar em consideração durante a simulação.

Para tal ele foi programado como uma máquina de estado e utiliza máscaras e os operadores de bit “e” e “ou” (em C++: “&” e “[|”)[1] bem como estados pré determinados em um `enum` para saber qual o estado atual. O estado do cenário pode ser alterado pela função `ToggleFlag (flag)` que por sua vez também utiliza máscaras e operadores de bit.

3.5 OpenGL

Para desenhar a animação foi utilizada a biblioteca OpenGL[13] bem como seus recursos de texturas e matrizes. Ela, junto com o Glut[12] permitiu que o projeto fosse multiplataforma.

3.6 Sistemas lineares

Para simular objetos com vínculo é preciso resolver o sistema linear das equações de movimento. Para tal foi utilizada⁵ a fatoração LU[14]. Esta fatoração, embora instável, é eficiente e adequada para o problema em questão e possui complexidade $\theta(n^3)$.

⁵Esta parte do trabalho era além do escopo do projeto e não foi concluída, embora o autor tenha planos de terminá-la.

3.7 Programação dinâmica de funções trigonométricas

Funções trigonométricas são calculadas a partir de séries em ponto flutuante e custam caro. Para evitar cálculos trigonométricos desnecessários foi criado um buffer de valores de $\cos(\frac{\pi}{180} \cdot n)$ onde $n \in \mathbb{N}$ tal que $0 \leq n < 360$. Desta forma conseguimos valores de $\sin(\theta)$ e $\cos(\theta)$ em tempo constante ($\Theta(1)$) com precisão suficiente simplesmente interpolando seus valores.

3.8 Redimensionamento dinâmico de círculos

Para desenhar círculos suaves de tamanho arbitrário com o mínimo de segmentos de reta necessários foi criada uma função que escolhe o mínimo de pontos necessários de acordo com o tamanho do círculo.

3.9 Modelos para desenho e interação

Uma grande dificuldade no projeto foi construir modelos para a interação mecânica. Isto aconteceu pois o mesmo modelo que foi usado para desenhar na tela foi também utilizado para interagir com a física do cenário.

Com isto era preciso acesso aos polígonos dos modelos de forma fácil e acessível, coisa que não encontrei ferramenta que o fizesse, em particular pois os modelos deveriam ser triangularizados e todos os seus triângulos deveriam ter a mesma orientação (a dizer, sentido anti-horário) olhando por fora do objeto de forma a permitir o cálculo correto de sua seção transversal e a contribuição de cada parte de sua superfície para a força resultante.

Com isto os modelos foram feitos a mão com o auxílio de uma ferramenta que seccionava uma imagem branco e preto em triângulos de mesma orientação.

4 Linguagem e Ambiente

O projeto é multiplataforma e foi desenvolvido no seguinte ambiente:

- Linguagem: C++[1]
- SO: MacOS-X.6[11]
- Processador: Intel Core2Duo 2.26 GHz
- Memória: 2Gb RAM
- Controladora de Vídeo: NVIDIA GeForce 9400M
- Eclipse + CDT[5]: Editor de C/C++
- Eclipse + TeXlipse: Editor \LaTeX

Folhas ao Vento - Diagrama UML

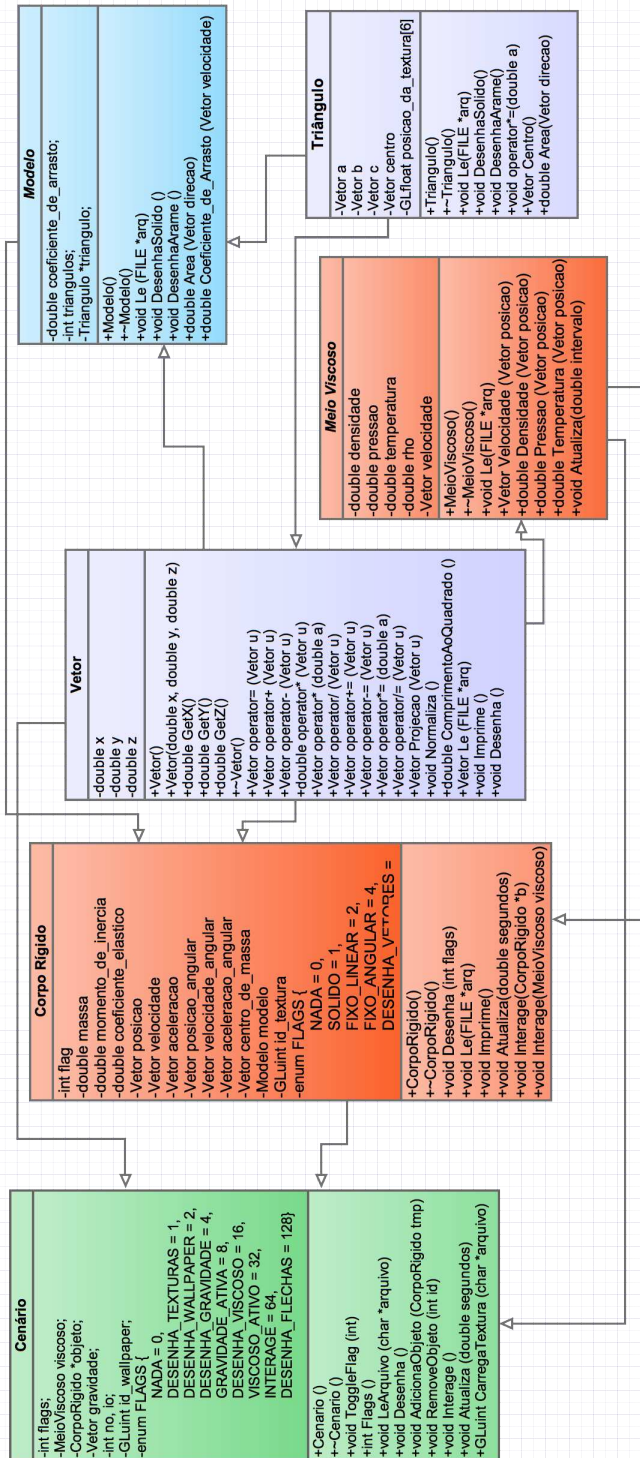


Figura 3: Diagrama UML do projeto.

- Eclipse + Subclipse + SVN + Rede-linux[7]: Controle de versão
- Compilador C/C++ MacOS-X: `i686-apple-darwin10-g++-4.2.1`
- Compilador C/C++ Linux: `i686-gnu project C and C++ 4.3`
- OpenGL[13]: Interface de Programação (API)
- Glut[12]: Sistema de janelas e eventos
- iWeb[10]: Ferramenta de criação de sítio na Internet
- Gimp[21]: Editor de imagens
- Gliffy[6]: Editor de diagramas UML

5 Atividades realizadas

5.1 Modelagem matemática

Embora as contas aqui descritas sejam conhecidas, a modelagem dos sistemas físicos foi trabalhosa e fundamental. Foram modeladas as seguinte interações:

- Aceleração constante
- Gravitação
- Arrasto
- Vento
- Vínculos⁶

5.2 Vetores

Devido a natureza do projeto foi necessário criar uma classe `Vetor`. Tal classe implementa operações atribuição, soma, subtração, produto por escalar, projeção, produto escalar, produto vetorial, normalização e comprimento com os respectivos operadores `=`, `+`, `-`, `*`, `projecao()`, `*`, `/`, `normaliza()`, `comprimento_ao_quadrado()` e é usada a exaustão por todo o projeto.

5.3 O código

As classes descritas no diagrama UML foram implementadas e são funcionais. binários e o fonte do projeto, bem como as cenas utilizadas na apresentação estão disponíveis no site[2] do projeto para download e testes.

⁶Além do escopo do trabalho e apenas parcialmente funcional. Apenas uma palmeira foi feita por ainda não ser possível a criação de qualquer vínculo entre objetos.

5.4 Modelos

Foram criados os seguintes modelos para demonstração do projeto:

- Círculo
- Esfera
- Gota d'água
- Flechas
- Folha de bordo (*Maple*)
- Palmeira

5.5 Triangulação de polígonos

Foi feito um programa externo para triangularizar imagens com um bit de cor com o intuito de facilitar a criação de modelos. O algoritmo utilizado foi retirado do primeiro capítulo do O'Rourke[19].

5.6 Leitura de cenas

Para entrada de uma cena foi implementado uma entrada por arquivo de texto. Cada linha deve começar com uma letra para informar o que será lido, seguido dos parâmetros necessários. Cada objeto da estrutura de dados sabe ler seus parâmetros de um arquivo texto. Os parâmetros de cada um estão descritos em seus métodos `Le(FILE *arq)`. As linhas possíveis são:

- `# Comentários`: Ignora tudo a partir de um `#` até o final da linha
- `o Objeto`: Acrescenta um objeto a sua cena. Você pode acrescentar quantos objetos couberem na memória de sua máquina
- `v Campo viscoso`: Ajusta os parâmetros do meio em que sua cena está inserida. Seu cenário possui apenas um campo viscoso. Caso mais de um seja especificado no arquivo de texto, o último será utilizado
- `g Gravidade constante`: Coloca uma aceleração constante para a gravidade em sua cena além da interação gravitacional entre as massas em questão
- `w Wallpaper`: Acrescenta um fundo de tela a seu cenário para torná-lo mais bonito

5.7 Flags e teclas que alteram seus estados

Algumas teclas podem alterar o comportamento do simulador. São elas:

- g: Liga e desliga a gravidade constante
- v: Liga e desliga o desenho do vento na tela
- V: Liga e desliga a interação com o vento
- f: desenha os vetores (flechas) velocidade e aceleração de cada objeto na tela em verde e vermelho respectivamente
- w: Liga e desliga o papel de parede
- i: Liga e desliga a interação e atualização do cenário
- t: Liga e desliga a textura dos objetos

6 Resultados

6.1 Animações

Para demonstração de funcionamento foram preparadas três cenas que estão disponíveis no site[2]. Uma quarta cena está em andamento⁷ e visa demonstrar o funcionamento de vínculo entre objetos. Um quadro de cenas pode ser visto na Figura 4.

6.2 Arte em tempo real

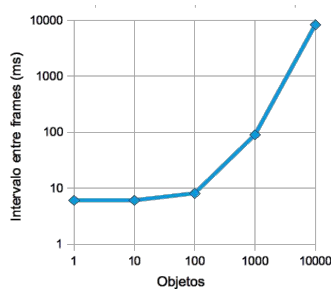


Figura 5: Desempenho do simulador. `Atualiza(double segundos)`.

O programa está pronto para servir como um cenário. Uma vez criada a cena, ela pode ser colocada dentro de um programa qualquer e fornecer características pontuais do sistema como pressão, temperatura ou velocidade do vento em algum ponto, mesmo que este não seja desenhado na tela. O estado do cenário pode ser atualizado a qualquer momento com chamadas consecutivas das funções `Interage()` e `Atualiza(double segundos)`.

⁷Apenas uma palmeira foi criada pois ainda não é possível criar vínculos quaisquer entre objetos. A palmeira é, por enquanto, uma primitiva básica do programa usada no auxílio do desenvolvimento do projeto.

Uma análise de desempenho do simulador é feita na figura 5 onde fica evidente sua natureza $\Theta(n^2)$. Vemos também neste gráfico que para 200 objetos obtemos uma simulação com 60 quadros por segundo (intervalo de aproximadamente 17ms entre frames) e portanto conseguimos simular em tempo real uma cena com 200 objetos.

6.3 Simulação industrial ou científica

Se desistirmos de simular em tempo real podemos fazer simulações mais precisas para, por exemplo, obter previsão da evolução da pressão de um sistema ou resistência plástica de uma peça⁸, podemos alimentar o programa com um modelo e condições de contorno bem precisas e atualizar seu estado fornecendo a `Atualiza(double segundos)` sempre a *mesma* pequena variação de tempo. Dependendo da quantidade de objetos, o simulador não mais conseguirá funcionar em tempo real, entretanto ele continuará sendo capaz de prever a evolução do sistema com precisão proporcional aos dados de entrada, permitindo a exploração de sistemas físicos para fins científicos e industriais de forma profissional.

6.4 Modularidade

O simulador é facilmente expansível. Tendo em mãos outros modelos de interação física tanto de objeto-cenário quanto objeto-objeto podemos facil-

⁸Para isto precisamos que as equações de vínculo estejam completamente funcionais

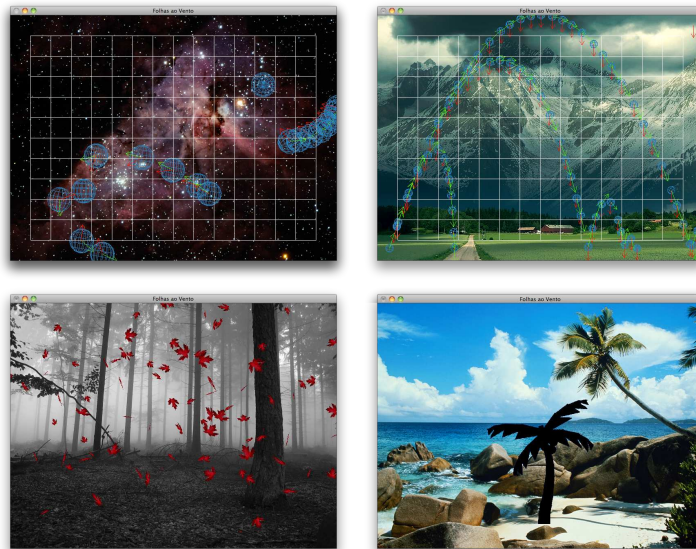


Figura 4: Quadro de cenas feitas com o simulador.

mente acrescentá-lo dentro das funções `Interage()` da classe `CorpoRigido` que automaticamente entrarão em vigor junto com a física já implementada.

7 Conclusão

Obtivemos com sucesso simulações de cenas com resistência do ar, vento e atração gravitacional.

Tais simulações podem ser usadas como arte se não nos preocuparmos com flutuações nos intervalos entre frames (permitindo assim que seja em tempo real) ou podem assumir caráter técnico científico se abrirmos mão da execução em tempo real e forçarmos intervalos constantes entre os frames fazendo com que, mesmo que cada frame demore horas para ser calculado, tenhamos ao final uma simulação realista do sistema.

O simulador pode ser facilmente expandido para tratar outros tipos de interação como, por exemplo, colisão.

Alguns algoritmos podem ser otimizados em prol da performance como, por exemplo, a implementação de uma variação do *algoritmo de pontos mais próximos*[15] para determinar quais objetos devem interagir, derrubando a complexidade $\Theta(n^2)$ do algoritmo utilizado.

8 Desafios passados e futuros

Os maiores desafios no desenrolar do projeto foram:

- Tempo insuficiente para colocar todas as idéias em prática. Infelizmente o TCC é desenvolvido junto com outras matérias, o que torna difícil a administração do tempo.
- Identificar erros de arredondamento onde intervalos muito pequenos faziam o sistema não evoluir como o esperado pois $dt_{peq} \rightarrow dx = 0$.
- Newton \times Hamilton: Entender qual modelagem era mais adequada e por quê. A Newtoniana foi escolhida pois, além de ser a vista no curso de computação, é também mais adequada para o trabalho vetorial desenvolvido.
- Modelo e sistema físico: Utilizar o mesmo modelo geométrico para desenhar na tela e interagir com o sistema físico do simulador.
- Criação de modelos: Bibliotecas como OGRE, que são capazes de ler modelos feitos no blender ou 3D Studio, não nos dão acesso fácil aos polígonos e nem garantem a orientação de seus triângulos necessária, o que tornou a criação de modelos uma tarefa manual e muito trabalhosa.
- Triangulação de polígonos.

Por outro lado, algumas coisa não puderam ser implementadas... Ainda. São elas:

- Colisões
- Vínculos arbitrários entre objetos
- Tratamento de líquidos
- Uma melhor modelagem do vento utilizando conceitos de termodinâmica[18]
- Criação de mais modelos
- Tratamento preciso de colisões

9 Disciplinas relevantes para o TCC

Muitas disciplinas foram essenciais para a elaboração deste trabalho. Cito aqui cada uma delas e sua contribuição.

`MAC0122` `Princípios de Desenvolvimento de Algoritmos`: Forneceu estruturas de dados e algoritmos básicos para toda a modelagem computacional do projeto.

`MAT0139` `Álgebra Linear para Computação`: Consolidou operações com vetores.

`FAP0126` `Física I`: Forneceu a base teórica para modelar toda a simulação do projeto.

`MAC0300` `Métodos Numéricos da Álgebra Linear`: Forneceu ferramentas para resolver sistemas lineares do simulador.

`MAC0338` `Análise de Algoritmos`: Forneceu forma de avaliar a eficiência dos algoritmos implementados.

`MAC0332` `Engenharia de Software`: Forneceu a metodologia que possibilitou desenvolver o projeto de forma eficaz.

`MAC0331` `Geometria Computacional`: Forneceu noções de como tratar problemas geométricos com o computador tais como pontos mais próximos e triangulação de polígonos.

`MAC0420` `Introdução à Computação Gráfica`: Forneceu ferramentas para desenhar com o computador e noções importantes sobre espaços afins.

10 Considerações finais

Para a realização deste projeto foi utilizado um conteúdo enorme de informações presentes em diversas matérias do curso que me eram antes desconhecidas e agora fazem parte de meu dia a dia.

Hoje fica para mim evidente o quanto o curso de computação agrega, tanto em conhecimentos teóricos quanto práticos.

11 Agradecimentos

Deixo aqui os meus mais sinceros agradecimentos a minha família, amigos e professores.

- A família pelo suporte incondicional
- Aos amigos por fazerem a vida valer a pena
- Aos professores do DCC pela motivação, exemplo profissional e por receber todos os alunos da computação com alegria e entusiasmo que não conheci em outro lugar desta universidade



Referências

- [1] Dennis M. Ritchie Brian W. Kernighan. *The C Programming Language (K&R) 2 edition*. Prentice Hall, 1988.
- [2] J.P. Kerr Catunda. Folhas ao vento, 2010. <http://www.linux.ime.usp.br/~jpkc/mac499>.
- [3] Jearl Walker David Halliday, Robert Resnick. *Fundamentals of Physics, 7th Edition*. John Wiley & Sons, 2005.
- [4] Paulo Feofiloff. Tabelas dinâmicas, 2010. http://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/tabelas-dinamicas.html.
- [5] The Eclipse Foundation. Eclipse helios (3.6.1), 2010. <http://www.eclipse.org/>.
- [6] Gliffy. Gliffy diagram software, 2010. <http://www.gliffy.com/>.
- [7] REDE PRÓ-ALUNO GNU/LINUX. Rede linux, 2010. <http://www.linux.ime.usp.br/>.
- [8] Hamilton Luiz Guidorizzi. *Um Curso de Cálculo (Vol. 1, 2 e 3), 5a. Edição*. LTC, 2001.
- [9] Albert Herr Howard Anton. *Calculus With Analytic Geometry, 5th Edition*. John Wiley & Sons, 1995.
- [10] Apple Inc. iweb 3.0.2, 2009. <http://www.apple.com/ilife/iweb/>.
- [11] Apple Inc. Mac os x snow leopard, 2010. <http://www.apple.com/macosex/>.
- [12] SGI Silicon Graphics International. The opengl utility toolkit, 2010. <http://www.opengl.org/resources/libraries/glut/>.
- [13] Mason Woo Jackie Neider, Tom Davis. *OpenGL Programming Guide – The Official Guide to Learning OpenGL, Release 1*. Addison Wesley, 1995.
- [14] David Bau III Lloyd N. Trefethen. - *Numerical Linear Algebra*. SIAM, 1997.
- [15] D. Hoey M. I. Shamos. *Closest-point problems*. Proc. 16th Annual IEEE Symposium on Foundations of Computer Science, 1975.
- [16] Sir Isaac Newton. *Philosophiæ Naturalis Principia Mathematica*. The Royal Society, borne by Edmund Halley, 1687.

- [17] Alexandre Noma. Caricatura do prof. coelho, 1999.
<http://www.vision.ime.usp.br/~noma/index.html>.
- [18] Moysés Nussenzveig. *Um curso de física básica (Vol. 1)*. Edgard Blücher, 2008.
- [19] Joseph O'Rourke. *Computational Geometry in C*. Cambridge University Press, second edition, 2008.
- [20] Jerry B. Marion Stephen T. Thornton. *Classical Dynamics of Particles and Systems, 5th Edition*. Brooks Cole, 2003.
- [21] The GIMP Team. Gnu image manipulation program 2.6.10, 2010.
<http://www.gimp.org/>.