Notas em Matemática Aplicada e-ISSN 2236-5915

Volume 67, 2012

Editores

Cassio Machiaveli Oishi
Universidade Estadual Paulista - UNESP
Presidente Prudente, SP, Brasil

Fernando Rodrigo Rafaeli
Universidade Estadual Paulista - UNESP
São José do Rio Preto, SP, Brasil

Rosana Sueli da Motta Jafelice (Editor Chefe)
Universidade Federal de Uberlândia - UFU
Uberlândia, MG, Brasil

Rubens de Figueiredo Camargo
Universidade Estadual Paulista - UNESP
Bauru, SP, Brasil

Sezimária de Fátima P. Saramago
Universidade Federal de Uberlândia - UFU
Uberlândia, MG, Brasil

Vanessa Avansini Botta Pirani (Editor Adjunto)
Universidade Estadual Paulista - UNESP
Presidente Prudente, SP, Brasil

Sociedade Brasileira de Matemática Aplicada e Computacional

2012

A Sociedade Brasileira de Matemática Aplicada e Computacional - SBMAC publica, desde as primeiras edições do evento, monografias dos cursos que são ministrados nos CNMAC.

Para a comemoração dos 25 anos da SBMAC, que ocorreu durante o XXVI CNMAC em 2003, foi criada a série **Notas em Matemática Aplicada** para publicar as monografias dos minicursos ministrados nos CNMAC, o que permaneceu até o XXXIII CNMAC em 2010.

A partir de 2011, a série passa a publicar, também, livros nas áreas de interesse da SBMAC. Os autores que submeterem textos à série Notas em Matemática Aplicada devem estar cientes de que poderão ser convidados a ministrarem minicursos nos eventos patrocinados pela SBMAC, em especial nos CNMAC, sobre assunto a que se refere o texto.

O livro deve ser preparado em **Latex (compatível com o Miktex versão 2.7), as figuras em eps** e deve ter entre **80 e 150 páginas**. O texto deve ser redigido de forma clara, acompanhado de uma excelente revisão bibliográfica e de **exercícios de verificação de aprendizagem** ao final de cada capítulo.

Veja todos os títulos publicados nesta série na página
http://www.sbmac.org.br/p_notas.php

SBMAC **Sociedade Brasileira de Matemática Aplicada e Computacional**

2012

ANIMAÇÃO EM 3D USANDO JAVA

Enivaldo Bonelli
bonelli@ufrnet.br

Departamento de Geofísica
Centro de Ciências Exatas e da Terra
Universidade Federal do Rio Grande do Norte

Sociedade Brasileira de Matemática Aplicada e Computacional

São Carlos - SP, Brasil
2012

Coordenação Editorial: Elbert Einstein Nehrer Macau

Coordenação Editorial da Série: Rosana Sueli da Motta Jafelice

Editora: SBMAC

Capa: Matheus Botossi Trindade

Patrocínio: SBMAC

Copyright ©2012 by Enivaldo Bonelli. Direitos reservados, 2012 pela SBMAC. A publicação nesta série não impede o autor de publicar parte ou a totalidade da obra por outra editora, em qualquer meio, desde que faça citação à edição original.

Catalogação elaborada pela Biblioteca do IBILCE/UNESP
Bibliotecária: Maria Luiza Fernandes Jardim Froner

Bonelli, Enivaldo
Animação em 3D usando Java - São Carlos, SP : SBMAC, 2012, 84 p., 21.5 cm - (Notas em Matemática Aplicada; v. 67)

e-ISBN 978-85-8215-028-3

1. Animação Gráfica 2. Programação em Java
3. Gráficos em 3D
I. Bonelli, Enivaldo. II. Título. III. Série

CDD - 51

Dedicado a

Regina e Marília

Agradeço aos incansáveis programadores que escrevem códigos gratuitos para LINUX, assim como aos criadores e mantenedores do sistema operacional gratuito UBUNTU. Especificamente, esse trabalho foi facilitado devido ao Kile, um ambiente integrado para LaTeX e ao software GIMP, de tratamento gráfico. Mais importante, claro, foi a distribuição gratuita do JAVA, pela ORACLE.

Conteúdo

Prefácio **11**

1 Introdução **13**
1.1 Applets 14
1.2 Programando em Java 14
1.3 Compilação dos programas 15
1.4 Os códigos dos programas 15

2 Nossos Primeiros Gráficos **17**
2.1 Primeiro passo 17
2.1.1 Uma classe que não faz nada 17
2.1.2 Desenhando ovais 19
2.1.3 Visualizando seus gráficos 20
2.1.4 Usando cores 22
2.2 Transição para 3D 24

3 Gráficos em 3D com $3_{DJA}H$ **27**
3.1 Girando curvas inteiras 31

4 A Biblioteca $3_{DJA}H$ **33**
4.1 Classe vetor 33
4.2 Classe opv 35

5 Animações **37**
5.1 Introdução 37
5.2 Inputs básicos de $3_{DJA}H$ 38
5.3 Exemplos simples 38
5.4 Curvas e trajetórias 45
5.4.1 O vetor deslocamento 46
5.4.2 Traçando curvas em 3D 46

5.5 As “partículas” da Física 48
5.6 Desenhando o nome dos vetores 52
5.7 Desenhando escondido do usuário 55
5.8 Girando curvas 57

6 Projetos Futuros **61**

Apêndices **62**

A Outros Gráficos **63**
A.1 Eixos cartesianos 63
A.2 Esfera $3_{DJA}H$ 65

B Códigos da Biblioteca **67**
B.1 Código da classe vetor 67
B.2 Código da classe opv 72

Bibliografia **83**

Prefácio

Esse livro se originou de vários mini-cursos, palestras e animações em minhas páginas, na internet. A idéia básica por trás de minha criação de uma biblioteca para animações em 3D foi escrever códigos pequenos, dirigidos para a comunidade de ciências exatas, que conhece o conceito de vetores. Outros programas, para 3D, disponíveis, não pode exigir do usuário, em geral artistas, que dominem essa área – e muito menos que sejam programadores. Assim, minha tarefa foi simplificada devido a uma maior exigência com relação à clientela.

A segunda idéia é que os usuários não precisassem conhecer muito bem a linguagem Java. Aqui sempre será possível se construir outras animações, usando-se um modelo, já pronto, de comandos de cabeçalho e finalização, com um corpo principal, em que a animação é desenvolvida. O usuário não precisa entender os detalhes dos comandos preliminares e finais, enquanto que o corpo é devidamente explicado.

A terceira idéia básica é de que os usuários não precisem de computadores sofisticados. Até um PC 386 pode rodar esses códigos, sem necessidade de nenhuma configuração especial. Pode-se fazer animações bem complexas, sem se sobrecarregar a memória RAM, nem exigir muito do processador.

Capítulo 1

Introdução

"O JAVA é um ambiente sofisticado e extensível, que oferece os alicerces para a construção de aplicativos de qualidade industrial." Essa é, para mim, a melhor forma de expressar o que é o Java. Ela foi usada por Hopson e Ingram em seu excelente livro sobre o desenvolvimento de applets com Java, em 1997 [2]. Como se transcorreu muito tempo, medido pelo relógio do avanço tecnológico, já dá para se ver que os aplicativos em Java tomaram o mundo. Até a declaração do imposto de renda já tem sua versão em Java. Já existem aplicativos em Java, para celulares Android e aparelhos semelhantes.

Não estou interessado em entrar em detalhes sobre o Java, mas em ensinar o pouco necessário para que possamos, autor e leitor, nos divertirmos com animações tridimensionais, de preferência fora da tela do computador ou da tela de projeção. Para que esse fim seja atingido, o leitor terá que aceitar certas explicações simplistas sobre alguns comandos e procedimentos. Isso porque o estudo detalhado do funcionamento do Java tomaria vários volumes, cada um maior que o tamanho deste livro. Mesmo ensinando comandos de maneira simplista, passarei o conceito correto, ao leitor, de forma bem didática, mas sem explicar o que acontece nas entranhas da máquina virtual do Java, ou nas entranhas do computador.

Para o leitor mais interessado, o livro de Hopson e Ingram deve ser lido, apesar de já ser "antigo", pois eles explicam tudo de uma maneira deliciosa. E sabem escrever e ensinar, uma habilidade que a maioria dos programadores está perdendo. Livros mais completos e mais modernos, tais como o de Deitel e Deitel ([1]) já discutem o 'swing', que permite se comunicar melhor com o usuário. Mas não cobrem o assunto da programação 3D, claro.

1.1 Applets

Nós estamos interessados em Applets, que são programas que podem ser executados via Web, independentemente do sistema operacional usado (Windows, Linux, Mac, etc.) Quando você programa um Applet, então, não precisa se preocupar em fazer um programa para cada sistema operacional. Para visualizar sua Applet, claro, o usuário precisa instalar um 'plugin' do Java, que é gratuito. Atualmente, para acessar sua conta corrente, no Banco do Brasil, você precisa instalar um 'plugin Java'. Esse mesmo plugin serve para você visualizar os Applets Java que criaremos durante o curso.

Os Applets podem consistir de texto e figuras estáticos, como botões e outros controles, através dos quais o usuário pode controlar a evolução do programa. Nosso principal objetivo não é usar botões e outros controles, mas sim desenhar sequências de figuras, que se sucedem, dando a impressão de movimento. Isso tudo em 3D, quando o usuário usar óculos 3D simples, aqueles com "lentes" vermelho-azul, que podem ser confeccionados com papel celofane. Também podem ser comprados prontos.

1.2 Programando em Java

A programação Java é orientada a objetos, o que tem algo a ver com objetos da vida real. Primeiro, ensinamos o programa a construir um objeto. Por exemplo: os objetos que mais usaremos, nesse curso, são os vetores matemáticos, com três componentes cartesianas digamos x, y, e z. Dizemos ao programa que o objeto tem 3 componentes, que são números reais. Esse objeto (vetor) ainda tem uma propriedade chamada módulo, que é obtido a partir de suas componentes. Podemos prosseguir, definindo as componentes do vetor em coordenadas esféricas e cilíndricas. Um conceito útil é o vetor unitário, paralelo a um dado vetor. Vamos dizer que o vetor unitário é propriedade do dado vetor. Podemos até incluir uma operação de rotação do vetor, em torno de um eixo cartesiano, como sendo uma propriedade do mesmo.

Como codificamos tudo isso, dentro de um programa, i.e., como definimos um objeto vetor e como acessamos suas propriedades? Isso é feito usando-se o conceito de classes . As classes contém a receita para a definição do objeto e de como se acessar suas propriedades. Assim, quando escrevemos

```
Vetor A, B, C
```

Estamos declarando que A, B, e C são vetores, com todas propriedades definidas pela classe vetor. A forma que essas atribuições são passadas a A, B, e C, é explicada na classe Vetor. Dentro da classe Vetor, dizemos que o módulo do vetor é a raiz quadrada da soma dos quadrados das componentes cartesianas, e assim

por diante. O leitor não terá que se preocupar como isso é feito, pois só terá que usar a declaração "Vetor A, B, C". Se a curiosidade for grande, poderá ver como faço isso, pois o código está incluído no final do livro.

1.3 Compilação dos programas

O Java tem evoluído rapidamente, podendo acontecer que alguns comandos que usei, nos códigos, fiquem ultrapassados (deprecated.) Mas não se preocupe com isso, pois o Java sempre informa quais os comandos que foram atualizados ou substituídos por outros, na hora da compilação. Nesse e na maioria dos casos, é mais fácil consultar o Google, por soluções de seus problemas do que consultar o tutorial do oficial do Java, que é muito complicado, na minha opinião.

O Java roda em uma "máquina virtual" que independe do sistema operacional. Assim, você pode compilar um programa em uma máquina com um sistema operacional e rodar em outra, com outro sistema operacional.

Uma observação sobre versões: deve-se tomar o cuidado de se compilar as classes e os programas que as invocam, com a mesma versão do Java. No nosso caso, isso se aplica aos diversos programas e às classes VETOR, OPV, e OUTROS. De qualquer forma, o compilador chamará sua atenção para isso.

Os códigos dos exemplos usados nesse livro e suas versões compiladas estão no CD. Para ver as animações, basta clicar nos arquivos HTML. Ficarei muito feliz se você fizer suas próprias animações, usando a biblioteca $3_{DJA}H$.

Os códigos dos exemplos do livro também se encontram nos apêndices, junto com outros códigos de gráficos que não foram discutidos no texto principal.

Cuidado! Nas listagens impressas, tive que separar alguns comandos em duas linhas, por causa da largura da página. Isso pode ser feito, em Java. Entretanto, posso ter dividido linhas onde não é permitido. Assim, para compilação, valem os códigos do CD.

1.4 Os códigos dos programas

Caso você não tenha o CD com os códigos das bibliotecas e dos exemplos do livro, poderá conseguí-los, gratuitamente, nos meus sites, *depois* da publicação desse livro pela SBMAC, nos meus sites:

1. `http://geofisica.ufrn.br/~bonelli`
2. `http://dinamo.geofisica.ufrn.br/bonelli`

Se esses sites não estiverem disponíveis, por alguma razão imprevista, tentarei sempre mantê-los na internet, sob o nome "3DJAH", o que será fácil localizar com o Google ou similares.

Capítulo 2

Nossos Primeiros Gráficos

Como mencionei, no capítulo anterior, a programação gráfica, em Java, para a internet, usa Applets, que são janelas gráficas onde se pode fazer praticamente qualquer coisa. Podemos criar botões e outras ferramentas com as quais o usuário controla a execução do programa, usando o mouse, o teclado, ou qualquer outra interface. Aqui, vamos fazer nosso primeiro gráfico em 3D. Não de uma só vez, mas passo a passo, para entender cada linha do programa que gera a applet. Iniciaremos com gráficos 2D e evoluiremos para 3D.

2.1 Primeiro passo

2.1.1 Uma classe que não faz nada

Criemos uma CLASSE que define todos os procedimentos que queremos usar: num editor de texto crie o seguinte arquivo, com nome

"primeiroPasso.java".

Tem que ser em texto puro, tais como o do notepad ou gedit (ou vi). Não se esqueça de salvá-lo, com o nome primeiroPasso.java.

```
//início do programa
public class primeiroPasso extends Applet {
// A linha acima define a classe
// Isso é um comentário
// O programa não tem o corpo.
}//fim do programa
```

As chaves "{" e "}" são obrigatórias. Elas marcam o início e fim de cada "fase" de programação. Nesse caso marcam o início e o fim da definição da nossa classe.

Vamos analisar o que fizemos:

Tentamos criar uma classe "pública", para que outros programas possam usá-la. Essa classe,"primeiroPasso.class", que será gerada depois de uma compilação bem sucedida, já vai usar todas as funções que o Java fornece com a classe Applet, daí usarmos o termo "extends", para dizer que nossa classe vai herdar as funções de Applet. Por exemplo, se Applet tem a função "apagar" a área de desenho, nosso primeiroPasso também poderá usar essa função. Isso se chama "herança" assim como em C++.

Para não nos perdermos, durante a escrita e análise de programas longos, é um bom costume se usar comentários, dentro do código. Os comentários são precedidos por duas barras de divisão seguidas (//). Tudo que vier, na mesma linha, depois dessas barras, deve ser ignorado, pelo compilador.

Agora, vamos à compilação. Dentro do diretório em que está seu código Java, digite, na linha de comando,

```
javac primeiroPasso.java
```

Isso fará o compilador Java analisar e compilar seu programa. Infelizmente, o nosso programa simples não compila corretamente: aparecem mensagens de erro. Você acha que deu errado porque nós não mandamos o programa fazer nada... Não, o Java não se preocupa que escrevamos programas que não fazem nada! O que falta é dizermos onde está a classe Applet, para que possa usar. Dessa forma, o Java evita que se carregue, na memória, *tudo que há disponível*, o que acarretaria uma perda de espaço e de tempo. Então, para dar todas informações necessárias, mudamos o código, assim:

```
//início do programa
import java.applet.*;
public class primeiroPasso extends Applet {
}//fim do programa
```

Note o ";" (ponto e vírgula) no final do comando import. Todos os comandos, em Java, são terminados por ";". Assim, um comando pode se extender por várias linhas, sem causar confusão. Os pontos depois de 'java' e depois de 'applet', representam subdiretórios. O asterisco, '*', representa 'tudo'. A classe 'Applet', com 'A' maiúsculo, está dentro do diretório 'applet', mas acabamos importando outras classes, por comodidade. Agora, compilamos novamente. Digite, na linha de comando:

```
javac primeiroPasso.java
```

Note que, agora, você não recebe nenhuma mensagem de erro e, além disso, aparece um arquivo “primeiroPasso.class”, no seu diretório. O nosso objetivo, aqui, é criar Applets que possam ser acessadas via internet. Para isso, seu program, ou classe, tem que ser chamado por um navegador de internet. Você pode navegar, também, no seu próprio computador, para testar seus códigos, antes de divulgá-los na internet. Mas não temos o que testar ainda, não é? Nosso código compila, mas não faz nada.

2.1.2 Desenhando ovais

Para melhorar o fato de que nossa classe 'não faz nada', vamos criar uma região, onde possamos desenhar. Mas não desenharemos, ainda. Se escolhermos um retângulo de 200 por 100 pixels. então, o código fica:

```
//início do programa
import java.applet.*; //linha 1
public class primeiroPasso extends Applet { //linha 2
   public void init { //linha 3
      resize(200,100);//linha 4
   } // linha 5
}//fim do programa - linha 6
```

Vamos comentar esse código, linha por linha:

linha 1 Importamos “*”, ou seja, todas as funções disponíveis no diretório

“java.applet”,

inclusive a classe “Applet”, com “A” maiúsculo, de quem precisávamos acima. Para o Java, o “.”(ponto) indica subdiretórios.

linha 2 Definimos a nossa classe e dizemos que ela herda funções de Applet.

linha 3 Usamos a função “init()”, de Applet, que permite darmos parâmetros iniciais do nosso painel para desenho.

linha 4 Mandamos init() usar um tamanho do painel para desenho, em pixels.

linha 5 Fim do init

linha 6 fim da nossa classe, primeiroPasso.

Finalmente, vamos fazer algo. Vamos desenhar na área preparada por init. Que tal desenhar uma circunferência? Para desenhar, chamamos a função “paint”, que permite desenharmos na área preparada. Dentro do “paint”, podemos usar todas funções do Java, para Applet. A função, para desenhar uma circunferência, é chamada assim:

```
drawOval(int x, int y, int largura, int altura);
```

Note que a função desenha ovais, onde x e y são as posições, na tela, em pixels, do vértice superior esquerdo do retângulo, dentro do qual será desenhada a oval. Os outros dois parâmetros são a largura e altura desse retângulo. Para largura=altura, temos uma circunferência. Cuidado: normalmente, em gráficos em tela de computador, a origem se encontra no lado esquerdo superior da tela. O eixo x vai para a direita e o eixo y vai para baixo. Na minha biblioteca 3D, conserto isso, colocando a origem do lado esquerdo inferior, da tela. Os símbolos "int" são para lembrar que os argumentos da função são número inteiros. Não os usamos, na hora de chamar a função, mas o compilador reclama se colocarmos não inteiros, lá. Vamos ao nosso desenho, então, lembrando que nosso "papel" para desenho tem tamanho (200 x 100 pixels):

```
import java.applet.*;\\Applet está aqui
import java.awt.*; \\Graphics está definido aqui
public class circunferencia extends Applet{
  public void init{ //inicia o ambiente grafico
    resize(200,100);//Tamanho da área de desenho
  } //fim do init
     public void paint(Graphics meuGrafico){
        //Diz que meuGrafico tem propriedades gráficas
        meuGrafico.drawOval(75,25,50,50);
        //Desenha oval em meuGrafico
     } //fim do paint
}//fim do programa
```

Copie esse código, no seu editor de texto, salve-o com o nome "circunferencia.java", e compile-o, com o comando

```
javac circunferencia.java
```

2.1.3 Visualizando seus gráficos

Se a compilação tiver sucesso, um arquivo circunferencia.class aparecerá, no seu diretório. Agora, vamos executar esse código em um navegador ou usando o "appletviewer", que é um visualizador de Applets, do Java. Com ele, você testa seus códigos, sem precisar de um navegador. Como a maioria já tem um navegador, vamos aprender como chamar um arquivo.class de dentro de um navegador.

É óbvio que não podemos mandar o navegador carregar um arquivo com extensão ".class" diretamente. O navegador foi feito para navegar por arquivos HTML e similares. Aqui é que está a parte interessante: se você tem um texto, em HTML, pode chamar um arquivo tipo ".class" de dentro desse HTML, e ter

figuras desenhadas, na hora, junto com seu texto. Isso é uma coisa revolucionária e fica mais interessante quando as figuras são animações que podem ser controladas pelo usuário, com o uso do mouse, por exemplo.

Então, vamos lá: o que carregamos, usando o navegador (no meu caso, uso o navegador gratuito Firefox), é um arquivo em HTML. Vamos chamá-lo de "circunferencia.html". Dentro desse arquivo, encontra-se a chamada para a classe que executará o desenho. Note que um mesmo arquivo HTML pode chamar muitas classes com animações, sendo que todas poderiam ser executadas simultaneamente. O código HTML, que chama uma classe, deve ser escrito em texto puro (não pode conter caracteres escondidos, que alguns editores de texto colocam.) Veja, como exemplo, o código HTML que chama a nossa circunferencia.class.

```
<html>
<head>
<title> circunferencia</title>
</head>
<body>
<p align="center">
<applet code="circunferencia.class" align="middle"
width="200" height="100">
Sinto muito - o seu navegador nao tem java plugin...
</applet> </p>
</body>
</html>
```

Isso não são comandos Java: são comandos HTML. Aqui se começa um ambiente com <isso> e se termina com </isso>. Quase todos HTML que usaremos serão semelhantes a este. Asim, poderemos sempre usá-lo novamente, mudando apenas o nome da classe e o título do Applet. Salve esse arquivo com o nome circunferencia.html, no mesmo diretório em que está o circunferencia.class. Para ver sua circunferência, abra o arquivo circunferencia.html no seu navegador ou use o comando

```
appletviewer circunferencia.html
```

Na figura 2.1, mostro uma cópia da tela, de como a nossa circunferência aparece, na área do navegador. No meu caso, a figura aparece em cor preta, sobre um fundo cinza. Isso pode ser mudado, e aprenderemos, como, em breve.
Se usarmos o appletviewer, obtemos uma tela do tamanho especificado no arquivo HTML (Fig. 2.2). A tela do appletviewer tem várias opções, inclusive o de imprimir a figura, no formato Postscript (ps) para arquivo, ou imprimir diretamente pela impressora. Uma opção interessante, do appletvier é o de clonagem da imagem. Voce pode fazer quantas cópias quiser, da tela do appletviewer. Todas aparecem

Figura 2.1: Circunferência, desenhada através do carregamento de circunferencia.html, pelo navegador

simultaneamente, na tela de seu computador. Parece que não há nada de interessante aí, não é? Mas há: quando estamos trabalhando com uma animação e fazemos uma clonagem, via appletviewer, podemos ver duas (ou mais) animações em fases diferentes, ou seja, as coisas não acontecem simultaneamente nos vários clones!

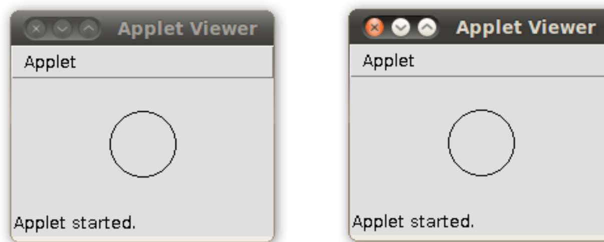

Figura 2.2: Circunferência, desenhada através do carregamento de circunferencia.html, pelo appletviewer, e uma clonagem do Applet. Qual é o clone?

Nessa altura, é interessante que você faça alguns exercícios, para praticar.

Exercício 1. *Modifique o código do desenho da circunferência para que ele desenhe uma oval também. Por exemplo, uma oval, com semi-eixo maior que o raio da circunferência, com a circunferência em seu centro.*

Exercício 2. *Modifique o código do programa da circunferência, para desenhar várias circunferências, de diversos tamanhos, em diferentes posições. Se precisar, pode aumentar o tamanho da tela gráfica.*

2.1.4 Usando cores

Para mudar a cor de fundo e para mudar a cor do desenho, podemos usar o mesmo procedimento: mudamos a cor da “caneta” que está desenhando. Não se

pode esquecer que, depois de mudada, essa será a cor usada, até que seja efetuada uma nova mudança. O comando para se definir a cor azul, por exemplo, é:

```
setColor(Color.blue);
```

Outras cores comuns são “white”, “black”, “yellow”, “red”, e “green”. Procurando, no Google, com o texto “cores em Java”, você verá todas. O Java permite, também, que trabalhemos com tonalidades de cores, onde ajustamos a quantidade de vermelho, verde e azul, de um “pixel”. Com esse procedimento, podemos gerar uma infinidade de cores, mas não vamos usar isso, nesse texto.

Vamos usar cores de duas formas. Primeiro, vamos desenhar um retângulo branco, para ser usado como fundo, para nosso desenho. Isso garante que o fundo será branco, não importando qual seja o default de um dado navegador. Você viu que, no meu navegador, o default (padrão) de fundo é o cinza. A segunda maneira de usar cores, será definir a cor com a qual desenharemos nossa circunferência. Vejamos o código:

```
import java.applet.*;
import java.awt.*;
public class circCor extends Applet{
    public void init(){ //inicia o ambiente grafico
       resize(200,100);//Tamanho da área de desenho
    } //fim do init
    public void paint(Graphics meuGrafico){
        //Diz que meuGrafico tem propriedades gráficas
        meuGrafico.setColor(Color.white);
        //Atribui cor branca
        meuGrafico.fillRect(0,0,200,100);
        // Preenche o retângulo com a cor atual
        meuGrafico.setColor(Color.blue);
        //Atribui cor azul
        meuGrafico.drawOval(75,25,50,50);
        //Desenha oval em meuGrafico
    } //fim do paint
}//fim do programa
```

O resultado é mostrado na figura 2.3, onde a tela do appletviewer é mostrada sobre a tela do terminal, onde se mostra o comando que a executou e uma listagem do programa Java que criou a figura.

Exercício 3. *Modifique o código acima, para que o retângulo seja branco, com a borda cinza (grey) e, dentro dele, desenhe uma oval vermelha.*

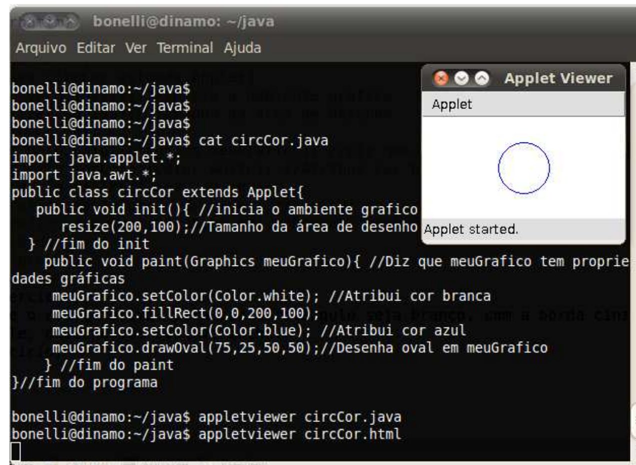

Figura 2.3: Circunferencia, em cor azul, com fundo branco. Ao fundo, a tela do terminal que gerou o comando. Note um comando errado, na penúltima linha do terminal: tentei usar 'appletviewer' com um arquivo '.java'!

2.2 Transição para 3D

Você notou, na seção anterior que, para desenharmos uma oval, precisamos dar a localização do retângulo dentro do qual ela será desenhada. Isso é feito fornecendo-se as coordenadas (x,y) do canto esquerdo superior, do retângulo, na tela Para desenhar em 3D, você deverá fornecer coordenadas para fora da tela (eixo x), para a direita (eixo-y) e para cima (eixo z). Você poderá fornecer coordenadas esféricas ou cilíndricas, ao invés das cartesianas. A figura 2.4 ilustra quatro conjuntos de eixos cartesianos, cada um com origem em lugares diferentes do campo de visão (agora, espacial.) Depois veremos o código de confecção dessa figura. Nela, as quatro origens estão atrás da tela do computador mas dois dos eixos saem da tela, ou da página, quando você usa seus óculos 3D.

Como foram confeccionados esses sistemas de eixos? Primeiro, definimos 3 vetores, que serão os eixos x = (200,0,0), y=(0,200,0) e z=(0,0,200), onde suas componentes são medidas em pixels. Depois, desenhamos esses vetores a partir de quatro diferentes origens, no espaço. E como especificamos as origens, no espaço? As quatro diferentes origens são extremidades de vetores FIXOS no canto esquerdo da tela, que é chamado de ORIGEM ABSOLUTA, nesse livro. Veja a ilustração, também desenhada com o $3_{D_JA}H$. Isso tudo funciona, porque nossos vetores são desenhados como segmentos de reta, sem a seta, que caracteriza o desenho de vetores, e que indica seu sentido. Apróxima versão do programa conterá essa

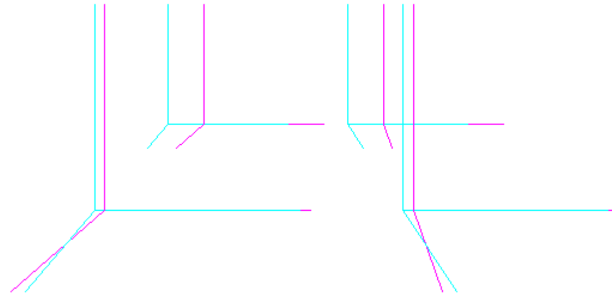

Figura 2.4: Quatro sistemas de eixos cartesianos com origens em diferentes pontos do espaço.

opção, de desenhar as setas.

Desenhando letras

Seria interessante indicar os eixos, acima, por letras x, y e z. Isso é possível. Pode-se desenhar texto na ponta de um dado vetor, com tamanho escolhido. A fonte padrão já está escolhida, mas o usuário pode escolher o tamanho da fonte. A perspectiva 3D funciona para o texto também. Se um vetor estiver girando em torno da vertical, o texto associado a ele girará também, mudando de tamanho quando se aproximar ou se afastar do observador, mas sempre ficará paralelo ao plano da tela, podendo ser sempre lido.

Desenhando ovais

Para animações de partículas, em Física, pode-se usar ovais dadas pelo programa, onde se escolhe o tamanho. Essas, entretanto, são sempre paralelas ao plano da tela. Ovais que giram, terão que ser contruidas com pequenos segmentos de reta (vetores, para nós), que giram. Parece difícil, mas não é.

Vamos à prática

No próximo capítulo começaremos a confeccionar nossas figuras 3D e, no capítulo seguinte, aprenderemos a criar movimento ou animação dessas figuras.

Capítulo 3

Gráficos em 3D com $3_{D_J A}H$

Um Applet contém um ambiente adequado para se desenhar. Até o texto que lá aparecerem serão desenhos. Assim, você pode desenhar um texto sobre outro, bastando especificar onde quer começar, o tamanho da fonte, cor, etc. O mais interessante, claro, é se desenhar figuras. Ovais e retângulos já vem no programa Java original – mas só para duas dimensões, i.e., para o plano da tela. Para se desenhar figuras mais complicadas, tal como uma curva em duas dimensões, precisamos construí-la a partir de segmentos de reta. Quanto menores esses segmentos, mais suave parecerá a curva. Como um exemplo, veja a figura abaixo, que foi gerada dessa forma. Olhando-se com cuidado, consegue-se ver os pequenos segmentos de reta (Fig. 3.1).

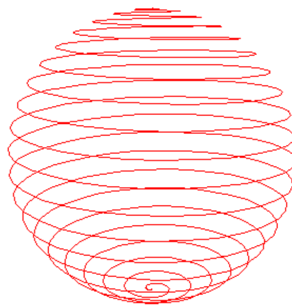

Figura 3.1: Esfera, desenhada com pequenos segmentos de reta

Os desenhos tridimensionais, desse pacote, são obtidos através da paralaxe da visão binocular: um olho vê uma imagem e o outro olho vê outra imagem, um

pouco diferente. Cada imagem é o que cada olho veria, quando o usuário estiver a uma distância determinada da tela, de um objeto tridimensional. Se o usuário se afastar da tela, obterá efeitos interessantes, tal como a imagem sair mais ainda da tela, quando é uma imagem fora da tela.

Como é feito o truque? Na ilustração (Fig. 3.2), consideramos um usuário observando um pequeno objeto, que está "fora da tela".

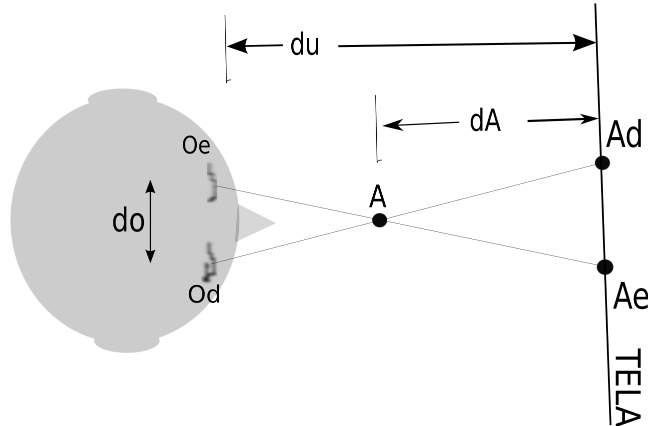

Figura 3.2: Como cada olho de um usuário "projeta" um objeto externo, A, na tela. Ae é a projeção causada pelo olho esquerdo e Ad a causada pelo olho direito. Note que a projeção referente ao olho esquerdo fica à direita do usuário, e a projeção causada pelo olho direito fica à sua esquerda.

Imaginando que cada olho do usuário projete a imagem do ponto A sobre a tela, vemos que a projeção devida ao olho esquerdo fica do lado direito do usuário, e a projeção devida ao olho direito, fica à esquerda do usuário. Então, o "truque", para causar o efeito de um ponto fora-da-tela é se desenhar as imagens para cada olho da maneira descrita acima. No caso de um objeto dentro-da-tela, procede-se da maneira inversa, como vemos na outra ilustração (Fig. 3.3). Nesse caso, a projeção correspondente a cada olho fica do mesmo lado que esse olho.

Em todos os métodos disponíveis, para se ver imagens em 3D, as TVs em 3D, sem óculos, inclusive, usam o artifício de direcionar imagens diferentes para cada olho, de maneira semelhante à discutida acima. Usa-se, para isso os métodos mais sofisticados, disponíveis pela tecnologia. No nosso caso usamos algo muito simples: a imagem para o olho esquerdo é desenhada em vermelho e aimagem para o olho direito, em azul. Assim, quando usarmos óculos azul-vermelho, cada olho verá uma imagem. A missão final fica para o cérebro, que inventará um objeto, numa posição que corresponda às duas imagens vistas pelos dois olhos! O meu

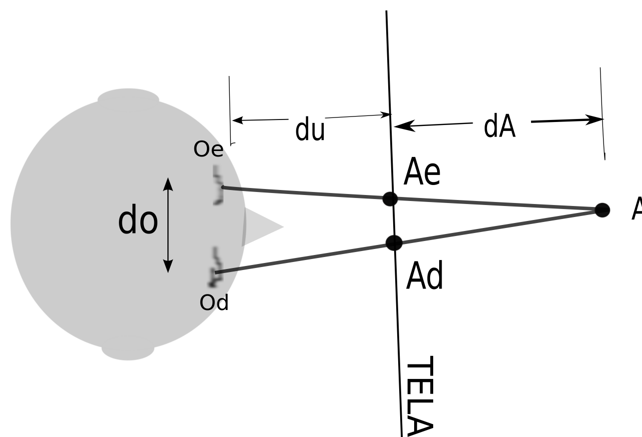

Figura 3.3: Como cada olho de um usuário "projeta" um objeto "interno", A, na tela. Ae é a projeção causada pelo olho esquerdo e Ad a causada pelo olho direito. Note que a projeção referente ao olho esquerdo fica à esquerda do usuário, e a projeção causada pelo olho direito fica à sua direita.

trabalho, para escrever o programa que possibilita facilitar esse desenho, consistiu em trabalhar com a trigonometria envolvida pelos triângulos Ae, A, Ad, Ae e Od, A, Oe, Od. Note que, se a pessoa estiver olhando o objeto de lado, i.e., se o objeto não estiver no centro da tela, os cálculos ficam mais complicados. Mas o leitor não tem que se preocupar com isso, pois já calculei tudo. Uma olhada no código, mostrará como fiz.

Vamos, agora, fazer um exercício simples, que nos preparará para ver melhor figuras em 3D. Coloque um dedo a uns 30 cm de seus olhos, estando à frente de uma parede. Fixe o olhar num ponto da parede e, depois, feche um olho de cada vez. Note como a imagem do dedo se desloca, na parede. Isso é análogo à ilustração acima, para o caso de "objeto-fora" da tela. Para fazer o teste com objeto dentro, você precisa usar, ao invés da parede, um vidro transparente (observando o objeto através de uma janela, por exemplo.) Tente!

Para se causar o efeito tridimensional em objetos extensos, ou seja, não pontuais, trata-se cada ponto do objeto da forma acima. Para segmentos de reta, basta se projetar os pontos extremos dos segmentos e se unir essas projeções por segmentos de reta. Considere, por exemplo, um vetor com origem na tela e que sai da mesma. Assim as projeções, na tela, da origem do vetor, são as mesmas para os dois olhos. A extremidade do vetor, que sai da tela, tem projeções diferentes, como vistas pelos olhos esquerdo e direito. Não temos que nos preocupar com pontos intermediários: para ter a projeção do vetor todo, basta unir as projeçoes

de suas extremidades. Assim, obtemos dois vetores, não coincidentes. Cada um, com o auxílio de óculos, será visto por um olho, dando a visão tridimensional. O efeito da figura impressa não será tão bom como o efeito da imagem na tela do computador, devido aos diferentes métodos para se produzir cores, na tela e na impressão.

Como trataremos figuras mais complicadas, tais como curvas parametrizadas? Para desenhar uma curva tridimensional, temos que fornecer a coordenada (x, y , z) de cada um de seus pontos. Em computação, não podemos fornecer um número infinito de pontos, ao computador. Assim, a curva é "discretizada", de forma que só um número significativo de pontos é fornecido. O "significativo" acima, tem muitas interpretações e depende do problema envolvido. Vamos evitar discussões detalhadas, sobre isso. Em Física, curvas estão associadas, em geral, com a posição de uma partícula, no tempo. Assim, para cada instante, a posição (x,y,z) da partícula pode variar, de acordo com um parâmetro diretamente proporcional ao tempo, que pode ser o próprio tempo. Assim, a curva obtida desenhando-se seus pontos, ordenados segundo o parâmetro temporal, descreve o que chamamos de trajetória da partícula. Mesmo que a curva intercepte a si mesma, não há problema para a compreensão do problema, já que essas intersecções ocorrem para diferentes valores do parâmetro temporal.

Matematicamente falando, as curvas parecem ser estáticas e não com um sequência de evolução, como nas trajetórias de partículas, em Física. Entretanto, quando calculamos derivadas, temos que atribuir um sentido de "evolução" da curva, obtido através de um parâmetro qualquer, s, análogo ao parâmetro temporal, em Física.

Então, para desenhar uma curva, em 3D, precisamos de um número finito de pontos, que a representam, e um parâmetro que organiza os pontos, para sabermos em que sequência uní-los para obter uma aproximação (é sempre uma aproximação) da curva. Aqui está a solução de como desenhar curvas: usamos segmentos de retas unindo pontos "consecutivos" e os desenhamos, em 3D, como discutido acima! As figuras 3.4 e 3.5 exemplificam isso. Na primeira, escolhi segmentos de reta grandes (usei menos pontos) e na segunda usei mais pontos, sendo que os segmentos de reta são quase imperceptíveis. O fato de aparecerem vários segmentos superpostos, é porque a curva é redesenhada várias vezes, com segmentos começando em diferentes posições, nas diversas "passadas". Na versão suave, isso não é percebido. a qualidade das imagens foi prejudicada na edição para a versão impressa. Você verá imagens muito melhores, na tela do computador. Isso vale para a maioria das imagens desse texto.

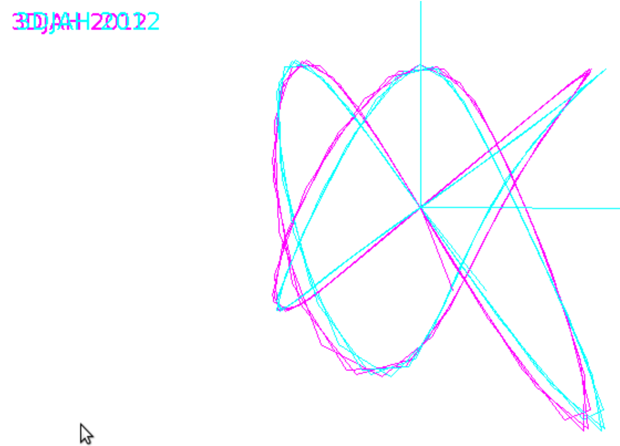

Figura 3.4: Curva em 3D desenhada com segmentos de reta. Vários segmentos se superpõe, pois a curva foi retraçada várias vezes, mas não com os mesmos segmentos.

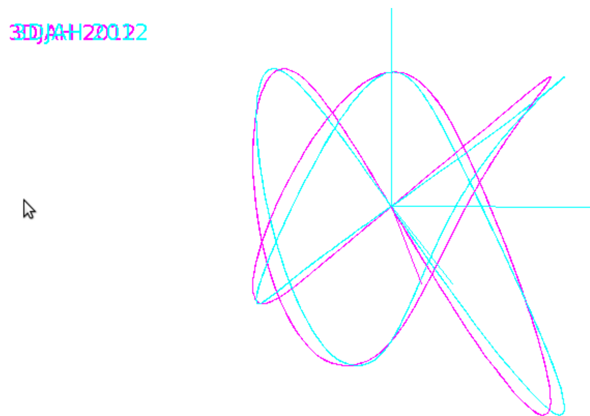

Figura 3.5: Curva em 3D desenhada com segmentos de reta. Aqui, também, a curva foi retraçada várias vezes, não com os mesmos segmentos. Mas nesse caso, os segmentos são menos visíveis.

3.1 Girando curvas inteiras

Às vezes uma curva, mesmo em 3D, pode ser difícil de ser visualizada. Nesse caso, o usuário pode ter a opção de girá-la. Essa opção seria obtida através do controle via

mouse ou outro qualquer. Não usaremos esses controles, aqui, por serem do Java básico e não terem nada com a animação 3D. Aqui, para exemplificar, giraremos a curva, assim que terminarmos de desenhá-la.

O método que usaremos, para girar uma curva será o método trabalhoso de se girar cada um dos pequenos segmentos que a constituem. Isso será feito na seção 5.8, onde mostramos como desenhar e girar as figuras de Lissajous mostradas acima.

Capítulo 4

A Biblioteca $3_{DJA}H$

O Java pode ser expandido, graças à definição de novos objetos, descritos com o uso de classes. No presente caso, definimos o objeto VETOR, que é determinado a partir de três números inteiros, que são suas componentes cartesianas. Depois de definidos vetores, podemos fazer operações entre vetores e entre vetores e escalares. Além disso, podemos obter um vetor unitário, correspondente a um dado vetor ou obter o vetor girado, de um certo ângulo, em torno de um dos eixos cartesianos. Quando a operação não envolve outros vetores, eu a considerei como uma propriedade do dado vetor. Por exemplo, o giro e o vetor unitário são propriedades de um dado vetor. Isso quer dizer que essas funções (métodos) são definidos dentro da classe VETOR.

Quando outros vetores ou escalares estão envolvidos, eu defini uma classe OPV (Operações Vetoriais). Nessa classe se enquadram o produto de um vetor por um escalar, o produto escalar, o produto vetorial, além da capacidade de desenhar vetores.

Então, vamos definir brevemente o uso das Classes VETOR e OPV. A listagem do código dessas classes estão no apêndice B.

4.1 Classe vetor

Definindo Vetores

1. vetor A = new vetor(): cria um vetor A, com 4 componentes nulas.

2. vetor A = new vetor(10,20,30): cria um vetor A, com as três componentes dadas.

3. vetor A = new vetor(10,20,31,23): cria um vetor A, com as quatro componentes dadas. Talvez você precise das quatro, quem sabe?
4. A.esfericas(r, teta, fi): atribui essas coordenadas esféricas a um vetor A, já existente. A deve ser criado, antes, usando-se o ítem (1), acima.
5. A.cilindricas(rho, fi, z): análogo ao de cima, para coordenadas cilíndricas.

Componentes dos vetores

1. A.x(): a coordenada cartesiana, x, do vetor A
2. A.y(): a coordenada cartesiana, y, do vetor A
3. A.z(): a coordenada cartesiana (ou cilíndrica), z, do vetor A
4. A.r(): a coordenada esférica, r, de A.
5. A.teta(): a coordenada esférica θ de A.
6. A.fi(): a coordenada esférica (ou cilíndrica), φ, de A.
7. A.rho(): a coordenada cilíndrica ρ de A.
8. A.fi(): coordenada cilíndrica (ou esférica) φ de A.
9. A.z() : a coordenada cilíndrica (ou cartesiana) z, de A.

Outras propriedades dos vetores

1. A.modulo(): o módulo do vetor A.
2. A.unitario(): um vetor unitário paralelo ao vetor A.

Rotação de um vetor, em torno dos eixos cartesianos

1. A.girax(a): novo vetor A, girado de um angulo a, em torno do eixo x.
2. A.giray(a): novo vetor A, girado de um angulo a, em torno do eixo y.
3. A.giraz(a): novo vetor A, girado de um angulo a, em torno do eixo z.

Inversão de componentes de um vetor

1. A.refleteX(): troca o sinal da componente x do vetor A.
2. A.refleteY(): troca o sinal da componente y do vetor A.
3. A.refleteZ(): troca o sinal da componente z do vetor A.

4.2 Classe opv

Operações matemáticas com vetores

1. soma(A,B): A soma dos vetores A e B.
2. soma(A,B,C): a soma dos 3 vetôres dados.
3. soma(A,B,C,D): a soma dos quatro vetores.
4. diferenca(A,B): vetor B-A.
5. produtoEscalar(A,B): o produto escalar dos dois vetores.
6. produtoVetorial(A,B): o produto vetorial dos dois vetores, na ordem dada.
7. escalarXvetor(a, A): produto do escalar a pelo vetor A.
8. vetorXescalar(A,a): produto do escalar a pelo vetor A.
9. angulo(A,B): o ângulo, em radianos, entre A e B.

Desenhando vetores

1. desenhaVetor(g, V, ORIGEM, Cor): desenha o vetor V, no gráfico g, a partir de um vetor ORIGEM, com a cor dada. Para desenhar em perspectiva, mas não para óculos.
2. desenhaVetor(g, V, ORIGEM): desenha o vetor V, no gráfico g, a partir de um vetor ORIGEM, em duas cores, para óculos 3D.
3. apagaVetor(g, V, ORIGEM): "apaga" o vetor V, desenhado a partir do vetor ORIGEM. Na realidade, o vetor tem suas duas projeções redesenhadas em branco, que é a cor de fundo que deve ser usada. Senão, o efeito não funciona.

Desenhando ovais

1. desenhaOval(g, V, Dh, Dv, Cor): desenha uma oval de eixos horizontal Dh e vertical Dv, na ponta do vetor V, origem absoluta, no gráfico g, com a cor dada.
2. desenhaOval(g, V, Dh, Dv): desenha uma oval de eixos horizontal Dh e vertical Dv, na ponta do vetor V, origem absoluta, no gráfico g, para óculos 3D.
3. apagaOval(g, V, Dh, Dv): "apaga" a oval descrita. Veja apagaVetor.

Desenhando texto

Se o texto consistir de duas letras ou mais, ele não será exatamente paralelo à tela. Acredite: é mais interessante, assim.

1. escreveTexto(g, V, texto, tamanho): escreve o dado texto, na ponta do vetor V, origem absoluta, no gráfico g, com o tamanho dado, para óculos 3D.

2. apagaTexto(g, V, texto, tamanho): desenha o texto dado em cor branca, o que dá a impressão de se apagar, se o fundo for branco.

Sobre a "Origem Absoluta"

A origem absoluta é o canto esquerdo inferior da tela. Um vetor 3D é desenhado a partir de um vetor origem, que sai dessa origem absoluta. Assim, a posição absoluta da ponta do seu vetor é apenas a soma desse vetor com o vetor origem. Isso ficará claro, nos exemplos.

Capítulo 5

Animações

5.1 Introdução

Quando uma figura parece se transformar e/ou se mover, dizemos que a mesma está "animada". O processo de se criar a ilusão de animação pode ser feito de duas formas:

1. Desenhando e apagando

Um jeito é se desenhar o objeto em uma posição, apagá-lo e desenhá-lo, de novo na nova posição. Se isso for feito de maneira suficientemente rápida, o usuário tem a ilusão que o objeto desapareceu de uma posição e reapareceu na outra. Isso dá a ilusão de movimento do objeto. No entanto, quando o desenho do objeto é muito demorado, o usuário percebe o apagar e o redesenhar do mesmo, destruindo a noção de movimento.

2. Desenhando "escondido"

Para evitar problemas, na animação de objetos complexos, procedemos da seguinte forma: Enquanto o usuário observa, na tela, a versão anterior do objeto, a nova versão é desenhada *escondida* dele. Quando a versão escondida estiver pronta, ela será mostrada, instantaneamente, no lugar da versão anterior. Esse efeito é muito mais dramático do que quando se usa o método de "apagar e desenhar".

Vetor: o elemento básico de $3_{DJA}H$

Os métodos gráficos de $3_{DJA}H$ se baseiam no conceito de vetores. Então. as animações possíveis são aquelas que decorrem de operações vetoriais tais como

rotações, esticamentos e compressões dos vetores, assim como a inversão de sentido. Uma curva, em $3_{DJA}H$, são deslocamentos da extremidade de um vetor que sofre todas as possíveis operações acima. Por exemplo, para desenhar uma circunferência, desenhamos a curva descrita pela extremidade de um vetor que gira em torno do eixo x, de 360 graus. Aqui, alguém poderia perguntar: porque não usar o desenho de ovais e circunferências, do JAVA, para isso? A resposta é simples: porque, com esse método, podemos desenhar circunferências (e outras curvas) dentro e fora da tela, ou com uma parte dentro e a outra parte fora da tela. Tudo isso irá depender da curva descrita pela extremidade do vetor, que pode entrar e sair da tela à vontade (do programador.)

Escala de tempo das animações

Computadores mais rápidos podem apresentar uma animação muito mais rapidamente do que imaginava o programador, assim destruindo algum efeito esperado. Para garantir que uma animação se comporte da mesma forma em computadores lentos e rápidos, devemos incluir pausas, em milisegundos, entre a exibição de uma imagem e outra. Para isso, criei uma função "pausa", para o $3_{DJA}H$.

5.2 Inputs básicos de $3_{DJA}H$

Para o $3_{DJA}H$ funcionar, você precisa carregar as classes vetor.class, opv.class e outros.class (para a função pausa, apenas.) Isso é feito no cabeçalho da sua classe, da seguinte forma:

```
import bonelli.vetor;
import bonelli.opv;
import bonelli.outros;
```

Ou, simplesmente,

```
import bonelli.*;
```

Para isso funcionar, no diretório em que você tem seu código Java, deve estar um diretório (pasta) *bonelli*, dentro do qual você colocou as classes vetor.class, opv.class, e outros.class, que acompanham esse texto. Melhor copiar a pasta toda, a partir do CD.

5.3 Exemplos simples de animação

Um dos exemplos mais simples de animação, que podemos fazer com $3_{DJA}H$, é o da rotação de um vetor em torno de um dos eixos coordenados. Para outros

programas, seria a animação de um ponto. Acontece que, aqui, o vetor é o bloco fundamental das construções. Faremos esse exemplo em vários passos, para facilitar o entendimento. Primeiro definimos dois vetores A e B. Depois, desenhamos o vetor B, na ponta do vetor A, sendo que A fica "amarrado" na origem absoluta. Escolhi A apontando para fora da tela ($A_x > 0$), de forma que a origem de B estará *fora da tela*, para nossa diversão. Para praticarmos animação, giramos B de 30 graus e o desenhamos, de novo, na ponta de A, sem apagar o desenho anterior. Depois, continuamos girando B, sempre de 30 graus, e sempre desenhando.o na ponta do vetor A. Aqui, teremos várias coisas a serem melhoradas, mais tarde:

1. Não pintamos o fundo de branco, de forma que o efeito 3D, apesar de presente, pode ser melhorado.

2. Para girar e desenhar B outras vezes, repetimos as linhas do programa, o que não é elegante.

3. A animação é tão rápida, que o usuário não percebe os 'novos B' sendo desenhados: aparecem todos os desenhos simultaneamente.

A figura 5.3 mostra esse resultado.

O código usado foi o seguinte:

```
 // Gira um vetor em torno do eixo y.
import bonelli.opv;
import bonelli.vetor;
import java.awt.*;
import java.applet.*;

public class vetorGirante extends Applet
   {
   double piSobre180 = Math.PI/180.;
   vetor B = new vetor(0., 150., 100.);
   vetor A = new vetor( 150., 150., 230.);

   public void init(){
     resize(400,400);
   }

   public void paint(Graphics g){
   opv.desenhaVetor(g, B, A);

   //Gira B de 30 graus
   B = B.giray(30.* piSobre180);
```

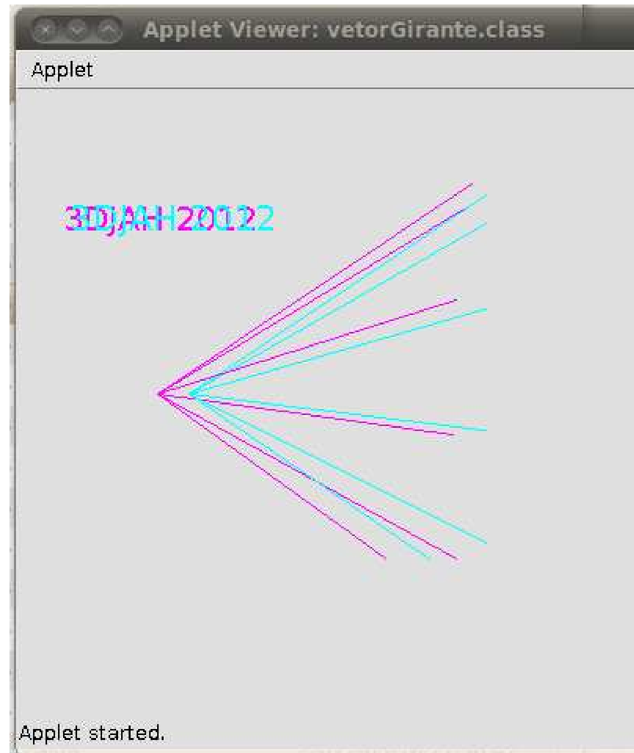

Figura 5.1: Vetor girando de saltos de 30 graus, em torno do eixo y. Tudo fora da tela.

```
//Desenha de novo, sem apagar o anterior
opv.desenhaVetor(g, B, A);
//Continua girando e desenhando várias vezes...
B = B.giray(30.* piSobre180);
opv.desenhaVetor(g, B, A);
B = B.giray(30.* piSobre180);
opv.desenhaVetor(g, B, A);
B = B.giray(30.* piSobre180);
opv.desenhaVetor(g, B, A);
B = B.giray(30.* piSobre180);
opv.desenhaVetor(g, B, A);
```

```
   } // do paint
} //da classe
```

Esse código deve ser compilado, através do comando

```
javac vetorGirante.java
```

e, para ser visualizado, deve se usar um arquivo HTML, assim:

```
<html>
<applet code="vetorGirante.class" width=400 height=400>
</applet>
</html>
```

sendo que esse arquivo em HTML pode ser aberto no navegador, ou com o appletviewer , distribuido com o Java.

Como exercício, voce pode girar o vetor em torno dos eixos x e/ou y e/ou z, para obter efeitos surpreendentes (e bonitos!)

Exercício 4. *Modifique o código para que B gire de 5 graus, em torno do eixo y, e em torno do eixo x. Você talvez tenha que mexer com o tamanho da área do desenho. A figura resultante é mostrada na figura 5.3a.*

Exercício 5. *Modifique o código vetorGirante para que B gire de 5 graus, em torno de cada um dos eixos x, y, e z. Você talvez tenha que mexer com o tamanho da área do desenho. A figura resultante é mostrada na figura 5.3b.*

Agora, vamos fazer as correções mencionadas nos três ítens, acima, da seguinte forma:

1. Pintaremos o fundo de branco, i.e., preencheremos um retângulo de cor branca.
2. Usaremos um laço (loop) para girar e redesenhar o vetor B várias vezes.
3. Vamos esperar alguns milisegundos entre cada dois desenhos, permitindo ver o movimento.
4. Apagaremos a posição anterior de B, aumentando a sensação de movimento.
5. Já que vamos usar um laço, podemos reduzir o ângulo de rotação.
6. Para deixar mais interessante, vamos girar o vetor origem, também.

Como não temos como mostrar a animação, aqui, vamos mostrar a figura (5.3) resultante quando NÃO se apaga o desenho anterior, cujo código mostramos a seguir. A animação estará disponível no CD. O código, para a versão que não apaga os desenhos anteriores, é o seguinte.

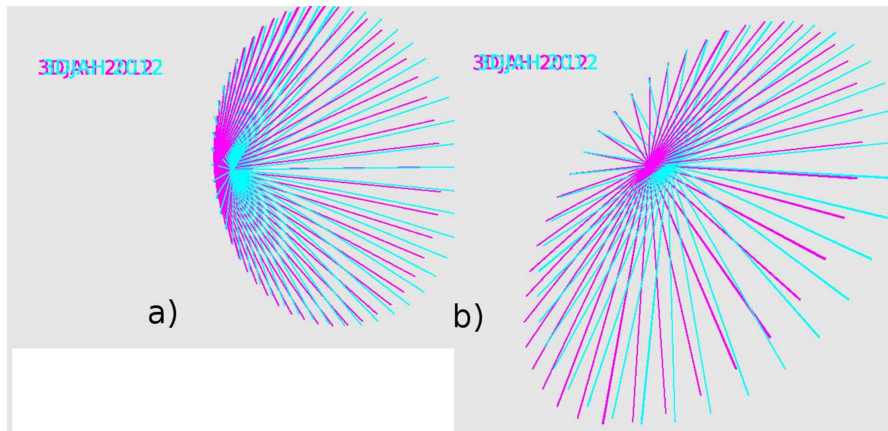

Figura 5.2: Vetor girando de saltos de 5 graus, em torno dos eixos x e y (a) e em torno dos três eixos (b). Tudo fora da tela.

```
 // Gira um vetor em torno dos eixos x,y,z.
import bonelli.opv;
import bonelli.vetor;
import bonelli.outros;
import java.awt.*;
import java.applet.*;

public class vetorGirante4 extends Applet
   {
   double piSobre180 = Math.PI/180.;
   vetor B = new vetor(0., 100., 100.);
   vetor A = new vetor( 150., 250., 250.);
   double angulo = 5.*piSobre180;
   double voltas = 360./5.;
   int i;  //contador
   public void init(){
     resize(1200,700);
   }

   public void paint(Graphics g){
   g.setColor(Color.white); //Atribui cor branca
   g.fillRect(0,0,1200,700);
```

```
   opv.desenhaVetor(g, B, A);

   //Gira B de 5 graus cada vez até completar 360 graus
   //Para ficar interessante, vamos girar A também.
   for(i=0;i<voltas;i++)
   {

      A = A.girax(angulo/10.);
      B = B.giray(2.*angulo);
      //Desenha de novo, sem apagar o anterior
      opv.desenhaVetor(g, B, A);
      outros.pausa(100);
      //Continua girando e desenhando várias vezes...
   }  // do for
   } // do paint
} //da classe
```

O código que apaga o vetor anterior, antes de desenhar o próximo, contém o comando "apagaVetor". Para apagar um vetor, no futuro, precisamos guardar sua posição e suas componentes anteriores. No código, usei "Aanterior" e "Banterior" para guardar esses valores aumentei o tempo de pausa, e diminuí os incrementos angulares, para o movimento ficar mais suave. A parte gráfica,dessa animação, está no CD e o código é o seguinte:

```
// Gira um vetor em torno dos eixos x,y,z
// e apaga de\-senho anterior
//Gira, também, o vetor origem.
//Não apagamos, pois nao eh desenhado
import bonelli.opv;
import bonelli.vetor;
import bonelli.outros;
import java.awt.*;
import java.applet.*;

public class vetorGirante4Apaga extends Applet
   {
   double piSobre180 = Math.PI/180.;
   vetor B = new vetor(0., 100., 100.);
   vetor A = new vetor( 150., 250., 250.);
   vetor Aanterior = A;
   vetor Banterior = B;
   double angulo = 5.*piSobre180;
```

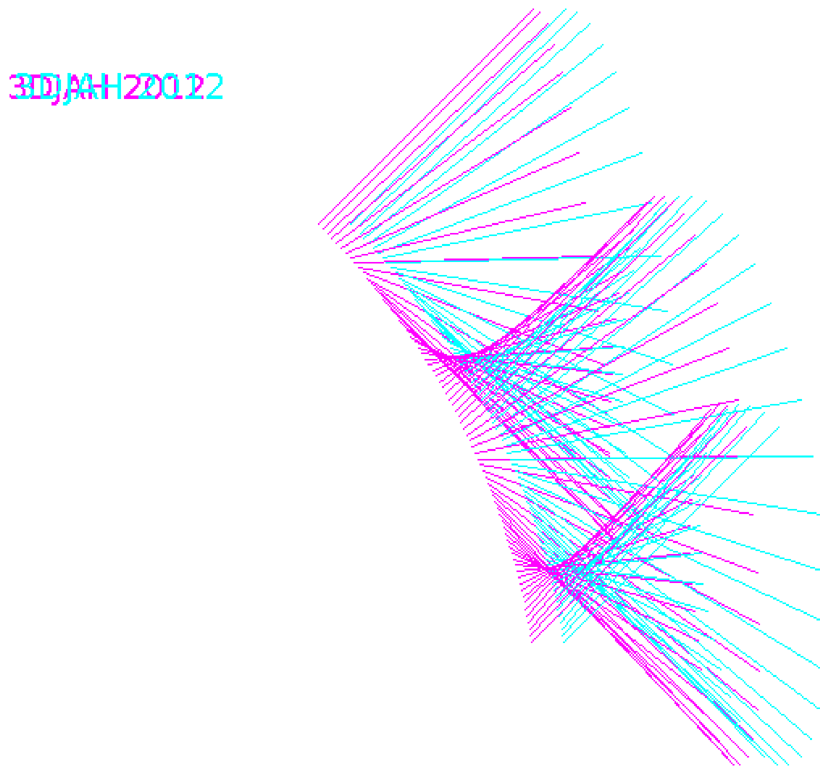

Figura 5.3: Vetor girando em torno do eixos x, y, e z, enquanto o vetor origem também gira. Tudo fora da tela.

```
double voltas = 360./5.;
int i;  //contador
public void init(){
  resize(1200,700);
}

public void paint(Graphics g){
g.setColor(Color.white); //Atribui cor branca
g.fillRect(0,0,1200,700);
```

```
   opv.desenhaVetor(g, B, A);
   outros.pausa(400);

   //Gira B de 5 graus cada vez
   //até completar 360 graus
   //Para ficar interessante,
   //vamos girar A também, mas pouco.
   for(i=0;i<voltas;i++)
   {

      A = A.girax(angulo/10.);
      B = B.giray(angulo);
      //Desenha de novo, sem apagar o anterior
      opv.apagaVetor(g, Banterior, Aanterior);
      opv.desenhaVetor(g, B, A);
      outros.pausa(200);
      Aanterior = A; //A se torna o anterior
      Banterior = B; //B se torna o anterior
      //Continua girando e desenhando...
   }  // do for
   } // do paint
} //da classe
```

5.4 Curvas e trajetórias

Em matemática, queremos visualizar curvas complicadas (não planas) em 3D. O mesmo ocorre na Física, onde queremos seguir a trajetória de uma partícula. Como um corpo sólido é constituido de partículas, podemos representar seu movimento pelo movimento de um número limitado de partículas, que representem sua translação e rotação. Para representar fluidos, a situação é mais complexa, e o número representativo de partículas, necessário, é muito grande.

Para aproveitar o trabalho já feito, acima, vamos desenhar a curva que representa o movimento da extremidade do vetor girante. Para isso, desenharemos apenas o deslocamento dessa extremidade, que também é um vetor, para cada iteração. Tanto para o matemático, quanto para o físico, essa é uma curva parametrizada. Para o físico, o parâmetro é o tempo, no caso, o instante de cada iteração, ou seja, de cada desenho.

5.4.1 O vetor deslocamento

Para um vetor origem A, fixo, o deslocamento da extremidade do vetor B é dado simplesmente por

$$\Delta \boldsymbol{B} = \boldsymbol{B}_2 - \boldsymbol{B}_1$$

onde os índices (1) e (2) indicam as posições anterior e posterior de B. Em $3_{DJA}H$, isso é codificado da seguinte forma:

```
deltaB = opv.diferenca(B2, B1);
//calculando a subtração de vetores
```

Infelizmente, esse é um caso muito particular: se o vetor origem girar, ou mudar de tamanho (sua origem é fixa, na tela), esse cálculo não vale mais. Em todos os casos, o deslocamento que importa é o da extremidade do vetor resultante da soma da origem com B. Então, se tanto o vetor origem, quanto nosso vetor mudarem, o cálculo do deslocamento é feito assim:

```
posicao1 = opv.soma(A1,B1);
posicao2 = opv.soma(A2, B2);
deslocamento = opv.diferenca(posicao2, posicao1);
```

A curva final constituirá do desenho de todos deslocamentos, sempre a partir de um vetor posição anterior, de acordo com a regra:

```
nova posição = posição anterior + deslocamento
```

Para desenhar o deslocamento, a partir de uma dada posição, usamos:

```
opv.desenhaVetor(g, deslocamento, posicao1);
//desenhamos o deslocamento a partir da
//posicao ABSOLUTA anterior.
```

5.4.2 Exemplo do traçado de curvas em 3D

Vamos nos ater ao exemplo do vetor girante, com origem também girante, para traçar a curva descrita pela ponta do vetor. No mesmo código, eliminaremos o desenho dos vetores e incluiremos o desenho dos deslocamentos. Eu marquei as novas linhas com a palavra "Nova". Note que esse é o procedimento geral para se desenhar curvas, com esse programa: um vetor, a partir da origem absoluta, indica a posição de pontos sobre a curva. O desenho da curva é aproximado por segmentos, que são as diferenças veotirais entre dois vetores posição consecutivos. No presente exemplo, a figura ficou meio difícil para os principiantes verem, pois sai uns 10 centímetro da tela (Quem vê em 3D, pode fazer a curva interceptar uma régua (real) colocada a partir da tela.)

O resultado é mostrado na figura 5.4. O código, para o traçado da curva descrita pela ponta do vetor que se move é o seguinte :

```
 // Gira um vetor em torno dos eixos x,y,z
//Gira, também, o vetor origem.
// Desenhamos apenas o deslocamento da extremidade
// do vetor girante.
import bonelli.opv;
import bonelli.vetor;
import bonelli.outros;
import java.awt.*;
import java.applet.*;

public class curva3D extends Applet
   {
   double piSobre180 = Math.PI/180.;
   vetor B = new vetor(0., 100., 100.);
   vetor A = new vetor( 150., 250., 250.);
   vetor Aanterior = A;
   vetor Banterior = B;
   vetor posicao1, posicao2, deslocamento;
   //linha acima é Nova
   double angulo = 5.*piSobre180;
   double voltas = 360./5.;
   int i;  //contador
   public void init(){
     resize(1200,700);
   }

   public void paint(Graphics g){
   g.setColor(Color.white); //Atribui cor branca
   g.fillRect(0,0,1200,700);
   outros.pausa(400);

   //Gira B de 5 graus cada vez até completar 360 graus
   //Para ficar interessante, vamos girar A também,
   //mas pouco.
   for(i=0;i<voltas;i++)
   {
      posicao1 = opv.soma(A, B);
      A = A.girax(angulo/10.);
      B = B.giray(angulo);
      posicao2 = opv.soma(A, B);//Nova
      deslocamento = opv.diferenca(posicao2, posicao1);
```

```
      //linha acima é nova
      opv.desenhaVetor(g, deslocamento, posicao1);
      //linha acima é nova
      outros.pausa(200);
      //Continua girando e desenhando várias vezes...
   }  // do for
   } // do paint
} //da classe
```

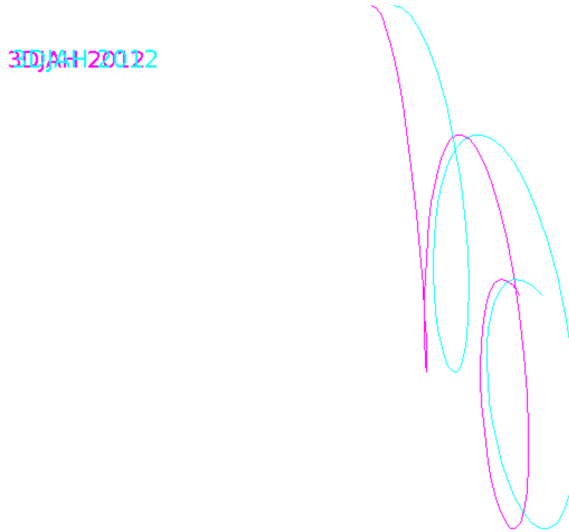

Figura 5.4: Curva, em 3D, fora da tela, obtida a partir dos deslocamentos da extremidade do vetor girante do exemplo anterior.

5.5 As “partículas” da Física

Em Física, quando ilustramos o movimento de uma partícula, desenhamos uma pequena circunferência, já que a partícula teria tamanho nulo (seria um ponto.) Mas essa visualização é útil pois podemos desenhar o movimento de nossa “partícula”, em 3D, de forma que o seu tamanho aumenta, quando se aproxima do observador e diminui, quando se afasta do mesmo. Eu programei isso, no código de OPV, da seguinte forma:

1. A esfera é um caso particular de uma oval
2. A oval é desenhada na ponta de um vetor, a partir da origem absoluta.
3. A cor pode ser escolhida. Não sendo escolhida uma cor, o desenho é feito em azul-vermelho, para óculos.
4. O tamanho da oval depende dos eixos especificados e da distância da mesma ao observador.

Para exemplificar o desenho de ovais em movimento, vamos fazer a animação de um projétil, i.e., um objeto lançado no campo gravitacional (constante) perto do solo. Não calcularemos a curva, que representa a trajetória, antes de desenhar. Na realidade construiremos a curva, vetorialmente, a partir do cálculo dos incrementos de velocidade e posição. Isso é útil pois permite se trabalhar até em casos em que a força da gravidade (ou qualquer outra força) não seja constante.

A nova posição da partícula, depois de um instante dt, é dada por

$$\boldsymbol{r}(t+dt) = \boldsymbol{r}(t) + \boldsymbol{v}(t+dt)\ dt$$

A variação da velocidade vetorial $\boldsymbol{dv}$ é dada pela ação da aceleração da gravidade $\boldsymbol{g}$, também um vetor, durante um pequeno intervalo de tempo dt, um escalar, da seguinte forma:

$$\boldsymbol{dv} = \boldsymbol{g}\ dt$$

A nova velocidade, depois desse intervalo de tempo, é

$$\boldsymbol{v}(t+dt) = \boldsymbol{v}(t) + \boldsymbol{dv}$$

E assim por diante. O que nós estamos fazendo é uma integração numérica, para a qual existem teorias detalhadas. Mas vamos integrar de uma forma bem simples, já que não temos espaço, aqui, para muita precisão. Os passos acima são repetidos um grande número de vezes, até que o instante final, ou alguma outra condição (o objeto atingir o solo, por exemplo) seja alcançada. Aqui vai o código, do movimento do projétil, sem apagar as posições anteriores. Para convertê-lo em uma animação, na tela, basta retirar o comentário da linha que contém o comando para apagar, i.e.,

```
// o.apagaOval(grafico, rAnterior, 10, 10);
// Remover o comentario, para apagar
```

Uma vez que se apague os desenhos anteriores, deve-se diminuir os incrementos temporais, no início do programa. A velocidade de movimento, na tela, pode ser controlada pelo valor usado na pausa (e, claro, pela escolha das componentes do vetor velocidade, no programa.) Veja o código completo:

```
import bonelli.*;
import java.awt.*;
import java.applet.*;

public class projetilOvalE extends Applet{
   vetor X,Y,Z;  //eixos cartesianos
   vetor ORIGEM;
   //vetor posicao, a partir da origem absoluta.
   bonelli.opv o;
   //passa para "o", todas operacoes vetoriais de opv.
   vetor g, r, deltar;
   //aceleracao da gravidade, posicao, posicao anterior.
   vetor rAnterior;
   vetor v,deltav;
   //velocidade, incremento de velocidade
   double deltat = .2;
   //incremento temporal: diminua, se for apagar cada oval
   public void init(){
      resize(2000,600);
      //dimensao da janela grafica
   }
   public void paint(Graphics grafico){
      //paint permite desenhar
      g = new vetor(0., 0.,-9.8,0.);
      //gravidade
      r = new vetor(0.,0.,0.,0.);
      //posicao inicial
      rAnterior = new vetor();
      //para guardar a posicao anterior
      v = new vetor(35.,35.,60.,0.);
      //velocidade inicial
      X = new vetor(450.,0.,0.,0.);
      //os eixos cartesianos
      Y = new vetor(0.,600.,0.,0.);
      Z = new vetor(0.,0.,300.,0.);
      ORIGEM = new vetor(-100.,100.,130.,0.);
      grafico.setColor(Color.white);
      //Para criar fundo branco
      grafico.fillRect(0,0,2000,600);
      //caso nao  seja default, no navegador.
      o.desenhaVetor(grafico,X,ORIGEM);
```

```
      //linha acima, desenha os eixos
      o.desenhaVetor(grafico,Y,ORIGEM);
      o.desenhaVetor(grafico,Z,ORIGEM);
      r = o.soma(ORIGEM,r);  // r eh atualizado
      while(r.x()<300){
         //Ateh sair da tela de 200 pixels
         // o.apagaOval(grafico, rAnterior, 10, 10);
         // Remover o comentario, para apagar
         deltav = o.escalarXvetor(deltat,g);
         v = o.soma(v,deltav);
         deltar = o.escalarXvetor(deltat,v);
         o.desenhaVetor(grafico,X,ORIGEM);
         //desenha os eixos
         o.desenhaVetor(grafico,Y,ORIGEM);
         o.desenhaVetor(grafico,Z,ORIGEM);
         o.desenhaOval(grafico, r ,10,10);
         outros.pausa(13);
         rAnterior = r;
         r = o.soma(rAnterior,deltar);
         //soma delta r aa posicao anterior.
      }
   }
}
```

A figura 5.5 mostra o projétil, quando não apagamos as imagens anteriores. A curva descrita é uma parábola, vista em perspectiva.

Exercício 6. *Desenhe só a trajetória do projétil, usando segmentos de reta (vetores) para denotar os deslocamentos. Não desenhe as esferas (ovais), de forma que obterá o desenho de uma parábola.*

Exercício 7. *Na animação do projétil, diminua as componentes x e y, da velocidade, de forma que o projétil atinge o solo antes da simulação terminar, e sobe outra vez. Desenhe o projétil maior, para representar uma bola de borracha.*

Exercício 8. *No exercício anterior, deixe a bola de borracha cair verticalmente, com velocidade inicial nula. Simule a colisão com o solo deformando a bola, usando várias ovais. Depois faça-a subir novamente. A velocidade de subida, logo após a colisão, é o vetor inverso da velocidade de descida, no momento da colisão.*

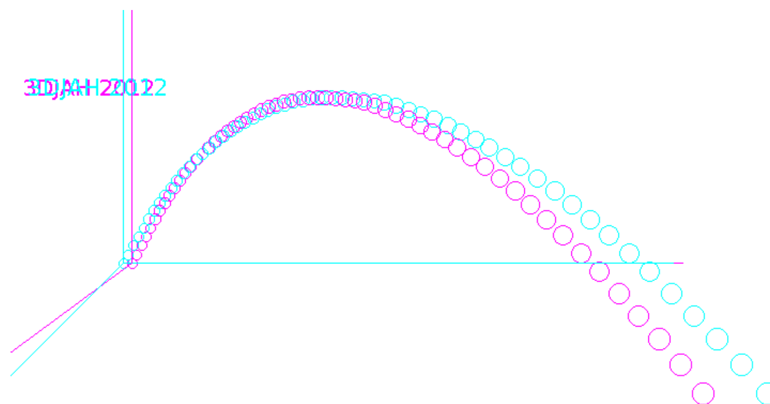

Figura 5.5: Projétil sob ação de gravidade. O desenho continua até que o projétil saia uma certa distância da tela/página.

5.6 Desenhando o nome dos vetores

Quando aparecem vários vetores, numa figura ou animação, é possível nomeá-los, usando-se texto bastando, em geral, uma letra. É o caso dos eixos cartesianos, x, y, e z, vetores indicando a posição de um ponto e vetores entre dois pontos genéricos. A técnica que criei consiste em se desenhar o texto na ponta do vetor. Quando esse vetor se move, o tamanho do texto pode variar, caso a extremidade do vetor se aproxime ou se afaste do usuário. A orientação do texto, entretanto, não gira: é sempre aproximadamente paralela à tela, podendo entrar ou sair da tela, dependendo da orientação do vetor. Isso só acontece para textos com duas ou mais letras. Nessa versão, do $3_{D_JA}H$, não é permitido ao usuário mudar a fonte que escolhi.

Para exemplificar o uso de texto na ponta de vetores, vamos desenhar um sistema de eixos cartesianos, e um vetor R, qualquer. Aqui os movimentos serão os seguintes: os eixos cartesianos(móveis) giram em torno do eixo z, da tela, inclusive o eixo Z, móvel, o que não o altera, claro. Durante essa rotação, as etiquetas com os nomes dos eixos também giram e mudam de tamanho. Na figura 5.6 só aparece um instante da animação.

Vejamos o código dessa animação:

```
//giraObservador.java: Sistema x,y,z gira
//em torno do eixo z.  Um outro vetor, R,
// gira em torno do eixo x.
```

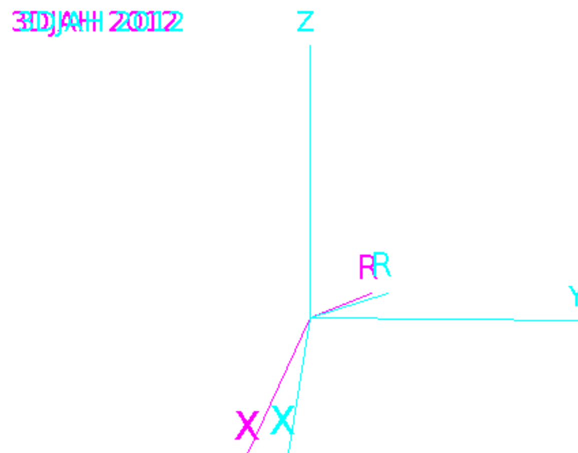

Figura 5.6: Sistema de eixos cartesianos X, Y, Z, girantes em torno do eixo z, da tela e um vetor R, girando em torno do eixo x, da tela. Esse exemplo é para mostrar como ficam as etiquetas 3D, dos vetores.

```
//Para exemplificar de\-senho dos nomes dos vetores.
//
import java.awt.*;
import java.lang.*;
import java.applet.*;
import bonelli.*;

public class giraObservador extends Applet{
   bonelli.opv o;
   vetor X, Y, Z, R, ORIGEM; //
   vetor Girado, Absoluta;

   public void init(){
   resize(2000,700);
   }

   public void paint(Graphics g){
```

```
      X = new vetor(200.,0.,0.,0.);
      Y = new vetor(0.,200.,0.,0.);
      Z = new vetor(0.,0.,200.,0.);
      R = new vetor(100.,50., 50.);
      ORIGEM = new vetor(0.,250.,100.,0.);
      double giro = 0.1; //cada rotação
      double doispi = 2.*Math.PI;
      double angulo = 0.;
      while(angulo < doispi){ // gira ateh 2*pi
         //Cria um fundo branco
         g.setColor(Color.white);
         g.fillRect(0,0,2000,700);
         angulo += giro;

         Girado = X.giraz(angulo);
         Absoluta = o.soma(ORIGEM, Girado);
         o.desenhaVetor(g, Girado, ORIGEM);
         o.escreveTexto(g, Absoluta, "X", 20);

         Girado = Y.giraz(angulo);
         Absoluta = o.soma(ORIGEM, Girado);
         o.desenhaVetor(g, Girado, ORIGEM);
         o.escreveTexto(g, Absoluta, "Y", 20);

         Girado = Z.giraz(angulo);
         Absoluta = o.soma(ORIGEM, Girado);
         o.desenhaVetor(g, Girado, ORIGEM);
         o.escreveTexto(g, Absoluta, "Z", 20);

         Girado = R.girax(angulo);
         Absoluta = o.soma(ORIGEM, Girado);
         o.desenhaVetor(g, Girado, ORIGEM);
         o.escreveTexto(g, Absoluta, "R", 20);


         outros.pausa(200);//pausa de 200 milisegundos
      }
   }
}
```

Note que os vetores “Absoluta” e “Girado” foram usados para X, Y, Z, e R, já que seus valores são redefinidos em cada caso. Note, quando vir a animação do

CD, que não apaguei os vetores ou suas etiquetas, apenas desenhei um novo fundo branco sobre cada desenho anterior. Abaixo, vemos como fazer isso desenhando escondido do usuário.

Exercício 9. *Modifique o código acima para fazer o vetor R girar em torno do eixo y, e deixe inalterada a rotação dos eixos cartesianos.*

5.7 Desenhando escondido do usuário

Em Java, é possível se desenhar uma figura escondida do observador e só mostrá-la quando estiver pronta. Para animações, isso é muito útil, pois o usuário só vê uma figura pronta substituída por outra, também pronta, de forma quase instantânea. No exemplo da seção anterior, eu apago toda a tela (pintando de branco) e depois desenho, novamente, os vetores e letras, em suas novas posições. Como esses desenhos são simples, eles são rápidos, até para meus computadores, que são lentos. A animação é tão rápida, que precisamos usar uma pausa de umas centenas de milisegundos (veja o código, acima.)

Para desenhos mais complicados, entretanto, é melhor não desenhar com o usuário observando. Além disso, a animação anterior fica "piscando" um pouco, durante a exibição. Vamos refazer a mesma animação acima só que vamos desenhar escondido e só mostramos a nova cena quando ela estiver pronta. Veja o código, a seguir. Dessa vez algumas explicações são necessárias. Eu as faço, depois do código.

```
 //
//giraObsEscondido.java: Sistema x,y,z gira
//em torno do eixo z.  Um outro vetor, R,
 gira em torno do eixo x.
//  Para exemplificar de\-senho dos nomes dos vetores.
// Versão anterior = giraObservador.java,
//  Nessa versão:  desenhamos escondido.
import java.awt.*;
import java.lang.*;
import java.applet.*;
import bonelli.*;

public class giraObsEscondido extends Applet{
   bonelli.opv o;
   vetor X, Y, Z, R, ORIGEM; //
   vetor Girado, Absoluta;

   public void init(){
```

```
resize(2000,700);
}

public void paint(Graphics visivel){
   Image imagemEscondida = createImage(2000, 700);
   //Essa nao eh vista pelo usuario, ainda.
   Graphics escondido = imagemEscondida.getGraphics();
   //O de\-senho em "escondido" eh escondido.
   X = new vetor(200.,0.,0.,0.);
   Y = new vetor(0.,200.,0.,0.);
   Z = new vetor(0.,0.,200.,0.);
   R = new vetor(100.,50., 50.);
   ORIGEM = new vetor(0.,250.,100.,0.);
   double giro = 0.1; //cada rotação
   double doispi = 2.*Math.PI;
   double angulo = 0.;
   while(angulo < doispi){ // gira ateh 2*pi
      //Cria um fundo branco
      escondido.setColor(Color.white);
      escondido.fillRect(0,0,2000,700);
      angulo += giro;

      Girado = X.giraz(angulo);
      Absoluta = o.soma(ORIGEM, Girado);
      o.desenhaVetor(escondido, Girado, ORIGEM);
      o.escreveTexto(escondido, Absoluta, "X", 20);

      Girado = Y.giraz(angulo);
      Absoluta = o.soma(ORIGEM, Girado);
      o.desenhaVetor(escondido,Girado, ORIGEM);
      o.escreveTexto(escondido,Absoluta, "Y", 20);

      Girado = Z.giraz(angulo);
      Absoluta = o.soma(ORIGEM, Girado);
      o.desenhaVetor(escondido,Girado, ORIGEM);
      o.escreveTexto(escondido,Absoluta, "Z", 20);

      Girado = R.girax(angulo);
      Absoluta = o.soma(ORIGEM, Girado);
      o.desenhaVetor(escondido, Girado, ORIGEM);
      o.escreveTexto(escondido, Absoluta, "R", 20);
```

```
            visivel.drawImage(imagemEscondida,0,0,this);
            //Aqui, a figura escondida fica visivel.

            outros.pausa(200);//pausa de 200 milisegundos
        }
    }
}
```

Note que esse processo de se desenhar uma imagem e só mostrá-la, depois de pronta, pertence à biblioteca do Java e não ao meu pacote $3_{DJA}H$. Aqui temos as seguintes classes importantes: "Graphics" e "Image". Assim, criamos um objeto gráfico, "visivel", usando

```
Graphics visivel
```

que denota a criação de "visivel" como um objeto gráfico. No caso do gráfico escondido, fica mais complicado. Primeiro criamos um objeto "Image",

```
Image imagemEscondida = createImage(2000, 700);
```

com "createImage", e depois atribuimos a essa imagem propriedades gráficas, para podermos desenhar nela, e damos o nome de "escondido" a esse

```
Graphics escondido = imagemEscondida.getGraphics();
```

novo ambiente gráfico.

Para terminar, precisamos mostrar o gráfico completo ao usuário, usando

```
visivel.drawImage(imagemEscondida,0,0,this);
```

Uma das coisas mais interessantes do Java, é a designação "this" que é uma forma do programa se referir a ele mesmo. Estranho, não é? Veja nos códigos de "vetor.java" e de "opv.jav", como usei muito isso, para definir componentes de vetores e operações com vetores.

5.8 Girando curvas

Na seção anterior giramos vetores simples. Como girar cuvas inteiras? Basta girar cada um dos segmentos de reta que compõem a curva. É isso que fazemos abaixo.

Figuras de Lissajous são muito conhecidas em duas dimensões, sempre mostradas em filmes de ficção científica, onde se vê figuras interessantes nas telas de osciloscópios e, agora, nas telas de computadores. As figuras de Lissajous em

duas dimensões consistem de curvas parametrizadas (não funções), onde as componentes da posição cartesiana de um ponto são x(t) e y(t), i.e., o parâmetro é t e

$$x(t) = Asen(\omega_x t + \phi_x)$$

e

$$y(t) = Asen(\omega_y t + \phi_y)$$

onde A é uma amplitude, ω_x e ω_y são constantes que acentuam ou atenuam a variação do parâmetro t. As fases, ϕ_x e ϕ_y são as fases da função senoidal. Quando se traça as curvas descritas por (x, y) com a variação do parâmetro, t, obtem-se curvas interessantes. Como estamos tratando de animação em 3D, acrescentamos mais um eixo, o eixo z, e nossa curva será desenhada quando o ponto (x, y, z) se deslocar no espaço, com a variação do parâmetro t.

Escolhi os parâmetros de tal forma que as figuras se fecham sobre si mesmas, i.e., são periódicas em t. Nós as desenhamos escondido do usuário e só as mostramos depois de prontas. Enquanto a nova figura não está pronta, o usuário vê a figura anterior. Mesmo para computadores lentos, como os meus, a animação fica muito boa.

Essa última animação incorpora quase tudo que estudamos, até aqui. Desenhamos a curva através de pequenos segmentos de reta, giramos cada segmento para obter o giro da curva como um todo e desenhamos escondido. E *eu* escrevo '3DJAH-2012' na ponta de um vetor. Vamos ao código?

```
 //
//lissajousRand.java: escolhe ao acaso os inteiros
//e gira a figura, em torno
//do eixo z.
//
import java.awt.*;
import java.lang.*;
import java.applet.*;
import java.awt.image.*;
import bonelli.opv;
import bonelli.vetor;

public class lissajousRandE extends Applet{
   opv o;
   vetor A, B, C, ANTERIOR, NOVO, DESLOCA; //
   vetor NOVOG, ANTERIORG;
   vetor ORIGEM;
   Color cor;

   public void init(){
   resize(2000,700);
```

```
   }
   public void paint(Graphics g){
      //Esse eh o que mostramos (g)
      Image imagemfora = createImage(2000, 700);
      //Essa nao eh vista pelo usuario, ainda.
      Graphics grafico = imagemfora.getGraphics();
      //O de\-senho em "grafico" eh escondido.
      double randx, randy, randz;
      randx = .01 * (int)(10 * Math.random()+1);
      randy = .01 * (int)(10 * Math.random()+1);
      randz = .01 * (int)(10 * Math.random()+1);

      A = new vetor(200.,0.,0.,0.);
      B = new vetor(0.,200.,0.,0.);
      C = new vetor(0.,0.,200.,0.);
      ORIGEM = new vetor(0.,400.,150.,0.);
      //vetor D = new vetor(50.,50.,50.);
      int i = 0, giro = 0;
      double doispi = 2.*Math.PI;
      double angulo = 0.;
    while(giro++<10.*doispi){ //girando os vetores.
      ANTERIOR = new vetor();
      grafico.setColor(Color.white);
      grafico.fillRect(0,0,2000,700);
         angulo = .1*giro;
      for(i=0; i<1000;i++){
         o.desenhaVetor(grafico,A.giraz(angulo),ORIGEM);
         o.desenhaVetor(grafico, B.giraz(angulo), ORIGEM);
         o.desenhaVetor(grafico, C.giraz(angulo), ORIGEM);

         NOVO = new vetor(130.*Math.sin(randx*i),
                130.*Math.sin(randy*i),130.*Math.sin(randz*i));
         NOVOG = NOVO.giraz(angulo);
         ANTERIORG = ANTERIOR.giraz(angulo);
         DESLOCA = o.diferenca(NOVOG, ANTERIORG);
         o.desenhaVetor(grafico,DESLOCA,o.soma(ANTERIORG,ORIGEM));
         ANTERIOR = NOVO;

      }
      g.drawImage(imagemfora,0,0,this);
      //Aqui, a figura escondida eh desenhada em g,
      // que eh visivel.
    }
   }
}
```

Cada vez que o programa for reiniciado, você verá uma nova figura, já que usei números aleatórios (através da função Random), para definir diferentes frequências ω para cada eixo. Como não dá para vermos animações, no livro, o código e a classe executável está disponível no CD que o acompanha.

Capítulo 6

Projetos Futuros

Linhas grossas

O Java, como mostrado nesse livro, tem uma grande limitação: as linhas dos desenhos sempre tem a mesma espessura. Algumas vezes, parecem mais grossas, porque eu editei as figuras. Para resolver isso, o Java sofreu uma grande mudança, com novas classes, disponíveis no Java2D. Além da espessura de linhas, o Java2D permite se traçar curvas Bezier quadráticas e cúbicas, além de operações com figuras geométricas que se superpõem, e muitas outras novidades. Algumas figuras dese livro, tais como as Figuras de Lissajous, poderiam se beneficiar de linhas mais grossas.

Mouse 3D

Para demonstrações em sala aula, com projeções em telas, ou na parede, as figuras 3D se situam em diferentes posições, dependendo da posição do observador. Então, quando se quer mostrar algum detalhe, na figura, não se tem como fazer isso: o mouse fica só no plano da tela. Assim, precisamos de um mouse 3D. Felizmente, instrumentos que servem para resolver isso, estão sendo desenvolvidos. O “Leapmotion”, que ainda vai ser lançado, reconhece a posição dos dedos das mãos ou da ponta de um lápis. Capturando essa informação, é possível se apontar algo no espaço, usando nosso programa 3D. Infelizmente, o “Leapmotion” só está disponível para Windows e Mac, por enquanto.

Imagens que ocultam outras imagens

Outra coisa a ser melhorada, na nossa animação em 3D, é o cruzamento de curvas ou figuras geométricas. No estado atual, cada linha é desenhada, sem se verificar se outra linha estaria na frente da mesma. Para o futuro, pode-se verificar se há algo desenhado em uma dada posição e a que posição do espaço corresponde aquele desenho. Assim, permanecem linhas que estão na frente. Isso acarretará maior uso de memória já que, no estado atual, não guardo informação de onde passam linhas já desenhadas.

Apêndice A

Outros gráficos interessantes

A.1 Eixos cartesianos em 3D

A construção da figura dos sistemas de eixos com origens em diferentes posições, vistos antes, e repetidos aqui (Fig. A.1)

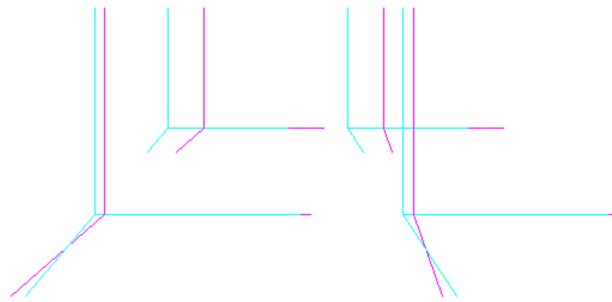

Figura A.1: Quatro sistemas de eixos cartesianos com origens em diferentes pontos do espaço.

são construídos baseados em apenas um conjunto de eixos cartesianos, que são desenhados a partir de uma origem que se desloca no espaço. Veja o código:

```
// eixos3d.java: desenha sistema de eixos em
// várias posições, para mostrar como
// o programa lida com perspectiva.
// (3djah.1207)
```

```
import java.awt.*;
import java.lang.*;
import java.applet.*;
import bonelli.*;
public class eixos3dE extends Applet{
bonelli.opv o;
//Passa propriedades de operacoes
// vetoriais para ''o''
vetor X, Y, Z, ORIGEM;
//Declara esses vetores
public void init(){
  resize(700,500);
}
public void paint(Graphics grafico){
  X = new vetor(200.,0.,0.);
  //So a componente x diferente de zero
  Y = new vetor(0.,200.,0.);
  //So a componente y diferente de zero
  Z = new vetor(0.,0.,200.);
  ORIGEM = new vetor(-100.,100.,100.);
  //A partir do canto esquerdo inferior.
  o.desenhaVetor(grafico,X,ORIGEM);
  o.desenhaVetor(grafico,Y,ORIGEM);
  o.desenhaVetor(grafico,Z,ORIGEM);
  //
  ORIGEM = new vetor(-600.,100.,100.);
  //Nova origem, e assim por diante.
  o.desenhaVetor(grafico,X,ORIGEM);
  o.desenhaVetor(grafico,Y,ORIGEM);
  o.desenhaVetor(grafico,Z,ORIGEM);
  //
  ORIGEM = new vetor(-600.,400.,100.);
  o.desenhaVetor(grafico,X,ORIGEM);
  o.desenhaVetor(grafico,Y,ORIGEM);
  o.desenhaVetor(grafico,Z,ORIGEM);
  //
  ORIGEM = new vetor(-100.,400.,100.);
  o.desenhaVetor(grafico,X,ORIGEM);
  o.desenhaVetor(grafico,Y,ORIGEM);
  o.desenhaVetor(grafico,Z,ORIGEM);
 }
}
```

Exercício 10. *Houve muitas linhas repetidas, nesse código. Reescreva esse pro-*

grama, de forma que os desenhos dos eixos estejam em uma função, que é chamada cada vez que os eixos vão ser desenhados, em uma nova origem.

A.2 Esfera símbolo do $3_{DJA}H$

Eu escolhi a esfera da figura (3.1), quando desenhada para óculos 3D, como sendo o símbolo do $3_{DJA}H$. Nessa animação usamos as coordenadas esféricas, definidas no programa. Fazemos um vetor radial percorrer a esfera, desenhando só o deslocamento de sua extremidade. Você já viu como fazer isso, nos capítulos anteriores. Coloquei uma pausa, depois que cada segmento é desenhado, para que se possa ver o traçado da figura. A imagem resultante é mostrada na figura A.2.

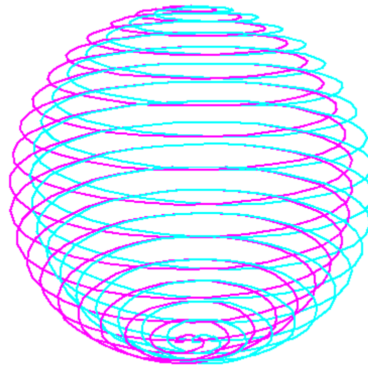

Figura A.2: Esfera, traçada com uma só linha curva, como demonstração do uso de coordenadas esféricas.

O código é o seguinte:

```
  import bonelli.*;
import java.awt.*;
import java.applet.*;
// Demonstracao de vetores definidos
// em coordenadas esfericas.
// Versao 1207
```

```
public class esfericasDemoE2009 extends Applet{
   vetor Novo, Anterior, DeltaR; //
   vetor ORIGEM;
   bonelli.opv opv;
   public void init(){
   resize(700,500);
   }
   public void paint(Graphics grafico){
      ORIGEM = new vetor(100., 300., 150.,0.);
      int i = 1;
      double raio = 100.;
      double fi = 0.;
      double teta = 0.;
      double dteta = .005, dfi = .2;
      Anterior = new vetor();
      //Para que o vetor passe a existir
      Novo = new vetor();
      //sem que tenhamos que escolher suas componentes
      Anterior = Anterior.esfericas(raio, teta, fi, 0.);
      Anterior = opv.soma(Anterior,ORIGEM);
      double fim = Math.PI/dteta;
      grafico.setColor(Color.white);
      grafico.fillRect(0,0,900,500);
      while (i++ < fim){
         teta += dteta;
         fi += dfi;
         Novo = Novo.esfericas(raio, teta, fi, 0.);
         // System.out.println("\n Novoz = " + Novo.z());
         Novo = opv.soma(Novo, ORIGEM);
         DeltaR = opv.diferenca(Novo, Anterior);
         opv.desenhaVetor(grafico, DeltaR, Anterior);
         Anterior = Novo;
         outros.pausa(10);
      }
   }
}
```

Exercício 11. *Remova a pausa e mude a posição da origem, para fazer a esfera se mover. Em vez de apagar toda a esfera anterior, você pode, simplesmente, pintar a tela de branco.*

Apêndice B

Os códigos da biblioteca

B.1 Código da classe vetor

```
 package bonelli;

/**
 * Essa classe permite definir um vetor em cartesianas,
 * esfericas e cilindricas, alem de permitir componentes
 * do vetor, em varios sistemas de coordenadas e
 * varios artificios uteis para desenho em 3D.
 * @version 120320
 * @see opv
 * @author Prof. Enivaldo Bonelli - UFRN
 */

public class vetor{
      double cx, cy, cz, ct;
      /**
       * Define um vetor a partir de suas componentes
       * cartesianas e do tempo
       * Esse vetor tem componentes x,y,z e t.
       * @param x a componente x do vetor.
       * @param y a componente y do vetor, etc.
       */
      public vetor(){    //tentando um construtor default.
         this.cx = 0.;
         this.cy = 0.;
         this.cz = 0.;
      }
```

```
public vetor(double cx,double cy,double cz,double ct){
      this.cx = cx;
      this.cy = cy;
      this.cz = cz;
      this.ct = ct;
   }
   /**
    * Define vetor com tres componentes.
    * @param x componente x, etc.
    */
   public vetor( double cx, double cy, double cz){
      this.cx = cx;
      this.cy = cy;
      this.cz = cz;
      this.ct = -1.0;
   }
   /**
   * Retorna a componente x de um vetor, mesmo
   * que ele tenha sido
   * definido em outras coordenadas.
   */
   public double x(){
      return cx;
   }
   public double y(){
      return cy;
   }
   public double z(){
      return cz;
   }
   public double t(){
      return ct;
   }
   /**
    * Modulo de um vetor
    */
   public double modulo(){
      return Math.sqrt(cx*cx+cy*cy+cz*cz);
   }
   /**
    * Define um vetor unitario,
    * paralelo ao vetor em questao
    */
   public vetor unitario(){
      double modulo = this.modulo();
```

```
         double ux = cx/modulo;
         double uy = cy/modulo;
         double uz = cz/modulo;
         double ut = ct;
         vetor U = new vetor(ux, uy, uz, ut);
         return U;
      }
      /**
       * Define um vetor dadas as coordenadas esfericas
       * @param r a coordenada radial
       * @param teta a coordenada meridional
       * @param fi a coordenada azimutal
       * @param t a quarta coordenada.
       */
    public vetor esfericas(
      double r,double teta,double fi,double t)
      {
         double vx, vy, vz;
         vetor ESF;
         vx = r*Math.sin(teta)*Math.cos(fi);
         vy = r*Math.sin(teta)*Math.sin(fi);
         vz = r*Math.cos(teta);
         ESF = new vetor(vx,vy,vz,t);
         return ESF;
      }
      /**
       * Define um vetor dadas as coordenadas esfericas
       * @param r, a coordenada radial
       * @param teta, a coordenada meridional
       * @param fi, a coordenada azimutal
       */
      public vetor esfericas(double r, double teta, double fi){
         double vx, vy, vz;
         vetor ESF;
         vx = r*Math.sin(teta)*Math.cos(fi);
         vy = r*Math.sin(teta)*Math.sin(fi);
         vz = r*Math.cos(teta);
         this.ct = -1.;
         ESF = new vetor(vx,vy,vz,ct);
         return ESF;
      }
      /**
      * Retorna a coordenada esferica meridional,
      * teta, de um vetor.
      */
```

```
  public double teta(){
     return Math.acos(this.z()/this.modulo());
  }
  /**
  * Retorna a coordenada cilindrica rho.
  */
  public double rho(){
     return this.modulo() * Math.sin(this.teta());
  }
  /**
  * Retorna a coordenada cilindrica/esferica azimutal, fi.
  */
  public double fi(){
     return Math.acos(this.x()/this.rho());
  }
  /**
  * Retorna a coordenada esferica radial.
  */
  public double r(){
     return this.modulo();
  }
  /**
   * Define um vetor dadas as coordenadas cilindricas
   * @param ro, a coordenada radial
   * @param fi, a coordenada azimutal
   * @param z, a coordenada zenital
   * @param t  a quarta coordenada
   */
public vetor cilindricas(
   double ro, double fi, double z, double tt)
{
     double vx, vy;
     vetor CIL;
     vx = ro*Math.cos(fi);
     vy = ro*Math.sin(fi);
     CIL = new vetor(vx,vy,z,tt);
     return CIL;
  }
  /**
   * Define um vetor dadas as coordenadas cilindricas
   * @param ro, a coordenada radial
   * @param fi, a coordenada azimutal
   * @param z, a coordenada zenital
   */
  public vetor cilindricas(double ro, double fi, double z){
```

```
      double vx, vy;
      vetor CIL;
      vx = ro*Math.cos(fi);
      vy = ro*Math.sin(fi);
      CIL = new vetor(vx,vy,z,this.t());
      return CIL;
   }
   /**
   * Retorna um vetor com a coordenada x invertida.
   */
   public vetor refleteX(){
      double vx = -this.x();
      return new vetor(vx, this.y(), this.z(), this.t());
   }
   public vetor refleteY(){
      double vy = -this.y();
      return new vetor(this.x(), vy, this.z(), this.t());
   }
   public vetor refleteZ(){
      double vz = -this.z();
      return new vetor(this.x(), this.y(), vz, this.t());
   }
   /**
    * Projeta vetor no plano xy
    * @param vetor a ser projetado
    */
   public vetor XY(){
      return new vetor(this.x(),this.y(),0.,this.t());
   }
   public vetor XZ(){
      return new vetor(this.x(),0.,this.z(),this.t());
   }
   public vetor YZ(){
      return new vetor(0., this.y(), this.z(), this.t());
   }
   /**
    *  Gira um vetor, de angulo teta, em relacao ao eixo y
    *  @param teta o angulo de rotacao
    */
   public vetor giray(double teta){
      double novox, novoz;
      double seno = Math.sin(teta);
      double cosseno = Math.cos(teta);
      novox = this.x() * cosseno + this.z() * seno;
      novoz = -this.x() * seno + this.z() * cosseno;
```

```
        return new vetor(novox, this.y(), novoz, this.t());
      }
      /**
       *  Gira um vetor, de angulo teta, em relacao ao eixo z
       *  @param teta o angulo de rotacao
       */
      public vetor giraz(double teta){
        double novox, novoy;
        double seno = Math.sin(teta);
        double cosseno = Math.cos(teta);
        novox = this.x() * cosseno + this.y() * seno;
        novoy = -this.x() * seno + this.y() * cosseno;
        return new vetor(novox, novoy, this.z(), this.t());
      }
      /**
       *  Gira um vetor, de angulo teta, em relacao ao eixo x
       *  @param teta o angulo de rotacao
       */
      public vetor girax(double teta){
        double novoz, novoy;
        double seno = Math.sin(teta);
        double cosseno = Math.cos(teta);
        novoy = this.y() * cosseno + this.z() * seno;
        novoz = -this.y() * seno + this.z() * cosseno;
        return new vetor(this.x(), novoy, novoz, this.t());
      }

}
```

B.2 Código da classe opv

```
 package bonelli;
import bonelli.vetor;
import java.awt.*;
/**
 * Essa classe fornece operacoes com vetores.
 * Atencao: eixo-x sai da tela, eixo-y
 * para a direita, z para cima.
 * @author Prof. Enivaldo Bonelli - UFRN
 * @version 120322
 */
public class opv{

   static double zobs = 300.;
```

```
   static double xobs = 600.;
   static double yobs = 300.;
   static double deltaYobs = 30.;
   static int alturaDaTela = 400;
   static vetor posicaoDaVersao = new vetor(40., 60., 300);

   /**
    * Soma vetorial de dois vetores.
    * @param vetor A e vetor B a serem somados.
    */

   public static vetor soma(vetor A, vetor B){
      double Cx = A.x() + B.x();
      double Cy = A.y() + B.y();
      double Cz = A.z() + B.z();
      double Ct = 2.;
      vetor C = new vetor(Cx, Cy, Cz, Ct);
      return C;
   }

  /**
    * Soma vetorial de tres vetores.
    * @param vetor A, vetor B, e vetor D a serem somados.
    */

   public static vetor soma(vetor A, vetor B, vetor D){
      double Cx = A.x() + B.x() + D.x();
      double Cy = A.y() + B.y() + D.y();
      double Cz = A.z() + B.z() + D.z();
      double Ct = 3.;
      vetor C = new vetor(Cx, Cy, Cz, Ct);
      return C;
   }


/**
    * Soma vetorial de quatro vetores.
    * @param vetor A, vetor B, vetor D, e vetor E
    * a serem somados.
    */

   public static vetor soma(vetor A,vetor B,vetor D,vetor E)
   {
      double Cx = A.x() + B.x() + D.x() + E.x();
      double Cy = A.y() + B.y() + D.y() + E.y();
```

```
   double Cz = A.z() + B.z() + D.z() + E.z();
   double Ct = 4.;
   vetor C = new vetor(Cx, Cy, Cz, Ct);
   return C;
}

/**
 * Diferenca entre dois vetores.
 * @param vetor A e vetor B a ser subtraido do primeiro.
 */

public static vetor diferenca(vetor A, vetor B){
   double Cx = A.x() - B.x();
   double Cy = A.y() - B.y();
   double Cz = A.z() - B.z();
   double Ct = 0.;
   vetor C = new vetor(Cx, Cy, Cz, Ct);
   return C;
}
/**
 * Desenha vetor a partir de vetor ORIGEM
 * @param vetor, vetor origem, cor
 */
public static void desenhaVetor(
   Graphics g,vetor V,vetor ORIGEM,Color C)
   {
   vetor VPO, VPOP, ORIGEMP;
   g.setColor(C);
   VPO = soma(ORIGEM, V);
   ORIGEMP = projetaNaTela(ORIGEM);
   VPOP = projetaNaTela(VPO);
   int oy = (int) ORIGEMP.y();
   int oz = alturaDaTela - (int) ORIGEMP.z();
   int vy = (int) VPOP.y();
   int vz = alturaDaTela - (int) VPOP.z();
   g.drawLine(oy,oz,vy,vz);
   //Identifica a versao do 3djah
   opv.escreveTexto(g, posicaoDaVersao, "3DJAH 2012", 20);
}
/**
 * Desenha vetor a partir de vetor ORIGEM,
 * em duas cores para oculos 3D
 * @param vetor, vetor origem
 */
public static void desenhaVetor(
```

```
   Graphics g,vetor V,vetor ORIGEM)
  {
   vetor VPO, VPOP, ORIGEMP;
   VPO = soma(ORIGEM, V);
   //Identifica a versao do 3djah
   opv.escreveTexto(g, posicaoDaVersao, "3DJAH 2012", 20);
   // Imagem da direita
   ORIGEMP = projetaNaTela(ORIGEM, deltaYobs);
   g.setColor(Color.magenta);
   VPOP = projetaNaTela(VPO, deltaYobs);
   int oy = (int) ORIGEMP.y();
   int oz = alturaDaTela - (int) ORIGEMP.z();
   int vy = (int) VPOP.y();
   int vz = alturaDaTela - (int) VPOP.z();
   g.setFont(new Font("SansSerif", Font.PLAIN, 15));
   g.drawLine(oy,oz,vy,vz);

   // Imagem da esquerda
   ORIGEMP = projetaNaTela(ORIGEM, -deltaYobs);
   g.setColor(Color.cyan);
   VPOP = projetaNaTela(VPO, -deltaYobs);
   oy = (int) ORIGEMP.y();
   oz = alturaDaTela - (int) ORIGEMP.z();
   vy = (int) VPOP.y();
   vz = alturaDaTela - (int) VPOP.z();
   g.setFont(new Font("SansSerif", Font.PLAIN, 15));
   g.drawLine(oy,oz,vy,vz);

 }
/**
 * Apaga vetor a partir de vetor ORIGEM,  para oculos 3D
 * @param vetor, vetor origem
 */
public static void apagaVetor(
   Graphics g,vetor V,vetor ORIGEM)
{
   vetor VPO, VPOP, ORIGEMP;
   VPO = soma(ORIGEM, V);
   // Imagem da direita
   ORIGEMP = projetaNaTela(ORIGEM, deltaYobs);
   g.setColor(Color.white);
   VPOP = projetaNaTela(VPO, deltaYobs);
   int oy = (int) ORIGEMP.y();
   int oz = alturaDaTela - (int) ORIGEMP.z();
   int vy = (int) VPOP.y();
```

```
   int vz = alturaDaTela - (int) VPOP.z();
   g.drawLine(oy,oz,vy,vz);
   // Imagem da esquerda
   ORIGEMP = projetaNaTela(ORIGEM, -deltaYobs);
   g.setColor(Color.white);
   VPOP = projetaNaTela(VPO, -deltaYobs);
   oy = (int) ORIGEMP.y();
   oz = alturaDaTela - (int) ORIGEMP.z();
   vy = (int) VPOP.y();
   vz = alturaDaTela - (int) VPOP.z();
   g.drawLine(oy,oz,vy,vz);
 }

/**
 * Desenha uma oval (ou circunferencia), na ponta de um vetor.
 * @param vetor na ponta do qual sera desenhada a oval.
 * @param Dh eixo horizontal
 * @param Dv eixo vertical
 * @param cor
 */
public static void desenhaOval(
   Graphics g,vetor V,int Dh,int Dv,Color C)
{
   g.setColor(C);
   vetor Vp = projetaNaTela(V);
   double fator = 1./ (1. - (V.x()/xobs));
   // System.out.println("\n fator=" + fator);
   Dv = (int)(Dv * fator ) + 1;
   Dh = (int)(Dh * fator) + 1;
   int vy = (int) Vp.y() - Dh/2;
   int vz = alturaDaTela - (int) Vp.z() -Dv/2 ;
   g.drawOval(vy, vz, Dh, Dv);
}
/**
 * Desenha uma oval 3D-estereo (ou circunferencia),
 * na ponta de um vetor.
 * @param vetor na ponta do qual sera desenhada a oval.
 * @param Dh eixo horizontal
 * @param Dv eixo vertical
 */
public static void desenhaOval(
   Graphics g,vetor V,int Dh,int Dv){
   // imagem da direita
   vetor Vp;
   double fator = 1./ (1. - (V.x()/xobs));
```

```
      g.setColor(Color.magenta);
      Vp = projetaNaTela(V, deltaYobs);
      Dv = (int)(Dv * fator ) + 1;
      Dh = (int)(Dh * fator) + 1;
      int vy = (int) Vp.y() - Dh/2;
      int vz = alturaDaTela - (int) Vp.z() -Dv/2 ;
      g.drawOval(vy, vz, Dh, Dv);

      // Imagem da esquerda
      fator = 1./ (1. - (V.x()/xobs));
      g.setColor(Color.cyan);
      Vp = projetaNaTela(V, -deltaYobs);
      Dv = (int)(Dv * fator ) + 1;
      Dh = (int)(Dh * fator) + 1;
      vy = (int) Vp.y() - Dh/2;
      vz = alturaDaTela - (int) Vp.z() -Dv/2 ;
      g.drawOval(vy, vz, Dh, Dv);

   }
//
  /**
    * Escreve Texto 3D-estereo , na ponta de um vetor.
    * O usuario nao escolhe o tipo de fonte,
    * por enquanto s so o tamanho.
    * @param vetor na ponta do qual sera escrito o texto
    * @param texto
    * @param tamanho
    */
 public static void escreveTexto(
   Graphics g,vetor V,String texto,int tamanho)
   {
      // imagem da direita
      vetor Vp;
      double fator = 1./ (1. - (V.x()/xobs));
      g.setColor(Color.magenta);
      Vp = projetaNaTela(V, deltaYobs);
      int vy = (int) Vp.y() - tamanho/2;
      int vz = alturaDaTela - (int) Vp.z() -tamanho/2 ;
      tamanho = (int)(fator * tamanho);
      g.setFont(new Font("SansSerif", Font.PLAIN, tamanho));
      g.drawString(texto,vy, vz);

      // Imagem da esquerda
      fator = 1./ (1. - (V.x()/xobs));
      g.setColor(Color.cyan);
```

```
      Vp = projetaNaTela(V, -deltaYobs);
      vy = (int) Vp.y() - tamanho/2;
      vz = alturaDaTela - (int) Vp.z() -tamanho/2 ;
      tamanho = (int)(fator * tamanho);
      g.setFont(new Font("SansSerif", Font.PLAIN, tamanho));
      g.drawString(texto,vy, vz);
   }

 public static void apagaTexto(
     Graphics g,vetor V,String texto,int tamanho)
     {
      // imagem da direita
      vetor Vp;
      double fator = 1./ (1. - (V.x()/xobs));
      g.setColor(Color.white);
      Vp = projetaNaTela(V, deltaYobs);
      int vy = (int) Vp.y() - tamanho/2;
      int vz = alturaDaTela - (int) Vp.z() -tamanho/2 ;
      tamanho = (int)(fator * tamanho);
      g.setFont(new Font("SansSerif", Font.PLAIN, tamanho));
      g.drawString(texto,vy, vz);

      // Imagem da esquerda
      fator = 1./ (1. - (V.x()/xobs));
      g.setColor(Color.white);
      Vp = projetaNaTela(V, -deltaYobs);
      vy = (int) Vp.y() - tamanho/2;
      vz = alturaDaTela - (int) Vp.z() -tamanho/2 ;
      tamanho = (int)(fator * tamanho);
      g.setFont(new Font("SansSerif", Font.PLAIN, tamanho));
      g.drawString(texto,vy, vz);
   }



//
//
   /**
    * Desenha uma oval 3D-estereo (ou circunferencia),
    * na ponta de um vetor, com cor branca, de forma que
    * apaga a anterior, se o fundo for branco.
    * Nessa visao 3D-estereo, precisamos apagar duas ovais.
    * @param vetor na ponta do qual sera desenhada a oval.
    * @param Dh eixo horizontal
    * @param Dv eixo vertical
```

```
  */
public static void apagaOval(
    Graphics g,vetor V,int Dh,int Dv)
  {
    // imagem da direita
    vetor Vp;
    double fator = 1./ (1. - (V.x()/xobs));
    g.setColor(Color.white);
    Vp = projetaNaTela(V, deltaYobs);
    Dv = (int)(Dv * fator ) + 1;
    Dh = (int)(Dh * fator) + 1;
    int vy = (int) Vp.y() - Dh/2;
    int vz = alturaDaTela - (int) Vp.z() -Dv/2 ;
    g.drawOval(vy, vz, Dh, Dv);

    // Imagem da esquerda
    fator = 1./ (1. - (V.x()/xobs));
    g.setColor(Color.white);
    Vp = projetaNaTela(V, -deltaYobs);
    Dv = (int)(Dv * fator ) + 1;
    Dh = (int)(Dh * fator) + 1;
    vy = (int) Vp.y() - Dh/2;
    vz = alturaDaTela - (int) Vp.z() -Dv/2 ;
    g.drawOval(vy, vz, Dh, Dv);

 }

  /**
   * Calcula a projecao de um vetor num plano
   * paralelo ao da tela(plano yz) desenhaVetor
   * usa esse metodo para mostrar um vetor em 2D.
   * @param vetor de entrada, vetor saida.
   */
 public static vetor projetaNaTela(vetor Dado){
   vetor Projetado;
   double t = -Dado.x()/(xobs-Dado.x());
   double pz = Dado.z() + (zobs - Dado.z()) * t;
   double py = Dado.y() + (yobs - Dado.y()) * t;
   Projetado = new vetor(0,py,pz,Dado.t());
   return Projetado;
 }

  /**
   * Projeta na tela, tendo em vista deslocamento
   * do observador, do valor padrao, na direcao y.
```

```
  * Util para desenhar para oculos 3D.
  * Calcula a projecao de um vetor num plano
  * paralelo ao da tela(plano yz)
  * desenhaVetor usa esse metodo para mostrar um vetor em 2D.
  * @param vetor de entrada, deslocamento y  do observador.
  */
public static vetor projetaNaTela(vetor Dado,double deslocaY)
{
  vetor Projetado;
  double t = -Dado.x()/(xobs - Dado.x());
  double pz = Dado.z() + (zobs - Dado.z()) * t;
  double py = Dado.y() + (yobs + deslocaY - Dado.y()) * t;
  Projetado = new vetor(0,py,pz,Dado.t());
  return Projetado;
}
/**
 * Produto escalar de dois vetores.
 * @param dois vetores
 */
public static double produtoEscalar(vetor A, vetor B){
   double p = A.x() * B.x() + A.y() * B.y() + A.z() * B.z();
   return p;
}
/**
 * Angulo, em radianos, entre dois vetores
 * @param Dois vetores
 */
public static double angulo(vetor A, vetor B){
   double num = produtoEscalar(A,B);
   double den = A.modulo() * B.modulo();
   return Math.acos(num/den);
}
/**
 * Efetua o produto vetorial de dois vetores (claro!)
 * A quarta componente eh herdada do primeiro.
 * @param primeiro vetor e segundo vetor, em ordem.
 */
public static vetor produtoVetorial(vetor A, vetor B){
   vetor C;
   double px, py, pz;
   px = A.y() * B.z() - A.z() * B.y();
   py = A.z() * B.x() - A.x() * B.z();
   pz = A.x() * B.y() - A.y() * B.x();
   C = new vetor(px,py,pz,A.t());
   return C;
```

```
    }
    /**
     * Multiplica vetor por escalar.
     * @param vetor, escalar
     */
    public static vetor vetorXescalar(vetor VETOR,double escalar)
    {
       return escalarXvetor(escalar, VETOR);
    }
   /**
     * Multiplica escalar por vetor.
     * @param escalar, vetor
     */
    public static vetor escalarXvetor(double escalar,vetor VETOR)
    {
       double vx = escalar * VETOR.x();
       double vy = escalar * VETOR.y();
       double vz = escalar * VETOR.z();
       vetor produto = new vetor(vx,vy,vz,VETOR.t());
       return produto;
    }
     /**
     * Desenha vetor a partir da origem absoluta da tela
     *@param vetor, cor
     */
    public static void desenhaVetor(Graphics g,vetor V,Color C)
    {
       g.setColor(C);
       int vx = (int) V.x();
       int vy = (int) V.y();
       //int vz = (int) V.z();
       g.drawLine(0,0,vx,vy);
     }
}
```

Bibliografia

[1] R. J. Deitel e H. M. Deitel, *Java, como Programar.* Bookman, Porto Alegre, 2001.

[2] K. C. Hopson e S. E. Ingram, *Desenvolvendo Applets Com JAVA.* Campus, Rio de Janeiro, 1997.

Índice

Applet, 13, 27
Appletviewer, 41

Códigos
- curva 3D, 46
- eixos3D, 63
- EsféricasDemo2009, 65
- giraObservador, 52
- giraObsEscondido, 55
- projetilOvalE, 49
- vetorGirante, 41
- vetorGirante4, 41

Classes, 14
Coordenadas esféricas, 65
Curva parametrizada, 45

Desenho
- Apagando, 37
- ApagaVetor, 43
- Curvas, 45
- Curvas 3D, 46
- Escondido, 37, 55
- Oval, 19, 49
- Pausa, 38
- Texto, 52
- Vetor, 37

Girando curvas, 57

Java, 13

LissajousRandE, 58

Origem absoluta, 24

Pausa, 66

Trajetória, 45

Vetor
- Origem, 46
- Resultante, 46