PhotoPixJ: Ambiente para Execução e Integração de Algoritmos de Processamento Digital de Imagens

Adriana Cássia Rossi de Almeida
Orientador: Arnaldo de Albuquerque Araújo
Co-orientador: Alisson Augusto Souza Sol

Dissertação apresentada ao Departamento de Ciência da Computação do Instituto de Ciências Exatas da Universidade Federal de Minas Gerais, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte, 31 de março de 1998

Resumo

Este trabalho apresenta o PhotoPixJ, um sistema para a execução e integração de algoritmos de Processamento Digital de Imagens. Foi construída uma plataforma que visa proporcionar fácil implementação de tipos de imagens, algoritmos de processamento e formatos de imagens. O sistema foi implementado em Java™ e apresenta as versões aplicação e *applet* com as restrições de segurança existentes para a implementação de *applets*. O sistema é multiplataforma e, para extendê-lo, através da adição de novos algoritmos e recursos, é necessária apenas uma implementação para todas as plataformas de execução. O uso do PhotoPixJ proporciona um grande ganho de produtividade para o desenvolvedor de algoritmos de PDI, na implementação não apenas de algoritmos mas também de diversas funções não-fundamentais.

Abstract

This work presents the PhotoPixJ, a system for the execution and integration of Digital Image Processing algorithms. It was build a framework that intend to enable an easy way to implement image types, image processing algorithms and image formats. The system was implemented in Java and presents the application and the applet versions with the security restrictions for applets implementation. The system is multiplataform and, to extend it, with the adittion of new algorithms and other resources, is necessary only one implementation for all plataforms of execution. The use of PhotoPixJ enables a great productivity gain for DIP algorithms developers, in the implementation of not only algorithms but also a lot of non-essencial functionalities.

Agradecimentos

Sou muito grata ao Professor Arnaldo de Albuquerque Araújo, pela orientação e sugestões valiosas que possibilitaram o desenvolvimento deste trabalho; ao meu co-orientador, Alisson Souza Sol, pelo apoio prestado e por sua disponibilidade dando sugestões de grande valia e ajudando em todo o desenvolvimento deste trabalho.

Agradeço aos meus pais que sempre me deram total apoio em toda a minha vida escolar e acadêmica, se sacrificando sempre que preciso, para que eu pudesse obter uma formação digna. Agradeço, em especial, pelo apoio dado nos momentos difíceis.

Agradeço ao meu irmão, Léo, pelo interesse no meu trabalho e apoio dispensado juntamente com meus pais.

Sou muito grata ao Luiz que esteve tão presente nos últimos meses, dando apoio e ajudando, das mais variadas formas, para a conclusão deste trabalho.

Agradeço à minhas amigas Fátima e Lígia, pelo incentivo dado para que concluísse este trabalho.

Sou grata aos amigos Marco Aurélio e Márcio Henrique que me prestaram auxílio em todos os momentos em que os solicitei ajuda.

Agradeço a todos os familiares e amigos por toda a atenção e apoio dispensados durante toda a tragetória deste trabalho, árduo, porém gratificante, no meu curso de mestrado.

Agradeço ainda ao CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio financeiro, que foi fundamental para o desenvolvimento deste trabalho de pesquisa.

Sumário

Lista de Figuras

Lista de Trechos de Código Fonte

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação e Objetivos

A área de processamento digital de imagens [Gonza93], [Jain89] pode ser dividida em duas categorias de funcionalidade: a primeira consiste na melhoria da qualidade das imagens digitais, e outra categoria consiste na extração de informação da imagem em um processo automático ou semi-automático. Ambas baseiam-se na aplicação de algoritmos de tratamento da imagem para atingir um objetivo específico.

Os implementadores de sistemas de PDI[1] se deparam com um problema que não os permite concentrarem-se unicamente na implementação dos algoritmos para tratamento de imagens: a diversidade de plataformas de *hardware* e de APIs[2] disponíveis, que dificulta a portabilidade dos sistemas, em especial do código relativo à interface com o usuário.

Com o propósito de se obter uma solução para o problema anteriormente exposto, analisou-se a filosofia da linguagem de programação Java [Arnold96], [Campio96], [Dacont96], [Flanag96], [Gosli96a], [Gosli96b], [Gosli96c], [JavaSoft] que apresenta a característica de ser independente de plataforma de *hardware*. Os programas em Java são compilados para o formato *bytecode* de uma arquitetura neutra. Uma aplicação em Java (no formato *bytecode*) pode ser executada em qualquer sistema operacional, bastando que o mesmo implemente a Máquina Virtual Java.

O objetivo deste trabalho foi a definição, projeto e concepção do PhotoPixJ, um sistema independente de plataforma de *hardware* para execução de algoritmos de PDI, que apresenta

---

[1] PDI: Processamento Digital de Imagens.

[2] API: *Application Programming Interface*, ou Interface de Programação do Usuário.

um conjunto de classes que proporciona fácil implementação e integração de novos algoritmos, além de outras funções como recursos de interface com o usuário.

1.2 Trabalhos Relacionados

Este trabalho partiu do estudo do sistema PhotoPix [Sol93], [Sol93b], [Sol95]. O trabalho em [Sol93], descreve como o uso de técnicas de análise e projeto orientados para objeto [Booch94], [Coad95], [Gamma95], [Jacobs92], [Jacobs94], [Martin95], [Pree94] podem ajudar a isolar a implementação de algoritmos de sistemas de PDI da codificação de funcionalidade "não essencial". Com essa diretiva, o sistema PhotoPix foi construído visando ser estendido em processos posteriores, adicionando-se a ele, novos algoritmos de tratamento de imagens e novos recursos.

O sistema PhotoPix foi implementado na linguagem C++, usando a interface para programação de aplicativos do ambiente MS-Windows®. A adição de algoritmos nesse sistema implica em um aprendizado da linguagem C++, que apresenta um certo grau de dificuldade. Além disso, para se implementar interfaces gráficas específicas dos algoritmos é necessário o aprendizado da API da plataforma específica (no PhotoPix, a API do MS-Windows) o que dificulta a portabilidade do sistema.

O uso da linguagem Java apresenta vantagens em relação à plataforma de desenvolvimento do sistema PhotoPix. No sistema objeto deste trabalho, o PhotoPixJ, para se adicionar algoritmos e outros recursos ou para a implementação de elementos de interface gráfica, utiliza-se a API de Java, que é portátil para vários sistemas.

Outros ambientes como o Khoros™ [Khoral] proporcionam a integração de programas particulares ao sistema. No entanto, o grau de abstração de seus elementos requer um conhecimento relativamente alto da arquitetura interna desses sistemas para que programas particulares sejam implementados e integrados a eles, o que pode ser uma tarefa difícil inicialmente. A abstração das classes do PhotoPixJ visa tornar a adição de novos algoritmos de PDI o mais fácil possível e com alto grau de independência de outros aspectos do sistema.

1.3 Notações

Os exemplos de código mostrados neste trabalho estão na linguagem Java[3] [Arnold96], [Gosli96b], [Gosli96c], [Flanag96], [JavaSoft]. Os diagramas de classes mostrados usam a notação do padrão UML (*Unified Modeling Language*) [Fowler97], [Rational].

---

[3] Utilizou-se neste trabalho a linguagem Java versão 1.0.2.

1.4 A Estrutura da Dissertação

O Capítulo 2 apresenta uma descrição da linguagem Java e apresenta as principais classes e funções que proporcionam representação e processamento de imagens presentes na API da linguagem. Esse capítulo pode ser utilizado como um guia objetivo para escrever programas de processamento digital de imagens utilizando a API de Java.

O Capítulo 3 apresenta uma revisão bibliográfica dos estudos sobre bibliotecas de classes para sistemas multimídia, apresentando uma plataforma para sistemas multimídia, base para a plataforma PhotoPixJ.

O Capítulo 4 descreve a especificação, projeto e implementação do PhotoPixJ. Apresenta a interface com o usuário, o sistema de ajuda e os ambientes de funcionamento do PhotoPixJ.

O Capítulo 5 descreve o procedimento para inserção de um algoritmo de PDI no PhotoPixJ.

A conclusão do trabalho é apresentada no Capítulo 6, seguindo-se os Apêndices e a Bibliografia.

Capítulo 2

A Linguagem Java e Processamento de Imagens

2.1 Introdução

Java iniciou-se como linguagem para *software* de eletrodomésticos. *Software* para dispositivos de consumo requerem certas funções, como por exemplo, funcionar quando um novo *chip* é introduzido. Foi então, desenvolvida pela Sun Microsystems® uma nova linguagem que era pequena, legível e independente de arquitetura. Posteriormente, percebeu-se que a linguagem poderia ser ideal para programação na Internet, uma vez que os programas poderiam ser executados em todos os tipos de computadores conectados à rede. Assim, Java evoluiu e foi lançada como linguagem de programação para construção de aplicações para páginas *Web* (*applets*), bem como linguagem de programação para construção de aplicações multiplataforma.

Neste capítulo, serão expostas características importantes presentes na linguagem Java. Será explicado como a manipulação e processamento de imagens podem ser feitos em Java. Este capítulo pode ser utilizado como um guia objetivo para escrever programas de processamento digital de imagens utilizando a API de Java. Para isso, é necessário que se tenha familiaridade com a linguagem [Campio96], [Dacont96], [Flanag96], [Gosli96c].

2.2 Características da Linguagem Java

Em [Gosli96a] Java é descrita como uma linguagem simples, orientada para objetos, distribuída, interpretada, robusta, segura, independente de arquitetura, portátil, de alto desempenho, de múltiplos processos (*multithreaded*) e dinâmica.

A fim de entender o potencial de Java, serão examinadas algumas características da linguagem.

2.2.1 Simples

Java é uma linguagem simples. Um dos objetivos em um projeto de uma linguagem simples [Gosli96a] é possibilitar que um programador possa aprendê-la rapidamente, portanto o número de construções da linguagem deve se manter pequeno. Outro plano de projeto consiste em fazer uma linguagem familiar para a maioria dos programadores de forma a se obter fácil migração. Nesse contexto, Java usa muitas das mesmas construções presentes em C e C++.

A fim de manter a linguagem tanto pequena quanto familiar, foram removidos uma série de recursos disponíveis em C e C++. Esses recursos são aqueles que, na maioria das vezes, levam a más práticas de programação ou são raramente usados. Por exemplo, Java não suporta a expressão goto; em contrapartida, provê as expressões break e continue e tratamento de exceção. Java não usa arquivos de cabeçalho e portanto elimina o pré-processamento que existe em C. Pelo fato de Java ser uma linguagem orientada para objetos, construções como struct e union foram eliminadas. Java também eliminou *operator overloading* e herança múltipla, recursos presentes em C++.

Talvez, a mais importante simplificação, entretanto, é que Java não usa apontadores. Apontadores, são uns dos elementos que mais provocam erros na programação em C e C++. Java automaticamente faz o controle de referência dos objetos para o programador. Java também implementa "coleta de lixo" (*garbage collection*), de forma que não é necessário se envolver com problemas de controle de liberação de memória. Nessa técnica, objetos que não estão mais sendo usados (um objeto não está mais em uso quando não há mais referências para ele) são automaticamente detectados e liberados da memória. Tudo isso livra o programador de se preocupar com apontadores pendentes, referências inválidas de apontadores, e dessa forma, pode-se despender o tempo desenvolvendo a funcionalidade dos programas.

2.2.2 Orientada para Objetos

Java é uma linguagem orientada para objetos. Em um sistema orientado para objetos [Booch94], [Jacobs92], [Pree94] uma classe define um tipo abstrato de dados que consiste em uma coleção de dados e métodos que operam sobre esses dados. Juntos, os dados e métodos descrevem o estado e comportamento de um objeto que constitui uma instância de determinada classe definida.

As principais características de uma linguagem orientada para objetos são a capacidade de abstração, o encapsulamento de dados, a hierarquia de classes e modularidade. Como

características secundárias, tem-se a consistência de tipos, a concorrência e persistência. Java apresenta todas essas características.

2.2.3 Interpretada

O compilador Java gera *bytecodes*, ao invés de código nativo da máquina. Para de fato executar um programa em Java, utiliza-se o interpretador Java que executa os *bytecodes* compilados, portanto, Java é uma linguagem interpretada[4]. Os *bytecodes* de Java provêem um arquivo objeto em um formato independente de arquitetura; o código é projetado para transportar programas eficientemente para múltiplas plataformas. Um programa em Java pode ser executado em qualquer sistema que implementa o interpretador Java e o sistema de tempo de execução. Em conjunto, o interpretador e o sistema de tempo de execução implementam a máquina virtual denominada Máquina Virtual Java.

Em um ambiente interpretado, a tradicional fase de ligação de referências (*link*) no desenvolvimento de um programa desaparece. Se Java tem uma fase de ligação dinâmica de fato, consiste apenas em um processo de carregar novas classes no ambiente, o que é um processo incremental com menor custo que a fase de *link* de compiladores tradicionais. Como resultado, Java suporta rápida prototipagem e fácil experimentação, o que pode levar ao desenvolvimento mais rápido de programas.

2.2.4 Segura

Um dos mais importantes aspectos da linguagem Java é que ela é uma linguagem segura [Gosli96a]. Segurança é uma preocupação importante, uma vez que Java é usada em ambientes de rede. Sem alguma garantia de segurança, um usuário certamente não iria querer obter uma *applet* de um servidor qualquer da Internet e deixá-la executando em seu computador. Java implementa vários mecanismos de segurança a fim de proteger o usuário contra códigos que possam tentar criar um vírus ou invadir seu sistema de arquivos.

Não se pode garantir que um programa em Java não conterá código malicioso de forma alguma. O que se fez no projeto da linguagem teve a intenção de dificultar que códigos maliciosos e danosos sejam escritos em Java. Java se antecipou e se defendeu contra a maioria das técnicas que têm sido historicamente usadas em *softwares* mal intencionados. Não é possível dizer que Java ou qualquer outro sistema seja totalmente seguro. Podem ser inventadas novas formas para atacar sistemas de *software*, e os projetistas da linguagem deverão estar a par dessas novas situações. Na prática, o que se consegue é uma segurança "razoável", onde "razoável" depende de algumas perdas e ganhos entre funcionalidade e segurança.

---

[4]Alguns autores preferem os termos semi-compilada ou semi-interpretada.

2.2.5 Independente de Arquitetura

Os programas em Java são compilados para um formato, o *bytecode*, independente de arquitetura. O código em *bytecode* é então interpretado durante a execução. A principal vantagem dessa solução é permitir às aplicações Java serem executadas em qualquer plataforma, bastando que a mesma implemente a Máquina Virtual Java. Ser independente de plataforma é característica que facilita uma outra: a portabilidade dos sistemas implementados na linguagem.

É extremamente desejável ter-se um sistema funcionando nas diversas plataformas existentes. Com a diversidade de sistemas fica difícil produzir *software* para todas as plataformas. Uma linguagem de programação independente de plataforma visa alcançar esse objetivo.

2.2.6 De Alto Desempenho

Sem o uso de técnicas para melhoria do desempenho na execução, aplicativos em Java, compilados para *bytecode*, apresentam desempenho satisfatório em aplicações interativas, GUI[5] e baseadas em rede, onde a aplicação está freqüentemente desocupada, esperando um usuário fazer alguma coisa, ou esperando por algum dado da rede.

Uma forma de se melhorar o desempenho é utilizar um compilador JIT[6] [Kramer96], [SunJIT]. Ele traduz o código em bytecode para o código específico da máquina antes do programa ser executado. O desempenho dos bytecodes convertidos para código de máquina são equiparáveis a de códigos nativos em C ou C++.

Java se encaixa no meio do espectro entre linguagens totalmente interpretadas e linguagens totalmente compiladas. O desempenho dos bytecodes interpretados em Java é melhor que o das linguagens script de alto nível, e ainda oferece a simplicidade e portabilidade de todas essas linguagens. Embora programas em Java usualmente não apresentem execução com desempenho tão alto), ela provê um ambiente independente de arquitetura onde se é possível escrever programas confiáveis.

2.2.7 Outras Características

Java é uma linguagem de múltiplos processos (*multithreaded*). Ela provê suporte para múltiplas *threads* de execução, também chamadas processos leves (*lightweight processes*) que podem tratar diferentes tarefas. O suporte nativo para implementação de múltiplos processos evita a necessidade de se usarem bibliotecas, que, por não estarem disponíveis em todos os ambientes, dificultam a portabilidade dos programas.

---

[5]GUI: *Graphical User Interface*, ou Interface de Usuário Gráfica.

[6] JIT: *Just in Time*.

Outra característica importante presente em Java é o suporte ao tratamento de exceções. As exceções permitem o tratamento de erros de forma regular e lógica por permitirem que todo o código de tratamento de exceção seja reunido em um lugar. Ao invés de se preocupar com todas as coisas que podem dar errado com cada linha de código, pode-se concentrar no algoritmo em questão e colocar todo o código de tratamento de erro em um único lugar.

2.3 Processamento de Imagens

Em Java existem classes que representam e proporcionam manipulação de imagens. A classe que representa uma imagem denomina-se Image e pertence ao pacote java.awt [Espese97], [Geary97], [Gosli96c]. Classes existentes para manipulação de imagens estão presentes no pacote java.awt.image.

A classe java.awt.Image é uma classe abstrata que define métodos que provêem informação sobre uma imagem. A Tabela 2.1 lista os métodos definidos para a classe java.awt.Image.

Método
public abstract int getWidth(ImageObserver Observer);
public abstract int getHeight(ImageObserver Observer);
public abstract ImageProducer getSource();
public abstract Graphics getGraphics();
public abstract Object getProperty(String Name, ImageObserver Observer);
public abstract void flush();

Tabela 2.1: Métodos da classe java.awt.Image.

Praticamente toda a infra-estrutura para criação e manipulação de imagens está no pacote java.awt.image (que não deve ser confundido com a classe java.awt.Image). O pacote java.awt.image define interfaces para produção e obtenção de uma imagem, juntamente com classes para manipulação de imagens. A classe java.awt.Image provê uma referência para a imagem e é o pacote java.awt.image que possui classes que proporcionam todo o trabalho associado com a obtenção e manipulação das imagens.

2.3.1 Representação de uma Imagem

Uma imagem em Java é representada [Espese97] como um arranjo bidimensional de *pixels*[7] (Figura 2.1), onde o *pixel* mais à esquerda, no topo, é localizado na coordenada (0, 0) e o *pixel* mais à direita, na parte inferior, é localizado na coordenada (largura da imagem - 1, comprimento da imagem - 1).

[7] *Pixels*: *Picture Elements*, ou Elementos de Imagem.

Cada *pixel* tem seu próprio valor RGB representado por três *bytes*, um para cada componente de cor: vermelho, verde e azul (*red, green, blue*). Esses *bytes* são todos empacotados em um inteiro. Assim, para alterar o valor de um *pixel*, é necessário fazer acesso a um valor ao invés de três valores separados. Além da informação do vermelho, verde e azul, cada *pixel* tem uma informação de transparência, denominado **canal Alpha**. Esse canal também é descrito por um *byte*. Se um *pixel* tem um valor alpha igual a 0, significa que ele é totalmente transparente; se seu valor é 255 o *pixel* é opaco. O canal alpha é integrado no mesmo inteiro em que os componentes vermelho, verde e azul são integrados.

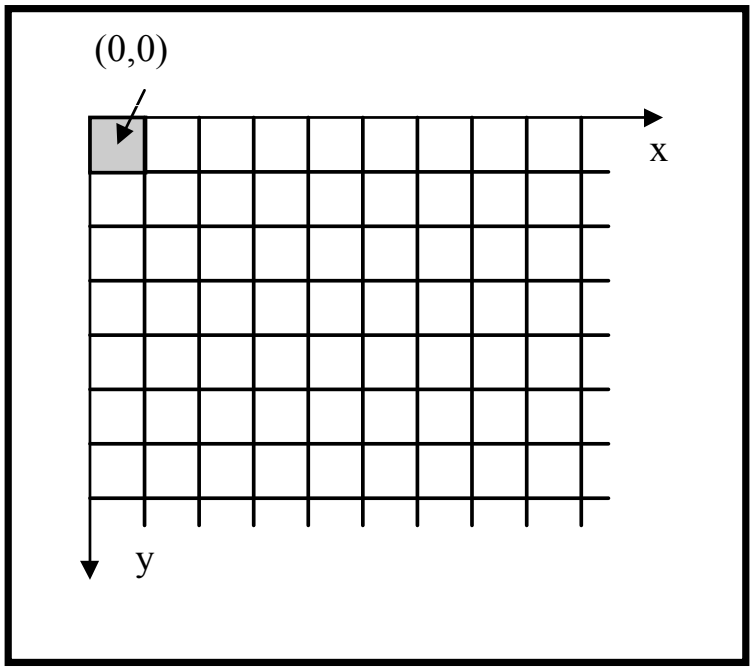

Figura 2.1: Arranjo bidimensional dos *pixels* de representação de uma imagem em Java.

Uma vez que Java armazena cada componente vermelho, verde e azul como um *byte*, cada componente pode representar 256 diferentes intensidades da cor correspondente. Isso significa que se pode trabalhar com mais de 16,7 milhões de cores. Isso é freqüentemente referenciado como cores reais (*true-color)*. Não é necessário se preocupar com a quantidade de cores que um usuário está apto a mostrar em seu sistema. Java cuida de converter a imagem em cores reais para o formato adequado antes de mostrá-la.

Os componentes de cor são organizados em um inteiro, como mostrado na linha a seguir:

```
0xAARRGGBB
```

Para se extrair os valores dos componentes vermelho, verde e azul mais o canal Alpha, de um inteiro `rgb`, pode-se usar o seguinte código:

```
Byte Alpha = (rgb & 0xff000000) >> 24;
Byte Red = (rgb & 0xff0000) >> 16;
Byte Green = (rgb & 0xff00) >> 8;
Byte Blue = rgb & 0xff;
```

2.3.2 Obtenção e Visualização de uma Imagem

Na API de Java, existem classes com métodos para a obtenção de imagens a partir da especificação de um arquivo ou de uma URL[8]. Em Java 1.0.2, é possível abrir uma imagem nos formatos GIF[9] e JPEG[10] [Kay92], [Murray94] de forma direta. Para a obtenção de imagens em outros formatos é necessária a implementação de decodificadores apropriados.

O Código 2.1 apresenta uma *applet* que obtém uma imagem e a mostra em um *canvas*. Nela são impressas as dimensões da imagem no método showImageSize.

```
import java.net.URL;
import java.awt.*;

public class ImageTestAppletSimple extends java.applet.Applet {
   private Image Img;

   public void init() {
      Img = getImage (getCodeBase(), "lena.gif");
      ShowImageSize();
   }
   public void paint(Graphics G)  {
      G.drawImage(Img, 0, 0, this);
   }
   private void showImageSize() {
      System.out.print("Image width = " + Img.getWidth(this));
      System.out.println(" height = " + Img.getHeight(this));
   }
}
```

Código 2.1: *Applet* que obtém uma imagem.

---

[8] URL: *Uniform Resource Locator.*

[9] GIF: *Graphical Interface Format.*

[10] JPEG: *Joint Photographic Experts Group.*

O método getImage da classe Applet retorna uma imagem (classe java.awt.Image). Para ilustrar o assincronismo do carregamento da imagem, chama-se showImageSize imediatamente após getImage ter retornado a imagem Img. Se a *applet* for executada, o método showImageSize imprimirá as seguintes largura e altura da imagem:

```
Image width = – 1 height = – 1
```

Isso se deve ao fato da imagem não ter sido totalmente carregada no momento em que se fez chamada aos métodos getWidth e getHeight; eles retornam – 1 até que a imagem seja totalmente carregada.

Quando se executa a *applet* no Código 2.1, pode-se ver a imagem sendo carregada por partes, de cima para baixo. Isso se deve ao fato do método drawImage também ser executado assincronamente. Observando-se a chamada à função:

```
G.DrawImage (Img, 0, 0, this)
```

é passado ao método drawImage:

- uma imagem (Image Img);
- coordenadas x, y (0, 0) (a quina superior à esquerda da imagem);
- Um objeto ImageObserver (this);

O último argumento, this, é a própria *applet*. Seguindo a hierarquia da definição da classe java.applet.Applet vê-se que ela é uma extensão da classe java.awt.Component que implementa um java.awt.ImageObsever. Classes como a Applet, que precisam lidar com imagens assincronamente, implementam a interface ImageObserver. A interface ImageObserver define uma coleção de constantes e um método, imageUpdate, para uso pelas classes que lidam com imagens. A classe imageUpdate é definida como:

```
public boolean imageUpdate(Image Img, int InfoFlags, int X, int Y, int Width, int Height);
```

Todos os métodos assíncronos que lidam com imagens (métodos da classe java.awt.Image) têm um ImageObserver como um argumento.

Para se obter uma imagem em uma *applet*, utiliza-se o método getImage presente na classe Applet. Como uma aplicação usual não é uma extensão da classe Applet, existe uma forma diferente para se obter imagens dentro de aplicações. O Código 2.2 ilustra como se obtém e se mostra uma imagem em uma aplicação.

Em uma aplicação, uma imagem é obtida usando-se o método getImage da classe java.awt.Toolkit. Primeiramente, chama-se o método estático getDefaultToolkit, que retorna o *toolkit default* para a plataforma em que a aplicação está executando, e posteriormente,

chama-se o método getImage da classe Toolkit. O método getImage da classe Toolkit pode receber tanto uma URL quanto uma String como argumento.

```
import java.awt.*;

public class ImageTestApplication extends Frame {
   Image Img;

   static public void main(String args[]) {
      ImageTestApplication App = new ImageTestApplication();
   }
   public ImageTestApplication() {
      super("Image Test");
      Img = Toolkit.getDefaultToolkit().getImage("lena.gif");
      reshape(100, 100, 220, 330);
      show();
   }
   public void paint(Graphics G) {
      G.drawImage(Img, 0, 0, this);
   }
   public boolean handleEvent(Event Ev) {
       if (Ev.id == Event.WINDOW_DESTROY)
          System.exit(0);
       return false;
   }
}
```

Código 2.2: Obtenção e visualização de uma imagem em uma aplicação.

2.3.3 Espera do Carregamento de uma Imagem

Uma imagem é carregada por partes, do topo até a parte inferior, devido à natureza assíncrona de seu carregamento [Geary97]. Uma forma mais agradável de se mostrar uma imagem seria aguardar que todos os seus *pixels* fossem carregados antes. Além disso, para se manipular os *pixels* de uma imagem, pode ser conveniente esperar que todos os *pixels* sejam carregados antes de se tentar ter acesso a eles. Uma forma de esperar que os *pixels* de uma imagem sejam totalmente carregados antes de manipulá-los consiste em implementar o método imageUpdate, da interface ImageObserver, que informa sobre o carregamento de uma imagem em questão. Uma solução mais conveniente para se esperar que os *pixels* sejam carregados consiste na utilização da classe java.awt.MediaTracker. Essencialmente, um objeto MediaTracker pode

monitorar o carregamento de uma imagem, e, dessa forma, espera-se que a imagem seja carregada para posteriormente fazer-se uso dela (mostrando-a ou fazendo acesso a seus *pixels*).

Implementação do Método imageUpdate

A *applet* no Código 2.3 eqüivale à do Código 2.1, exceto que ela implementa o método imageUpdate.

```
import java.net.URL;
import java.awt.*;

public class ImageTestAppletWithUpdate extends Applet {
   private Image Img;

   public void init() {
      Img = getImage (getCodeBase(),"lena.gif");
      showImageSize();
   }
   public void paint(Graphics G) { G.drawImage(Img, 0, 0, this); }
   public boolean imageUpdate(Image I, int Flags, int x, int y, int w, int h  ) {
      System.out.println("imageUpdate: x = " + x + ", y = " + y +
                              " w = " + w + ", h = " + h);
      if ((Flags & ALLBITS) != 0)
         repaint();
      return true;
   }
   private void showImageSize() {
      System.out.print("Image width = " + Img.getWidth(this));
      System.out.println(" height = " + Img.getHeight(this));
   }
}
```

Código 2.3: Implementação do método imageUpdate.

Na implementação da *applet* no Código 2.3, imprime-se o x, y, largura(w), altura(h) a cada vez que uma linha de bytes é obtida. Ao se executar a *applet*, ter-se-á algo do tipo impresso:

```
Image Width = – 1 Height = – 1
imageUpdate: x = 0, y = 0 w = 128, h = 128
imageUpdate: x = 0, y = 0 w = 128, h = 1
imageUpdate: x = 0, y = 1 w = 128, h = 1
imageUpdate: x = 0, y = 2 w = 128, h = 1
imageUpdate: x = 0, y = 3 w = 128, h = 1
imageUpdate: x = 0, y = 4 w = 128, h = 1
```

Não aparece explicitamente, nesta *applet*, a chamada ao método imageUpdate. Toda imagem (classe Image) tem um ImageProducer associado, que provê os *bits* para a imagem [Geary97]. Tão logo a chamada ao getImage é feita, o ImageProducer para a imagem Img começa a carregá-la. À medida em que a imagem vai sendo carregada, o ImageProducer associado com a imagem chama o método imageUpdade toda vez que ele obtém uma nova linha de *pixels*. No exemplo, a cada linha que é obtida, imprime-se algumas estatísticas para ilustrar a natureza assíncrona do carregamento da imagem.

Na implementação do imageUpdate do Código 2.3, utiliza-se a constante ALLBITS da interface ImageObserver para determinar quando a imagem é carregada por completo. Quando a expressão (Flags & ALLBITS) for diferente de zero, sabe-se que todos os *bits* da imagem foram carregados. Uma vez que a imagem foi totalmente carregada, chama-se o método repaint, que chamará o método paint, que desenhará a imagem.

Uso da classe MediaTracker

Uma solução mais conveniente para esperar que uma imagem seja carregada consiste na utilização da classe java.awt.MediaTracker. Essencialmente, um objeto MediaTracker pode monitorar o carregamento de uma imagem, e, dessa forma, espera-se que a imagem seja carregada para posteriormente fazer-se uso dela.

Para se carregar uma imagem, utiliza-se o método getImage em uma *applet*:

```
Image Lena = getImage("/tmp/lena.gif");
```

ou em uma aplicação:

```
Image Lena = Toolkit.getDefaultToolkit().getImage("/tmp/lena.gif");
```

Para esperar que todos os *pixels* sejam carregados, utiliza-se a classe java.awt.MediaTracker. Para fazer isso, um objeto MediaTracker é adicionado para monitorar o processo de carregamento, como mostrado no seguinte trecho de código:

```
    int ID = 0;
1   MediaTracker Tracker = new MediaTracker(this);
    Image Lena = Toolkit.getDefaultToolkit().getImage("/tmp/lena.gif");
2   Tracker.addImage(Lena, ID);
3   try {
        Tracker.waitForID(ID);
    } catch (InterruptedException E);
```

Esse trecho de código começa a carregar a imagem requisitada "/tmp/lena.gif", e se certifica de que todos os *pixels* sejam recebidos antes de continuar. Usar um objeto MediaTracker é simples e pode ser descrito em três passos:

- Criar uma instância da classe MediaTracker, como na linha 1.
- Usar o método addImage para especificar a imagem que precisa ser monitorada, como na linha 2.
- Criar um bloco try/catch, como na linha 3. O bloco espera pela imagem associada com o ID ser totalmente carregada. O método waitForID pode gerar uma exceção, portanto é necessário implementar um bloco catch. Nesse caso, é feito um *catch* mas não se faz nada com ele.

2.3.4 Criação de uma Imagem

Para a criação de uma imagem, deve-se, primeiramente, construir um arranjo de inteiros (*pixels*) correspondendo ao tamanho da nova imagem. Como exemplo, se a imagem tiver as proporções de 200x100 *pixels*, deve-se construir um arranjo de 20.000 inteiros, como mostrado a seguir:

```
int ImageWidth = 200, ImageHeight = 100;
int Pixels = new int [ImageWidth * ImageHeight];
```

O arranjo deve ser unidimensional, embora seja mais fácil o acesso individual dos *pixels*, passando as coordenadas x e y de um arranjo bidimensional. Conhecendo a largura da imagem (ImageWidth), o deslocamento correto dos *pixels* pode ser facilmente calculado a partir das coordenadas x e y:

```
Pixels[y * ImageWidth + x] = NewColor;
```

Muitas vezes, é desejável que o acesso aos *pixels* seja feito na mesma ordem em que eles são representados no arranjo, dessa forma, a representação unidimensional de armazenamento das cores é conveniente.

Uma vez preenchido o arranjo de cores no formato 0xAARRGGBB, cria-se a imagem utilizando o método createImage e um objeto da classe MemoryImageSource como parâmetro:

```
Image MyImage = createImage(new MemoryImageSource
                                    (ImageWidth, ImageHeight, Pixels, 0, ImageWidth));
```

A classe MemoryImageSource produz valores de *pixels* para uma imagem a partir de um arranjo. No exemplo, se produz *pixels* para uma imagem de cores reais, mas pode-se produzir *pixels* para imagens com outros modelos de cor (veja Seção 2.3.6).

O método createImage está disponível na classe java.awt.applet.Applet. Ao se criar uma imagem dentro de uma aplicação, utiliza-se o método createImage da classe java.awt.Toolkit, da seguinte forma:

```
Image MyImage = Toolkit.getDefaultToolkit().createImage (new MemoryImageSource
                                    (ImageWidth, ImageHeight, Pixels, 0, ImageWidth));
```

Java transforma a representação em cores reais da imagem para um formato adequado para a visualização. A imagem, poderá, por exemplo, ser passada ao método drawImage que desenhará a imagem. Um exemplo de uma *applet* em que se cria uma imagem de uma fonte de cores reais pode ser vista no Código 2.4. O resultado da execução dessa *applet* pode ser visto na Figura 2.2.

```
import java.awt.*;
import java.awt.image.*;

public class CreateImage extends java.applet.Applet {
   Image MyImage;

   public void init() {
      int ImageWidth = size().width;
      int ImageHeight = size().height;
      int Pixels[] = new int [ImageWidth * ImageHeight];

      int Index = 0;
      for (int Y = 0; Y < ImageHeight; Y++) {
         int Red = (Y * 255) / (ImageHeight - 1);
         int Green = 255 - (Y * 255) / (ImageHeight - 1);
         for (int X = 0; X < ImageWidth; X++) {
            int Alpha = (X * 255) / (ImageWidth - 1);
            int Blue = (X * 255) / (ImageWidth - 1);
            Pixels[Index++] = (Alpha<<24) | (Red<<16) | (Green<<8) | Blue;
         }
      }
      MyImage = createImage(new MemoryImageSource
                        (ImageWidth, ImageHeight, Pixels, 0, ImageWidth));
   }
   public void paint(Graphics G) {
      G.drawImage(MyImage, 0, 0, this);
   }
}
```

Código 2.4: Criação de uma imagem a partir de uma fonte de cores reais.

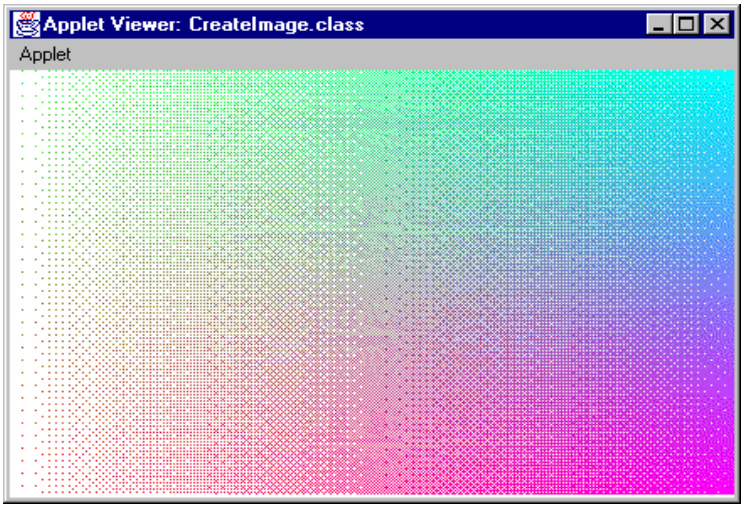

Figura 2.2: Resultado da execução da *applet* no Código 2.4.

2.3.5 Leitura de *Pixels* de uma Imagem

Na aplicação de um algoritmo para processar uma imagem, é necessário fazer a leitura dos *pixels* da imagem, uma posterior manipulação desses *pixels*, e finalmente a geração do resultado do processamento que pode ser uma nova imagem ou novas imagens, ou outro tipo de informação.

Será exposto como se faz a leitura dos *pixels* de uma imagem dando exemplo de um processamento simples, onde toma-se uma imagem como fonte, lê-se e manipula-se seus *pixels* para então gerar-se uma nova imagem modificada a partir da primeira. A leitura dos *pixels* das imagens deverá ser feita de forma análoga para os diversos tipos de processamento (algoritmos): processamento que envolve um grupo de imagens ou uma única imagem; processamento onde o resultado corresponde a uma imagem, grupo de imagens ou outro tipo de informação em PDI.

A manipulação de uma imagem para a obtenção de uma segunda poderia ser feita seguindo os seguintes passos:

- Obter a imagem que se deseja processar e esperar que seus *pixels* sejam carregados.
- Converter a imagem para um arranjo de inteiros, onde cada inteiro representa o valor de um *pixel* no formato ARGB (*alpha*, *red*, *green*, *blue*).
- Manipular o arranjo de *pixels*, de forma a obter uma imagem processada; as mudanças seriam feitas em uma cópia do arranjo da imagem.
- Construir uma imagem resultado do processamento a partir do arranjo de *pixels* alterado.

Já foram descritos os passos um e quatro; como fazer o passo três, depende do algoritmo de processamento utilizado. Será descrito como capturar os *pixels* de uma imagem em Java.

Após a imagem ter sido obtida e ter-se esperado que seus *pixels* tenham sido carregados, lê-se a altura e largura da imagem e reserva-se memória para o arranjo de *pixels* que conterá a imagem duplicada, como mostrado a seguir:

```
int ID = 0;
MediaTracker Tracker = new MediaTracker(this);
Image Lena = Toolkit.getDefaultToolkit().getImage("/tmp/lena.gif");
Tracker.addImage(Lena, ID);
try {
   Tracker.waitForID (ID);
} catch(InterruptedException E);
int ImageWidth = Lena.getWidth(this);
int ImageHeight = Lena.getHeight(this);
int Pixels[] = new int[ImageWidth * ImageHeight];
```

Posteriormente, capturam-se os *pixels* da imagem, colocando-os no arranjo de inteiros. Para isso, utiliza-se a classe java.awt.image.PixelGrabber que extrai uma região retangular específica de *pixels* do arranjo de *pixels* de uma imagem:

```
PixelGrabber PG = new PixelGrabber (Lena, 0, 0, ImageWidth, ImageHeight,
                                    Pixels, 0, ImageWidth);
try {
   PG.grabPixels();
} catch(InterruptedException E);
```

Após a captura dos *pixels* para o arranjo, utiliza-se o mesmo para fazer algum processamento. Um filtro simples para exemplificar um processamento é o filtro negativo, que inverte cada componente de cor na imagem como mostrado no trecho de código a seguir:

```
InvertedRed = 255 – CurrentRed;
InvertedGreen = 255 – CurrentGreen;
InvertedBlue = 255 – CurrentBlue;
```

Para se otimizar, o ou exclusivo (^) pode ser utilizado. Assim, a alteração dos *pixels* poderia ser feita desta forma:

```
Pixels[Index] = Pixels[Index] ^ 0xffffff ;
```

O Código 2.5 mostra a implementação completa do filtro negativo.

```
import java.awt.*;
import java.awt.image.*;

public class NegativeFilter extends java.applet.Applet {
   Image Lena, InvertedLena;
   int ImageWidth, ImageHeight;

   public void init() {
      MediaTracker Tracker = new MediaTracker(this);
      Lena = getImage(getCodeBase(), "lena.gif");
      Tracker.addImage(Lena, 0);
      try {
         Tracker.waitForID(0);
      } catch(InterruptedException E);

      ImageWidth = Lena.getWidth(this);
      ImageHeight = Lena.getHeight(this);
      int Pixels[] = new int [ImageWidth * ImageHeight];

      PixelGrabber PG = new PixelGrabber(Lena, 0, 0, ImageWidth, ImageHeight,
                                         Pixels, 0, ImageWidth);
      try { G.grabPixels(); } catch(InterruptedException E);

      for (int Index = 0; Index < ImageWidth*ImageHeight; Index++)
         Pixels[Index] = Pixels[Index]^0xffffff;
      InvertedLena = createImage(new MemoryImageSource
                              (ImageWidth, ImageHeight, Pixels, 0, ImageWidth));
   }
   public void paint(Graphics G) {
      G.drawImage(Lena, 0, 0, this);
      G.drawImage(InvertedLena, ImageWidth, 0, this);
   }
}
```

Código 2.5: *Applet* que implementa o Filtro Negativo.

A classe java.awt.image.PixelGrabber foi feita para a captura de *pixels* no modelo de cores RGB *default* (veja Seção 2.3.6) – cada *pixel* corresponde a um inteiro representando os valores A, R, G, B. Para imagens com modelos de cor indexado é necessário implementar uma classe análoga à classe PixelGrabber, por exemplo BytePixelGrabber, para se capturar os

pixels como *bytes* que correspondem a índices de uma Tabela de Mapeamento[11] que armazena as cores (inteiros) da paleta que a imagem usa.

2.3.6 Modelos de Cores de Imagens

Todo objeto da classe java.awt.Image possui um modelo de cor que especifica uma tradução dos valores de *pixels* para os componentes alpha, vermelho, verde, azul. Em Java existe a classe java.awt.image.ColorModel que é a superclasse de todas as classes que implementam um modelo de cor e estão disponíveis as classes java.awt.image.DirectColorModel e java.awt.image.IndexColorModel, subclasses da classe anterior.

A classe DirectColorModel especifica uma tradução dos valores de *pixels* para os componentes alpha, vermelho, verde e azul em que se usam os próprios *bits* do valor do *pixel*. A classe IndexColorModel especifica um modelo de cor no qual um valor de *pixel* é convertido para os componentes alpha, vermelho, verde e azul usando o valor do *pixel* como um índice de um mapa de cores.

O tipo da imagem é identificado pelo modelo de cor que ela apresenta. Imagens de cores reais possuem o modelo de cor da classe DirectColorModel. Imagens de 256 cores e de tons de cinza apresentam modelos de cor da classe IndexColorModel. Uma imagem com modelo de cor indexado pode ser colorida ou de tons de cinza; nesse caso, pode-se identificá-la como imagem de tons de cinza verificando o valor dos *pixels* no mapa de cores – todos os *pixels* do mapa apresentam valores iguais para o vermelho, o verde e o azul.

Para imagens com modelos de cor indexado, pode-se criar uma classe, por exemplo, LookUpTable que implementa um mapa de cores especificado pelo modelo de cor. A manipulação dos *pixels* da imagem seria feita através desse mapa.

Em Java, para se obter o modelo de cor de uma imagem, deve ser criada uma classe que implementa a interface java.awt.image.ImageConsumer [Espese97], [Geary97], [Gosli96c] onde é implementado um método para ajuste e outro para obtenção do modelo de cor (objeto da classe java.awt.image.ColorModel).

2.4 Conclusões

A linguagem Java apresenta classes com conjunto de funções que possibilitam a criação e manipulação de imagens de forma fácil.

Para se obter uma imagem, pode-se utilizar os métodos disponíveis para obtenção de imagens nos formatos GIF e JPEG e deve-se implementar decodificadores para a aquisição de imagens em formatos variados.

---

[11] Tabela de Mapeamento ou *Look Up Table*.

Para salvar imagens em arquivos, deve-se implementar os codificadores para os formatos desejados.

As imagens em Java são carregadas de forma assíncrona. Uma abordagem conveniente para o processamento das imagens consiste em abrir uma imagem e esperar que ela seja totalmente carregada antes de usá-la. É fácil sincronizar o carregamento das imagens utilizando um objeto da classe java.awt.MediaTracker.

No desenvolvimento de programas de PDI em Java, é conveniente usar o conjunto de funções que as classes para criação e manipulação de imagens oferecem e projetar um conjunto de classes adequado ao processamento requerido.

Capítulo 3

Plataforma para Sistemas Multimídia

3.1 Introdução

Uma plataforma (*framework*) [Cotter95], [Gibbs94] pode ser definida como uma coleção de classes abstratas relacionadas entre si, cooperando para fazer uma solução de projeto reutilizável para um dado problema.

Classes de plataformas especificam usualmente interfaces (classes abstratas), deixando em aberto a implementação. Adicionando novas classes através da especialização, as interfaces podem ser estendidas. Por exemplo, uma plataforma para sistemas multimídia [Buford94], [Gibbs94] poderia conter duas classes abstratas: Imagem e ProcessamentoDeImagens para representação da imagem e para processamento de imagens através da aplicação de algoritmos. As interfaces para essas classes podem ser especificadas sem preocupação com os detalhes de representação de uma imagem em particular ou detalhes de um processamento específico.

A plataforma PhotoPixJ foi feita com base na plataforma para sistemas multimídia descrita neste capítulo. Inicialmente será descrita a visão geral da plataforma e posteriormente serão descritos os conjuntos de classes que a compõem.

3.2 Visão Geral da Plataforma

Uma plataforma para programas multimídia compreende o que inicialmente pode parecer um grande número de classes e métodos não relacionados. Deve haver, no entanto, uma estrutura geral e as classes usualmente são divididas em três grupos: classes de mídia, classes de processamento e classes de formatos. Em síntese, classes de mídia correspondem a áudios,

vídeos, imagens e outros tipos de mídia; classes de processamento representam operações sobre as mídias de forma flexível e extensível; classes de formatos tratam das representações externas dos dados de mídia.

3.3 Classes de Mídia

Mídias são divididas em tipos: cada tipo é representado por uma classe. Essas são chamadas classes de mídia e uma possível estrutura hierárquica das mesmas pode ser vista na Figura 3.1.

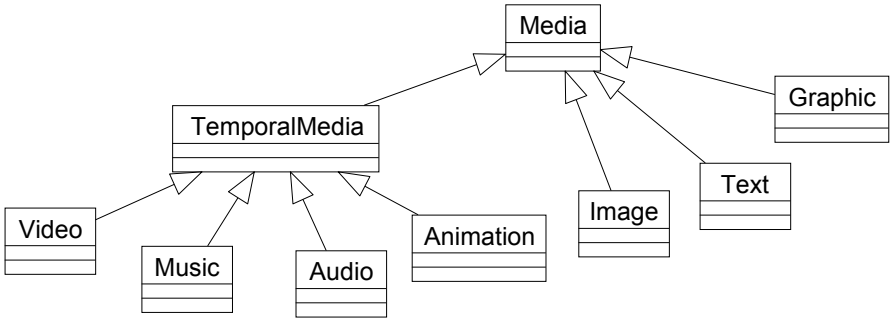

Figura 3.1: Classes de mídia.

Instâncias das classes de mídia são os objetos de mídia. Um objeto de mídia consiste em um descritor e um valor de mídia. O descritor compreende os atributos dos objetos de mídia como o seu tamanho e formato, enquanto o valor de mídia são os dados propriamente ditos (no formato legível por computador) usados para representar o objeto de mídia.

A classe de mídia mais genérica será chamada Media. Ela é uma classe abstrata que especifica métodos que deverão ser implementados por uma classe de mídia concreta e também métodos já implementados disponíveis para qualquer classe de mídia. A declaração para essa classe pode ser vista no Código 3.1.

A classe Media e suas subclasses têm um descritor, o atributo Info que armazena os atributos dos objetos de mídia, como nome, tamanho, formato. Elas têm também o atributo Selection que armazena a informação da área selecionada do objeto de mídia. Elas apresentam os métodos para ajustar e obter os atributos Info e Selection e os métodos abstratos de edição que deverão ser implementados em cada subclasse concreta da classe Media.

Tanto a classe MediaInfo quanto a classe MediaSelection são classes base na hierarquia de classes de informação sobre uma mídia e de seleção de uma mídia. Por exemplo, subclasses da MediaInfo agrupam atributos de um tipo específico de mídia.

Um exemplo de uma subclasse de Media seria a classe Image que pode ser vista no Código 3.2.

Os objetos de informação e seleção da imagem serão, analogamente, ImageInfo e ImageSelection, subclasses de MediaInfo e MediaSelection, respectivamente. Um objeto ImageInfo contém, além dos atributos genéricos do MediaInfo, atributos específicos de uma imagem como largura, comprimento, modelo de cor.

```
public abstract class Media extends Object {
   protected MediaInfo Info;
   protected MediaSelection Selection;

   public void setInfo(MediaInfo I) { Info = I; }
   public MediaInfo getInfo() { return Info; }
   public void setSelection(MediaSelection S) { Selection = S; }
   public MediaSelection getSelection() { return Selection; }

   // edição
   public abstract void cut(MediaSelection);
   public abstract Media copy(MediaSelection);
   public abstract void paste(MediaSelection, MediaContext);
}
```

Código 3.1: Classe abstrata Media.

```
abstract class Image extends Media {
   // atributos específicos de uma imagem
   // ...
   // métodos específicos de uma imagem
   // ...
}
```

Código 3.2: Classe abstrata Image.

As subclasses imediatas de Media são a classe Image, Graphic, Text e TemporalMedia. O propósito de uma subclasse da classe Media é agrupar as operações específicas daquela mídia. Portanto, a classe Image especifica métodos para imagem, enquanto a classe Text especifica métodos para texto. Essas classes são abstratas e têm outras subclasses. Por exemplo, algumas operações podem ser aplicadas somente sobre imagens de tons de cinza enquanto outras aplicam-se somente sobre imagens coloridas. Portanto, subclasses da classe Image poderiam incluir a classe GrayScaleImage e ColorImage. Indo mais adiante, a classe ColorImage poderia ter como subclasses as classes IndexedColorImage e TrueColorImage.

Em geral, cada subclasse da classe Media terá um número variado de subclasses; isso permite múltiplas representações para os diversos tipos de mídia e enriquece a variedade de operações específicas de uma mídia.

Quando se consideram as subclasses da classe Image e dos outros tipos de mídia, a hierarquia da classe Media se torna consideravelmente profunda. Estender paralelamente as classes MediaInfo, MediaSelection, MediaContext no mesmo grau pode se tornar complicado. Essas classes poderiam ser projetadas com um alto grau de generalidade de forma que, por exemplo, a classe ImageInfo poderia descrever instâncias tanto de GrayScaleImage quanto de ColorImage e qualquer outra subclasse de Image.

Ao se definir os métodos dentro da hierarquia das classes de mídia, pode não ser claro quais métodos seriam providos pela classe Media ou por uma subclasse dela. Os métodos devem ser analisados para serem disponibilizados na classe adequada dentro da hierarquia.

3.4 Classes de Processamento

Adicionar classes de mídia é uma forma de se estender a plataforma. Outra forma é através da adição de novas classes de processamento. Uma classe de processamento implementa o que pode ser chamado de "objetos funcionais" – objetos que encapsulam tanto uma função quanto os argumentos dessa função [Gibbs94]. Um objeto de uma classe de processamento é usado para armazenar os argumentos da função de processamento e existe um método na classe de processamento, o execute, que invoca a função usando os argumentos objetos de entrada, e estabelece os argumentos objetos de saída. Um exemplo simplificado pode ser visto no Código 3.3, onde existe um argumento de entrada e um argumento de saída (ambos uma imagem).

```
public class InvertFilter {
   private Image Output;

   public InvertFilter(Image Input) { ... };
   public void execute() { ... };
   public Image getOutput() { return Output; };
}
// Exemplo de uso do objeto funcional
// Abre uma imagem
Image InImg = Toolkit.getDefaultToolkit().getImage("lena.gif");
// instancia um objeto InvertFilter
InvertFilter IF = new InvertFilter(InImg);
// executa a função propriamente dita
IF.execute();
// obtém a saída da função
Image OutImg = IF.getOutput();
```

Código 3.3: Implementação de um objeto funcional.

Alguns motivos para a utilização de objetos funcionais são:

1. Não são conhecidas todas as operações quando as classes de mídia são definidas. Com objetos funcionais, novas operações podem ser adicionadas sem a necessidade de modificar as classes existentes.

2. Filtros de imagem, filtros de áudio, efeitos de vídeo e transições de vídeo são grandes em número. Ao invés de se fazer uma série de declarações de métodos em uma classe de mídia, eles podem ser representados por objetos funcionais.

3. Objetos funcionais permitem o armazenamento e processamento eficientes de "valores de mídia derivados" (um valor de mídia derivado é o valor que é calculado a partir de valores existentes). Por exemplo, separação de cores produz uma imagem a partir de outra. É muito mais eficiente armazenar o objeto funcional que especifica a separação de cores do que a própria imagem em separado.

4. Objetos funcionais são facilmente criados e executados em processos separados (*threads* separadas) o que facilita a aplicação de paralelismo e processamento distribuído.

Considerando que um processamento é feito através da aplicação de um algoritmo, classe de processamento e classe de algoritmo serão consideradas a mesma coisa.

A interface para objetos funcionais é especificada pela classe abstrata MediaAlgorithm e pode ser vista no Código 3.4.

```
public abstract class MediaAlgorithm extends Object {
   // atributos Input e Output: funcionam como argumentos da função
   protected MediaAlgorithmInput Input;
   protected MediaAlgorithmOutput Output;
   protected MediaAlgorithmParameters Parameters;

   public abstract void setParameters();
   public abstract void init();
   // a função propriamente dita
   public abstract boolean execute();
   public abstract void terminate();
   public MediaAlgorithmOutput getOutput() { return Output; }
}
```

Código 3.4: Classe abstrata MediaAlgorithm.

A classe MediaAlgorithm possui várias subclasses abstratas em que o propósito é agrupar famílias de processamentos similares como filtros de imagens, efeitos de vídeo. Essas classes têm efetivamente subclasses concretas representando processamentos (algoritmos) específicos.

Os atributos Input e Output representam os argumentos da função do objeto funcional. Input armazena os dados de entrada para o algoritmo e Output armazena os dados de saída. O atributo Parameters armazena os parâmetros, caso existam, para a execução do algoritmo; por exemplo, considerando um algoritmo para imagens, o filtro da média teria como parâmetro o tamanho da máscara. O método setParameters estabelece os valores dos parâmetros, requisitando-os do usuário, por exemplo. O método execute invoca a função propriamente dita e determina os dados de saída. Os métodos init e terminate são invocados no início e final do método execute e servem para efetuarem operações de inicialização e finalização, respectivamente.

Além do que foi apresentado no Código 3.4, todo algoritmo deverá ter um atributo estático Info que representará informações sobre o algoritmo como quais objetos de entrada para o algoritmo são válidos e quais são os objetos de saída correspondentes. Todo algoritmo deverá ter também dois métodos estáticos: um define o objeto Info (setInfo) e o outro retorna esse objeto (getInfo). A definição do atributo e dos métodos estáticos estão no Código 3.5.

```
private static MediaAlgorithmInfo Info; // armazena informações sobre o algoritmo

public static void setInfo() { … }
public static MediaAlgorithmInfo getInfo() { return Info; }
```

Código 3.5: Interface dos atributos e métodos estáticos de toda classe concreta de processamento.

Seguindo a hierarquia de classes de algoritmos, poderia ser definida, para processamento de imagens, a classe abstrata ImageAlgorithm (e as classes ImageAlgorithmInfo, ImageAlgorithmInput, ImageAlgorithmOutput, ImageAlgorithmParameters correspondentes). Um algoritmo específico de processamento de imagens, por exemplo o Filtro da Média, seria uma subclasse concreta da classe ImageAlgorithm. Esse exemplo pode ser visto no Código 3.6.

```
abstract class ImageAlgorithm extends MediaAlgorithm { ... }

public class AverageFilter extends ImageAlgorithm {
   private static ImageAlgorithmInfo Info;

   public AverageFilter(ImageAlgorithmInput Input);
   // implementação dos métodos abstratos
   public void setParameters() { ... };
   public void init() { ... };
   public boolean execute() { ... };
   public void terminate() { ... };
   // métodos estáticos
   public static void setInfo() { ... }
   public static ImageAlgorithmInfo getInfo() { return Info; }
}
```

Código 3.6: Classe concreta de processamento de imagem: classe AverageFilter.

3.5 Classes de Formato

Representações externas de valores de mídia são chamadas formatos de mídia [Kay92], [Murray94]. Há duas categorias básicas: formatos de arquivos, usados para armazenar valores de mídia, e formatos de cadeias (*stream formats*), usados para enviar valores de mídia através de linhas de comunicação (*communication links*).

São necessários métodos para decodificar diferentes formatos de mídia e representá-los no valor de mídia, e métodos para codificar o valor de mídia para diferentes formatos (considerando formatos que sejam compatíveis com o valor de mídia). Por exemplo, dois

formatos comuns de imagens são o TIFF[12] e o GIF [Kay92], [Murray94]. A classe de imagem Image poderia ser declarada como:

```
abstract class Image extends Media {
   // métodos de decodificação e codificação
   public static Image DecodeTIFF (String FileName);
   public static Image DecodeGIF (String FileName);
   public void EncodeTIFF (String FileName);
   public void EncodeGIF (String FileName);
   // ...
}
```

E o seguinte código decodificaria uma imagem GIF e a codificaria para o formato TIFF:

```
Img = Image.DecodeGIF (”NomeDoArquivoDeImagem.gif”);
Img.EncodeTIFF (”NomeDoArquivoDeImagem.tif”);
```

Outra alternativa seria definir métodos gerais tomando o tipo do formato como argumento:

```
abstract class Image extends Media {
   // métodos genéricos de decodificação e codificação
   public static Image Decode (String FileName, FormatType T);
   public void Encode (String FileName, FormatType T);
   //...
}
```

Ambas as soluções apresentam problemas. Não são conhecidos todos os formatos quando as classes de mídia são definidas, e formatos adicionais provavelmente serão necessários em extensões futuras da aplicação. Suportar novos formatos ou novas versões de formatos requer a adição de novos métodos ou a modificação de métodos existentes, o que implica na modificação das classes e torna mais difícil a integração e manutenção de *software*.

Uma solução melhor consiste em implementar classes de formatos. Cada formato é representado por uma classe que implementa métodos de importação (decodificação) e exportação (codificação).

Classes de formato são organizadas em uma hierarquia com estrutura similar à hierarquia das classes de mídia. A raiz da hierarquia será a classe abstrata MediaFormat; ela tem subclasses, também abstratas, para cada tipo principal de formato de mídia. Os formatos reais são subclasses concretas aparecendo como folhas na hierarquia. Um exemplo de hierarquia pode ser visto na Figura 3.2.

---

[12] TIFF: *Tagged Image File Format*.

A declaração para a classe MediaFormat pode ser vista no Código 3.7. O método canDecode verifica se o objeto da classe de formato pode importar um arquivo específico (ou uma cadeia de *bytes*). O método decode cria uma instância de alguma subclasse concreta de Media e a inicializa usando os dados de um arquivo (ou de uma cadeia de *bytes*). O método canEncode verifica a compatibilidade entre o objeto de mídia e o formato de mídia. O método encode escreve o dado associado com o objeto de mídia para um arquivo (ou para uma cadeia de *bytes*).

Toda subclasse concreta de uma classe de formato deverá implementar os métodos abstratos da(s) superclasse(s) e também os métodos estáticos que podem ser vistos no Código 3.8 e no Código 3.9.

O método estático guessFormat tenta identificar o tipo de formato de um arquivo específico (ou de uma cadeia de *bytes*). Ele deve ser implementado em uma classe de formato abstrata de um tipo principal de mídia, por exemplo na classe ImageFormat no Código 3.8.

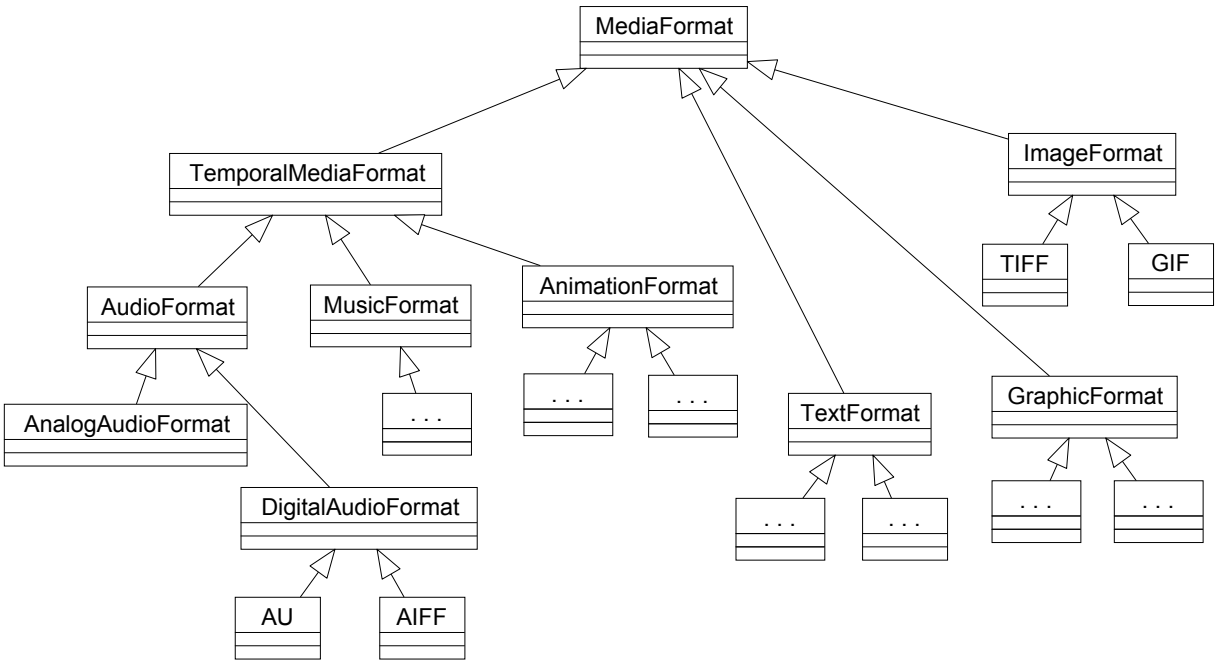

Figura 3.2: Classes de formatos de mídia.

```
public abstract class MediaFormat {
   // Decodificação e codificação
   public abstract boolean canDecode (String FileName);
   public abstract boolean canDecode (byte [] Stream);
   public abstract Media decode (String FileName);
   public abstract Media decode (byte [] Stream);
   public abstract boolean canEncode (Media Value);
   public abstract void encode (String FileName, Media Value);
   public abstract void encode (byte [] Stream, Media Value);
}
```

Código 3.7: Classe abstrata MediaFormat.

```
public class ImageFormat extends MediaFormat {
   public static MediaFormat guessFormat(String FileName) {...}
   public static MediaFormat guessFormat(byte [] Stream) {...}
   // ...
}
```

Código 3.8: Classe de formato abstrata de um tipo principal de mídia: classe ImageFormat.

No Código 3.9, o atributo estático Info é um objeto que apresenta atributos que descrevem o formato; os métodos estáticos getInfo e setInfo retornam e determinam o valor de Info, respectivamente. O método estático isFormatOf verifica se um arquivo (ou cadeia de *bytes*) usa o formato representado na classe de formato específica.

```
public class Gif extends ImageFormat {
   private static FormatInfo Info;
   // ...
   public static boolean isFormatOf(String FileName) {...}
   public static boolean isFormatOf(byte [] Stream) {...}
   public static void setInfo() {...}
   public static FormatInfo getInfo() { return Info; }
   // ...
}
```

Código 3.9: Classe concreta de formato de mídia: classe Gif.

3.6 Conclusões

A plataforma para as classes de mídia permite múltiplas representações para os diversos tipos de mídia e enriquece a variedade de operações específicas de uma determinada mídia, além de proporcionar forma fácil de acréscimo de novos tipos de mídia.

O uso de objetos funcionais na implementação de algoritmos de sistemas multimídia é uma solução adequada para prover fácil extensão do sistema através da adição de novos processamentos (algoritmos).

O uso de classes de formato se apresenta como solução adequada para implementação de decodificação de formatos de mídia para valores de mídia e codificação de valores de mídia para formatos de mídia. A solução adotada facilita a extensão da aplicação através do acréscimo de novos formatos.

A plataforma para sistemas multimídia apresentada pode ser usada para um sistema de processamento de uma mídia específica, como um sistema de processamento de imagens, ou para um sistema de processamento de mídias variadas como de imagens, textos, vídeos, animação, música.

Capítulo 4

O Sistema PhotoPixJ

4.1 Introdução

O objetivo principal deste trabalho foi construir um sistema independente de plataforma para execução de algoritmos de processamento digital de imagens, e que apresente um conjunto de classes que proporcione fácil implementação e integração de novos algoritmos.

Para que o sistema tivesse boa qualidade ele foi concebido a partir de um projeto orientado para objetos com base no uso de uma plataforma para sistemas multimídia (Capítulo 3).

No projeto, foram identificados pacotes lógicos e as classes que compõem esses pacotes.

Este capítulo descreve a especificação, projeto e implementação do PhotoPixJ. São detalhados a sua estrutura interna que compreende a estrutura de representação e manipulação das imagens no sistema, a estrutura de algoritmos de processamento e a estrutura de formatos de imagens. São apresentados também a interface com o usuário, o sistema de ajuda e os ambientes de funcionamento do sistema.

4.2 Especificação do Sistema

Foi feita uma especificação do sistema com base em três tipos de requisitos básicos: os tipos de imagens e os formatos externos de imagens com os quais o sistema trabalhará, e o conjunto de funções de interface com o usuário disponíveis.

4.2.1 Tipos de Imagens e Formatos Externos de Imagens

O sistema deverá trabalhar com três tipos de imagens inicialmente:

- Imagens em tons de cinza, 8 *bits* por *pixel*;

- Imagens de 256 cores, 8 *bits* por *pixel*;
- Imagens de cores reais, 24 *bits* por *pixel*.

O sistema deverá trabalhar com formatos externos de imagens compatíveis com cada tipo de imagem do sistema. O sistema trabalhará, inicialmente, com os formatos GIF (para imagens de tons de cinza e de 256 cores), e JPEG e TIFF (para imagens de cores reais).

4.2.2 Funções do Sistema

- Carregamento de imagens no formato GIF e JPEG;
- Salvamento de imagens no formato GIF e TIFF (somente versão aplicação do sistema);
- Desistência de uma ação e repetição de uma ação anteriormente desfeita sobre as imagens do sistema;
- Visualização de Histogramas, Perfis de Linha e de Coluna de imagens;
- Conversão de imagens para os tipos de valores de imagens existentes no sistema:
  - imagens de 256 cores para imagens de cores reais;
  - imagens de 256 cores para imagens em tons de cinza;
  - imagens de cores reais para imagens de tons de cinza;
  - imagens de cores reais para imagens de 256 cores (aplicação de algoritmos de quantização);
- Visualização de informação sobre uma imagem:
  - Nome (identificação única da imagem durante execução do sistema);
  - Dimensão;
  - *Bits* por *pixel*;
  - Tipo;
- Disponibilidade de um sistema de auxílio do sistema e dos algoritmos do sistema;
- Acesso para execução dos algoritmos de PDI de forma fácil e rápida.

4.3 Pacotes Lógicos do Sistema

No projeto de um sistema orientado para objetos, são identificados pacotes lógicos, cada um agrupando classes correlatas. Conjuntos de classes candidatos a pacotes lógicos podem ser: classes de interface com o usuário, classes de acesso a uma base de dados, classes de estruturas de dados, classes correspondentes a grandes grupos funcionais.

A partir da plataforma do sistema, pode-se identificar três pacotes: pacote com classes para representação e manipulação de valores de mídia, pacote com classes de processamento de mídia, pacote com classes de formatos de mídia. Outro pacote que pode ser identificado é o de implementação da interface com o usuário.

No PhotoPixJ, foram identificados os pacotes que podem ser vistos na Figura 4.1. O pacote ppixj.media apresenta classes para representação e manipulação de uma mídia genérica. As classes abstratas Media, MediaAlgorithm, MediaFormat se encontram nesse pacote. O pacote ppixj.media.image apresenta classes para representação e manipulação de imagens. Analogamente ao pacote ppixj.media, as classes abstratas Image, ImageAlgorithm, ImageFormat se encontram no pacote ppixj.media.image. As classes de processamento específico de uma imagem (subclasses de ImageAlgorithm), que estarão disponíveis através da interface com o usuário, pertencerão ao pacote ppixj.media.image.algorithms. O pacote ppixj.media.image.algorithm apresenta classes de gerência e controle de execução de algoritmos de imagens. As classes de formatos específicos de uma imagem (subclasses de ImageFormat) pertencerão ao pacote ppixj.media.image.formats. O pacote ppixj.ui contém as classes de implementação da interface com o usuário. O pacote ppixj.util contém classes de utilidade geral como de estruturas de dados não específicas usadas no sistema.

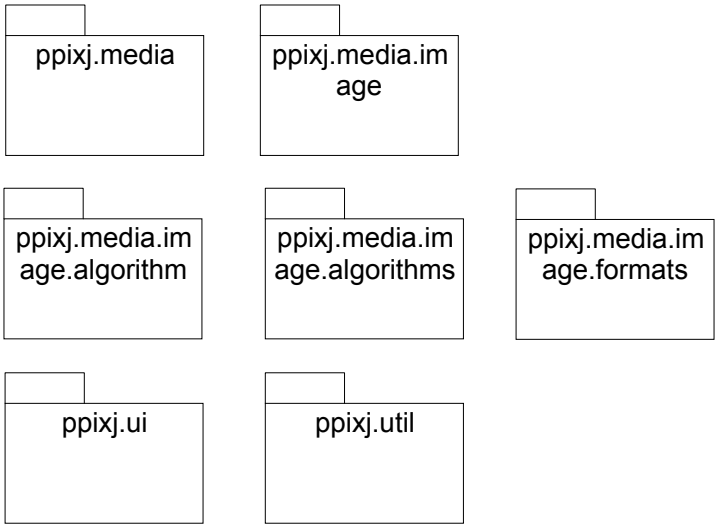

Figura 4.1: Pacotes lógicos do PhotoPixJ.

4.4 Imagens

As imagens são a matéria-prima do processamento digital de imagens. Portanto, uma imagem constitui uma classe básica para qualquer sistema de PDI. Java apresenta em sua API a classe Image (pacote java.awt) e o pacote java.awt.image para representação e manipulação de imagens [Espese97], [Geary97]. No entanto, foi necessário criar uma classe Image específica para o PhotoPixJ para a implementação da plataforma proposta para o sistema.

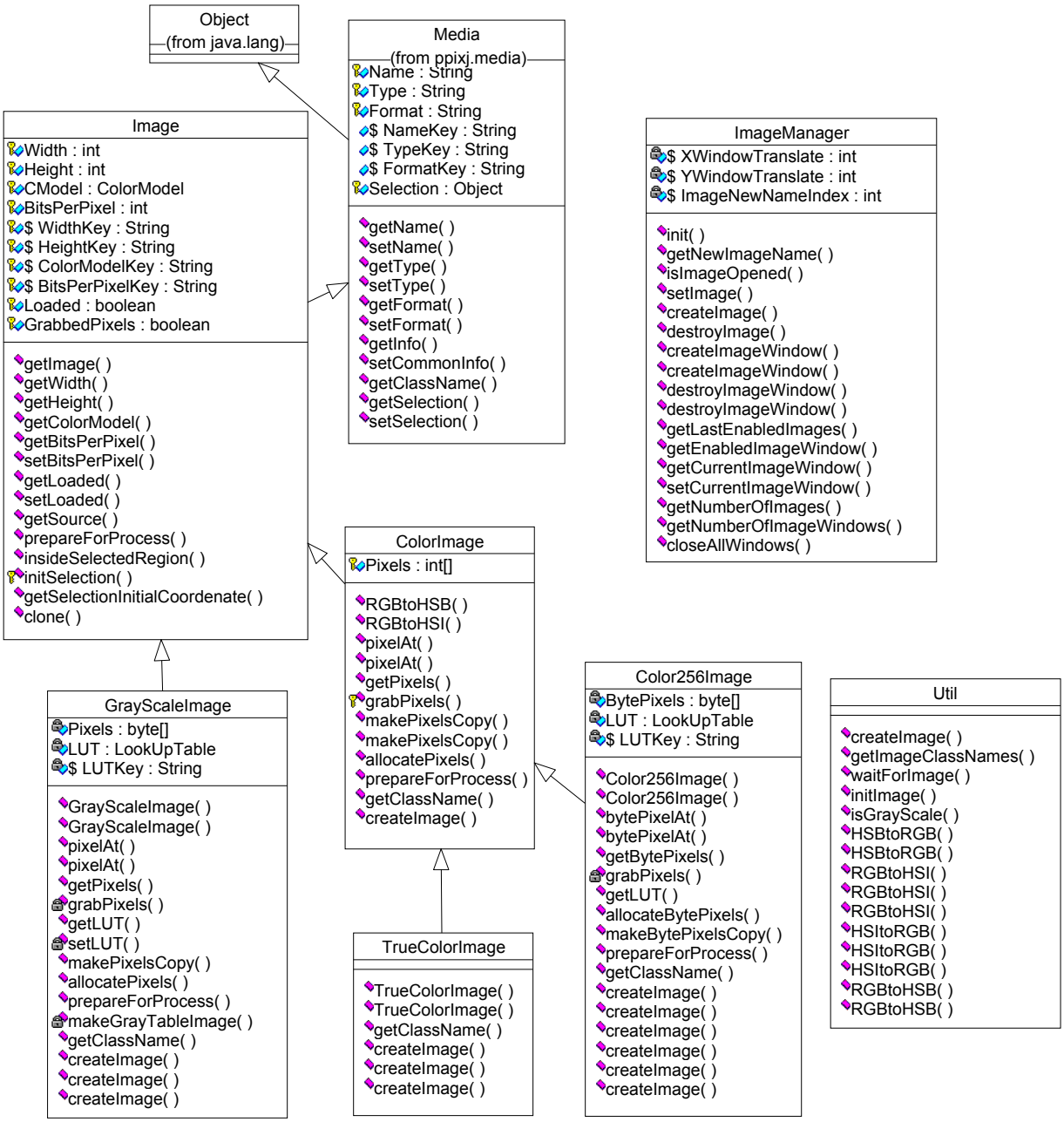

Figura 4.2: Principais classes do pacote ppixj.media.image.

Seguindo a especificação da plataforma, foi construído o conjunto de classes para representação e manipulação de imagens. Todas essas classes formam um pacote, o ppixj.media.image. As principais classes desse pacote podem ser vistas no diagrama da Figura 4.2. Essas classes serão descritas a seguir.

Classe Image: é uma classe abstrata que apresenta atributos e métodos específicos de uma imagem. Ela apresenta um atributo, Img, que constitui um objeto da classe java.awt.Image da API de Java.

Classe GrayScaleImage: é uma classe concreta, subclasse de Image, que representa uma imagem em tons de cinza.

Classe ColorImage: é uma classe abstrata que representa uma imagem colorida.

Classe TrueColorImage: é uma classe concreta, subclasse de ColorImage, que representa uma imagem de cores reais.

Classe Color256Image: é uma classe concreta, subclasse de ColorImage, que representa uma imagem de cores indexadas, especificamente de 256 cores.

Classe ImageManager: classe com métodos e atributos estáticos para o controle de criação e destruição de imagens e para a gerência geral associada às imagens no sistema.

Classe Util: classe que coloca disponível métodos genéricos para o processamento de imagens.

A classe Image do pacote ppixj.media.image é a superclasse de todas as classes de imagens no sistema. No PhotoPixJ, pode-se trabalhar com imagens em tons de cinza, imagens de 256 cores e imagens de cores reais. Para que o sistema possa trabalhar com outros tipos de imagens, é necessário que se acrescente o novo tipo que será uma subclasse concreta na hierarquia de classes de imagens do sistema.

4.5 Algoritmos de Processamento de Imagens

No projeto de interface dos algoritmos, é essencial prover forma fácil e robusta de integração de novas operações para proporcionar a extensão do sistema. A melhor forma considerada para proporcionar a fácil inserção de novos algoritmos em um sistema multimídia consiste na implementação de objetos funcionais (veja Capítulo 3).

A plataforma para processamento de dados multimídia constitui a interface para implementação de algoritmos no PhotoPixJ. A relação das classes pode ser vista na Figura 4.3.

Todo algoritmo de processamento de imagens do sistema será uma classe concreta, subclasse de ImageAlgorithm, e pertencerá ao pacote ppixj.media.image.algorithms. O procedimento para a adição de um algoritmo no sistema pode ser visto em detalhes no Capítulo 5.

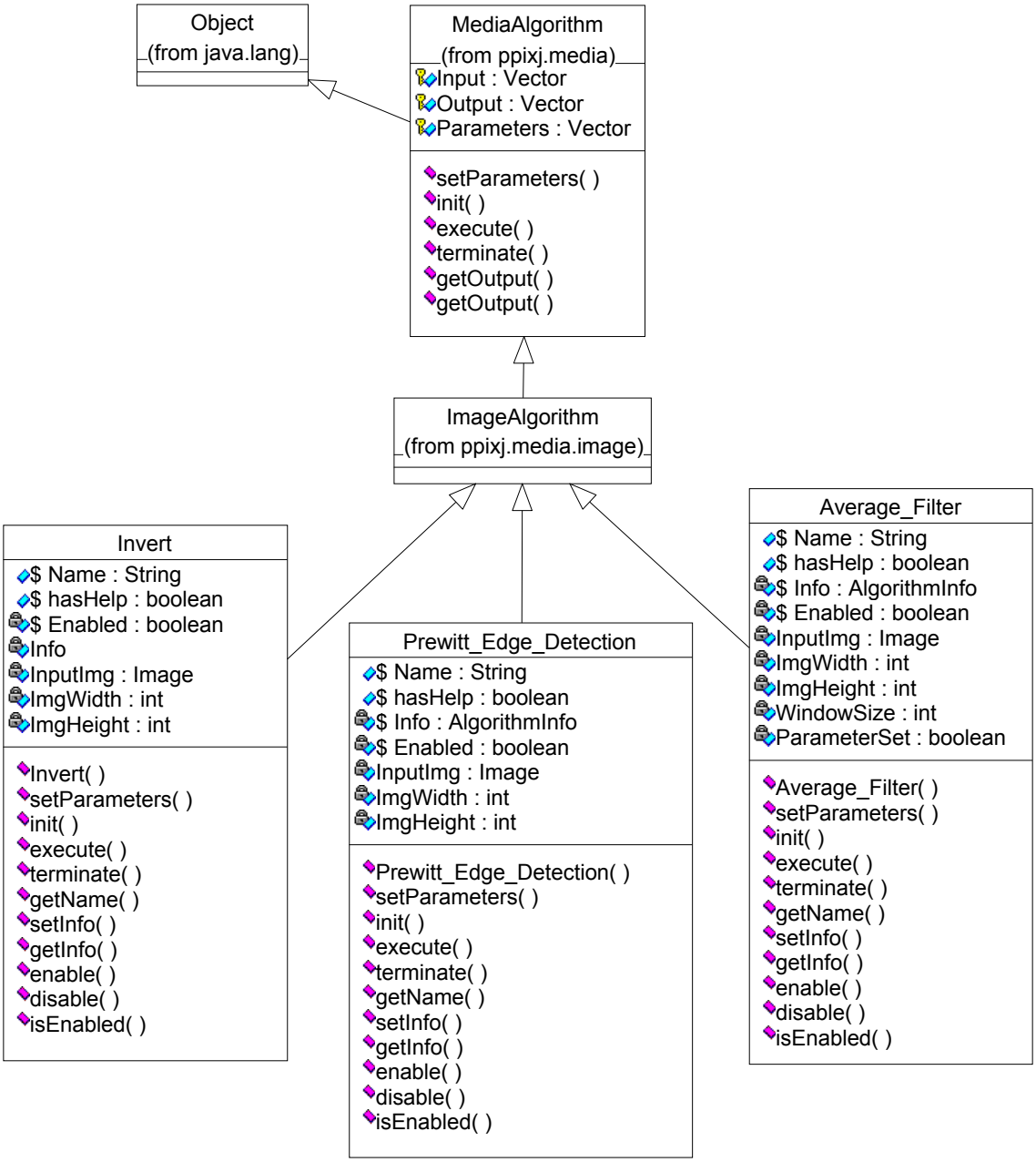

Figura 4.3: Relação das classes de processamento no PhotoPixJ.

4.5.1 Interface com o Usuário para os Algoritmos

A interface com o usuário para novos algoritmos constitui um fator a ser estudado. Cada algoritmo pode exigir parâmetros diferentes uns dos outros a serem fornecidos pelo usuário, e portanto exige interfaces com o usuário diferentes.

Uma solução na implementação de interfaces com o usuário para captura de parâmetros de execução de algoritmos seria projetar, conceitualmente, o sistema como um todo e prever todas as operações e recursos que existirão. Isso permitirá estabelecer de antemão como será a interface com o usuário para a utilização de cada uma das operações. Se no projeto do sistema

for suposto que todas as operações possíveis foram previstas e só algoritmos que implementem aqueles tipos de operações poderão ser adicionados, fatalmente o sistema não estará bem projetado. Não é possível prever todos os tipos de operações com 100% de segurança, portanto, pode-se até definir a interface para as operações previstas no projeto, mas é essencial que se proveja solução fácil para a adição de novos tipos de operações.

O indivíduo que acrescenta um algoritmo dispõe das interfaces disponíveis e, caso um novo algoritmo exija uma interface específica, ele mesmo a projeta, a implementa e a acrescenta ao sistema juntamente com o algoritmo. As interfaces com o usuário disponíveis no sistema podem ser vistas no Apêndice A.

O uso da linguagem Java possibilita que a mesma versão de codificação de interface com o usuário seja executada em diversas plataformas.

4.6 Formatos de Imagens

Toda classe de formato pertencerá ao ppixj.media.image.formats. A relação das classes do pacote pode ser vista na Figura 4.4.

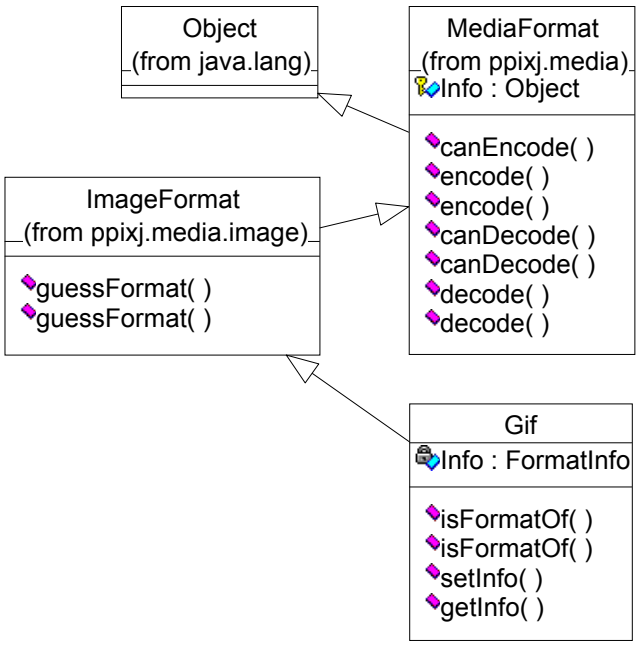

Figura 4.4: Relação das classes de formato no PhotoPixJ.

Toda classe de formato de imagens será uma subclasse de ImageFormat. No PhotoPixJ, existe atualmente, implementado e integrado ao sistema, o formato GIF [Kay92], [Murray94].

Na figura, na classe Gif, foram usados vários atributos e foram implementados vários métodos específicos para a implementação do formato que não estão indicados. Além disso, outras classes úteis na implementação do formato foram criadas.

Como cada formato poderá necessitar da implementação de outras classes, uma solução adequada seria criar um pacote dentro do pacote ppixj.media.image.formats para cada formato implementado no sistema. Por exemplo, a classe de formato Gif e todas as classes implementadas específicas para a implementação desse formato pertenceriam ao pacote ppixj.media.image.formats.gif.

Novos formatos, assim como novos algoritmos de processamento, são facilmente integrados ao PhotoPixJ.

4.7 Interface com o Usuário

O primeiro objetivo no projeto e construção de uma interface com o usuário é prover devidamente o conjunto de funções requisitado. Um segundo aspecto consiste em prover confiabilidade, disponibilidade, segurança e integridade: os comandos devem funcionar como especificados, dados mostrados devem refletir o conteúdo da base de dados e atualizações devem ser aplicadas corretamente. Uma terceiro aspecto se refere à padronização, integração, consistência e portabilidade. Outro aspecto importante é o planejamento com base no tempo que se tem para concluir o projeto [Shneid92] .

A interface com o usuário, assim como os demais aspectos do sistema, foi feita com base em um projeto orientado para objetos [Blattn92], [Collin95]. Isso permite que o conjunto de funções de interface seja facilmente estendido.

4.7.1 Janela Principal

A janela principal do PhotoPixJ (Figura 4.5) apresenta um menu e um painel de visualização em árvore, para acesso aos algoritmos de processamento.

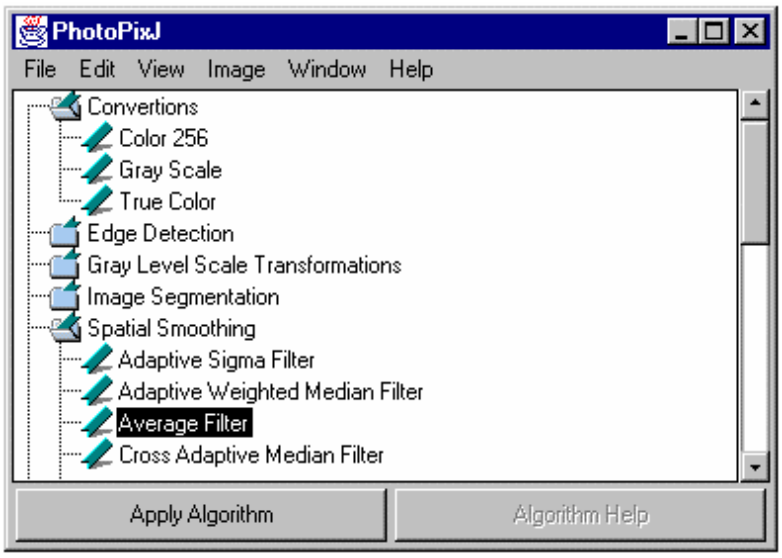

Figura 4.5: Janela Principal do PhotoPixJ.

Menus

A hierarquia de menus da janela principal é apresentada na Figura 4.6.

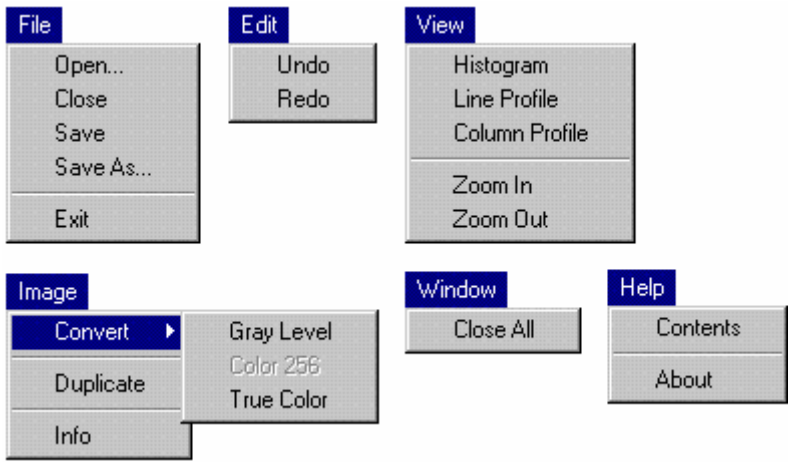

Figura 4.6: Menus do PhotoPixJ.

O Menu **File** (Arquivo) provê operações para transferências de imagens entre a memória do sistema e o ambiente externo. Ele não está presente na versão *applet*, e no lugar dele existe o menu **Applet** com operações para carregamento e fechamento de imagens (sem operações de salvamento). O menu **Applet** pode ser visto na Figura 4.7.

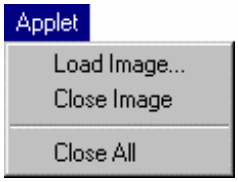

Figura 4.7: Menu **Applet** da versão *applet* do PhotoPixJ. Substitui o menu **File** da versão aplicação.

O menu **Edit** (Editar) permite a desistência de uma ação ou a repetição da ação que foi desfeita anteriormente, sobre as imagens no sistema.

O menu **View** (Visualizar) provê comandos para apresentação das imagens nas suas janelas de forma variada (ampliação e redução na visualização das imagens). Além disso, provê comandos para visualização de Histogramas, Perfis de Linha e de Coluna de Imagens.

O menu **Image** (Imagem) provê comandos para conversão de imagens para os tipos de valores de imagem existentes no sistema. Provê comando para duplicação de uma imagem e comando para visualização de informação sobre uma imagem.

O menu **Window** (Janela) provê comandos para controle de visualização das janelas de imagens.

O menu **Help** (Ajuda) destina-se à apresentação de ajuda *on-line* ao usuário do sistema. A ajuda inclui informações sobre o conteúdo do sistema e sobre funcionamento e descrição dos algoritmos.

4.7.2 Visualização dos Algoritmos

Como o sistema poderá ter um número grande de algoritmos classificados dentro de categorias de forma flexível em uma hierarquia, é importante obter uma solução de interface com o usuário em que seja fácil ter acesso aos algoritmos. Além disso, na introdução de novos algoritmos é desejável que o implementador não se preocupe com a inserção de elementos de interface para acesso ao seu algoritmo. Optou-se por uma estrutura de visualização em árvore que representa os algoritmos na sua estrutura hierárquica (Figura 4.5).

A solução adotada é adequada para que se obtenha acesso fácil aos algoritmos quando estes estão em uma hierarquia grande ou profunda.

Tipos de Algoritmos

Foram classificados dois tipos básicos de algoritmos no PhotoPixJ: algoritmos que operam sobre uma única imagem de entrada e algoritmos que operam sobre mais de uma imagem de entrada.

Para os algoritmos que operam sobre uma imagem de entrada, a execução é feita com base na janela de imagem ativa no momento da aplicação do algoritmo. Para os algoritmos que operam sobre mais de uma imagem de entrada (por exemplo, algoritmos de Aritmética de Imagens), uma caixa de diálogo (Figura 4.8) é apresentada para que as imagens de entrada sejam selecionadas (a partir das janelas de imagens presentes no momento), antes que o algoritmo possa ser executado.

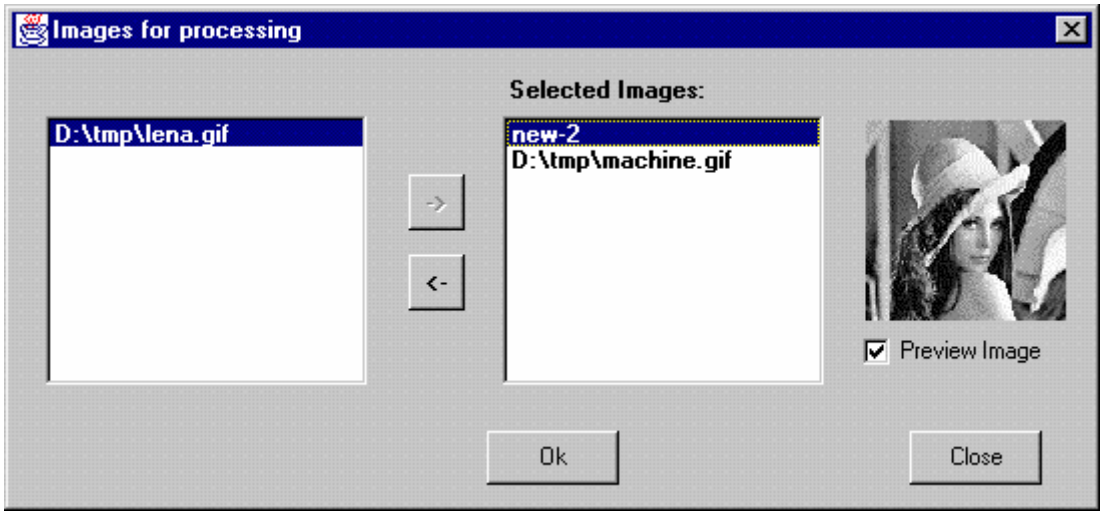

Figura 4.8: Caixa de diálogo para seleção das imagens a serem utilizadas na execução de um algoritmo que opera sobre mais de uma imagem de entrada.

4.7.3 Janelas de Imagem

As imagens, no sistema, são visualizadas em uma janela. Um exemplo pode ser visto na Figura 4.9.

Figura 4.9: Janela de Imagem do PhotoPixJ.

4.7.4 Outras Funções de Interface

O sistema apresenta outros recursos de interface como janelas de visualização de informação sobre imagens, de Histogramas e de Perfis de Linha e de Coluna.

Uma janela de informação sobre uma imagem de 256 cores pode ser vista na Figura 4.10. São fornecidas informações sobre a imagem e sobre a configuração atual do vídeo.

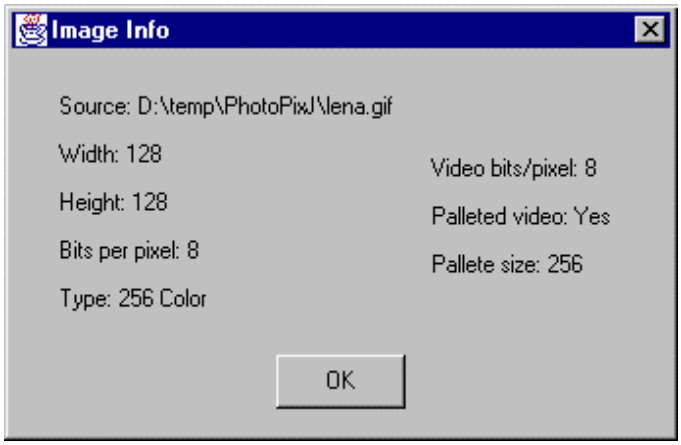

Figura 4.10: Janela de visualização de informação sobre uma imagem.

Um exemplo de janela de visualização de Histograma para uma imagem colorida pode ser visto na Figura 4.11.

Janelas de visualização do Perfil de Linha e do Perfil de Coluna de uma imagem podem ser vistas na Figura 4.12. Na Figura 4.12, estão sendo mostrados o perfil da linha 23 e o perfil da coluna 127 de uma imagem em tons de cinza com dimensão 128x128 *pixels*.

Tanto os Histogramas como os Perfis de Linha e de Coluna podem ser visualizados para imagens em tons de cinza e para imagens coloridas (de 256 cores e de cores reais).

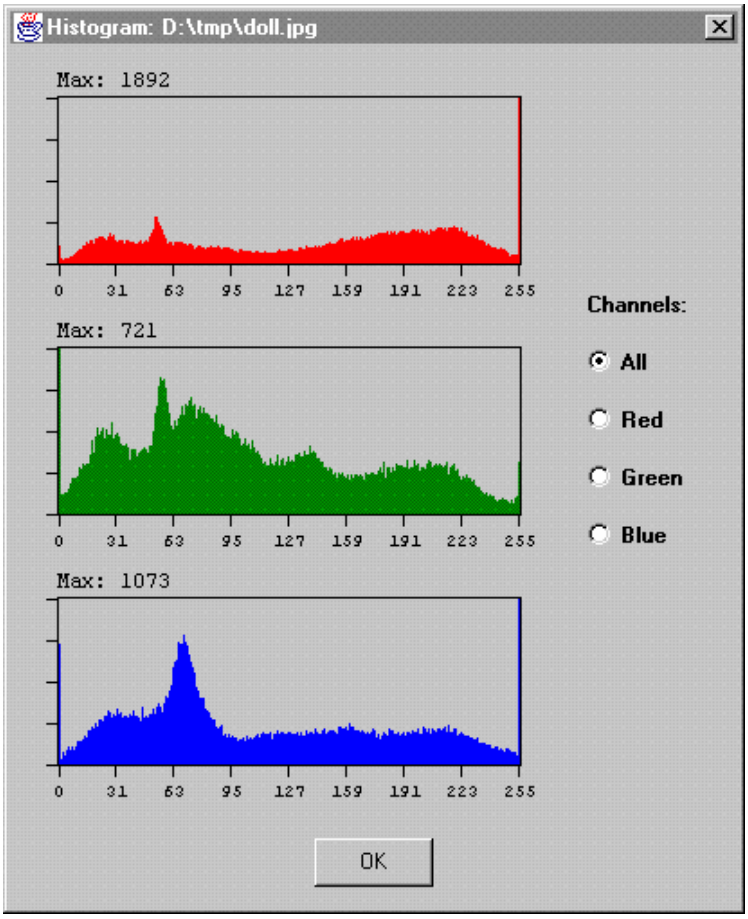

Figura 4.11: Janela de visualização de Histograma para uma imagem colorida.

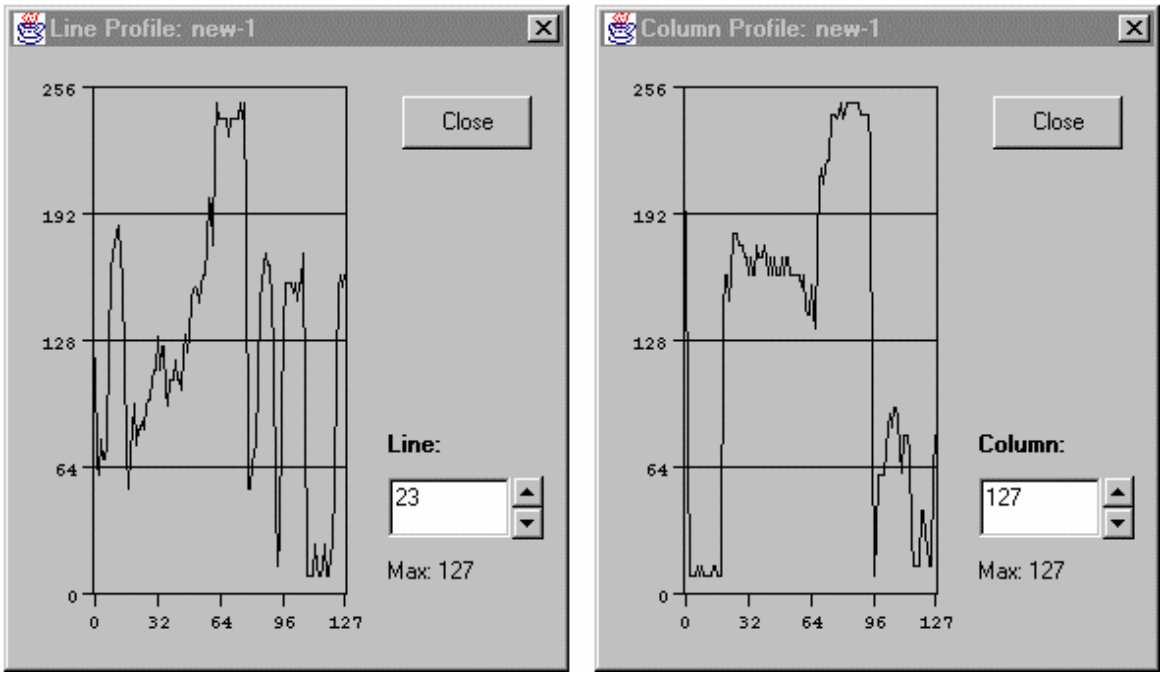

Figura 4.12: Janelas de visualização de Perfis de Linha e de Coluna para uma imagem em tons de cinza.

4.8 Arquivos de Ajuda

O PhotoPixJ possui um sistema de auxílio na utilização de suas funções. É possível também obter uma descrição do funcionamento dos algoritmos.

No sistema de ajuda, é desejável que seja fácil a sua implementação bem como a sua manutenção, através da mudança do conteúdo dos arquivos de ajuda e da adição de auxílio para novas funções do sistema, como novos algoritmos. Além disso, é desejável que a solução usada não dificulte a portabilidade do sistema. Com essa diretiva, optou-se por implementar a ajuda no sistema usando arquivos no formato HTML[13] [HTML97] associado a um navegador para acesso aos mesmos.

Para a versão aplicação, existe um arquivo de configuração do sistema de ajuda, o help.inf, que indica a localização do arquivo executável do navegador a ser usado.

Todos os arquivos de ajuda ficam sob um subdiretório dentro do diretório raiz do sistema, o subdiretório help.

4.9 Ambientes de Funcionamento

O PhotoPixJ foi implementado em Java 1.0.2 e testado no Windows® 95 e no Solaris™ versão 2.5 no ambiente CDE (*Common Desktop Environment*) versão 1.0.2. O sistema funciona nos ambientes testados e não apresentou problemas maiores durante o desenvolvimento. Alguns aspectos de interface gráfica funcionam de forma diferente de uma plataforma para outra (devido às diferenças nas soluções em cada ambiente) o que requer que alguma mudança seja feita para que o mesmo conjunto de funções seja satisfeito nos diferentes ambientes.

A API de Java 1.0.2 apresenta alguns problemas de funcionamento nas diferentes plataformas. É desejável que, à medida em que esses problemas sejam solucionados em novas *releases* ou novas versões da linguagem, se faça uma atualização do sistema para que se consiga a correção de problemas e novos recursos possam ser utilizados.

Java, como linguagem interpretada (ou semi-interpretada), faz com que o sistema apresente tempos de execução altos nos algoritmos de processamento de imagens. Uma solução para isso é o uso de um compilador JIT [Kramer96], [SunJIT]. Ele traduz o código em *bytecode* para o código específico da máquina antes do programa ser executado. Com um compilador JIT a eficiência de programas na linguagem Java se equipara a de programas em linguagens como C++.

---

[13] HTML: *HyperText Markup Language.*

4.10 Conclusões

A plataforma PhotoPixJ se apresentou adequada para prover a fácil adição de novas classes de mídia, novas classes de algoritmos (uso de objetos funcionais) e novos formatos de mídia.

A solução de interface com o usuário adotada se mostrou bastante adequada para o acesso fácil aos algoritmos.

O uso de Java apresentou vantagens para a implementação do PhotoPixJ. Java, sendo uma linguagem independente de plataforma, faz com que o sistema seja altamente portátil. Para a inserção de novas operações, só é necessário usar a API de Java, tanto para o código relativo aos algoritmos como para o código relativo à interface com o usuário, que eventualmente deverá ser implementada. Portanto, para que o sistema se mantenha multiplataforma, com a adição de uma nova função, é necessário implementar apenas uma versão de código.

Java sendo uma linguagem interpretada, produz programas bem mais lentos do que programas em uma linguagem como C. Isso pode ser crítico quando se trata de processamento de imagens e de mídias em geral. Devem ser usados compiladores JIT para se obter execução eficiente desses programas.

Capítulo 5

Adição de Algoritmos de PDI no PhotoPixJ

5.1 Introdução

O objetivo neste trabalho foi que o PhotoPixJ servisse como plataforma para adição de novos algoritmos de PDI. É importante que esteja disponível, para os eventuais implementadores de algoritmos, uma descrição do processo de adição de um algoritmo no sistema. Este capítulo apresenta o procedimento de inserção de um algoritmo de PDI no PhotoPixJ.

Este capítulo apresenta também uma análise das experiências obtidas com a adição de algoritmos no sistema.

5.2 Procedimento Geral

Um diagrama com os passos do processo de inserção de um novo algoritmo no sistema pode ser visto na Figura 5.1.

Para se adicionar um algoritmo ao sistema, deverá ser implementada uma nova classe de algoritmo, subclasse de ImageAlgorithm. Para explicar o processo de inserção de um algoritmo será utilizado um exemplo: a inserção do algoritmo do Filtro da Média.

A nova classe a ser implementada será a Average_Filter. Ela será inserida na categoria dos filtros de Suavização Espacial (*Spatial Smoothing*) e portanto deverá pertencer ao pacote ppixj.media.algorithms.Spatial_Smoothing.

Ao se implementar a nova classe de algoritmo, deverão ser implementados todos os métodos abstratos das superclasses (incluindo métodos estáticos, que em Java 1.0.2 não podem ser abstratos).

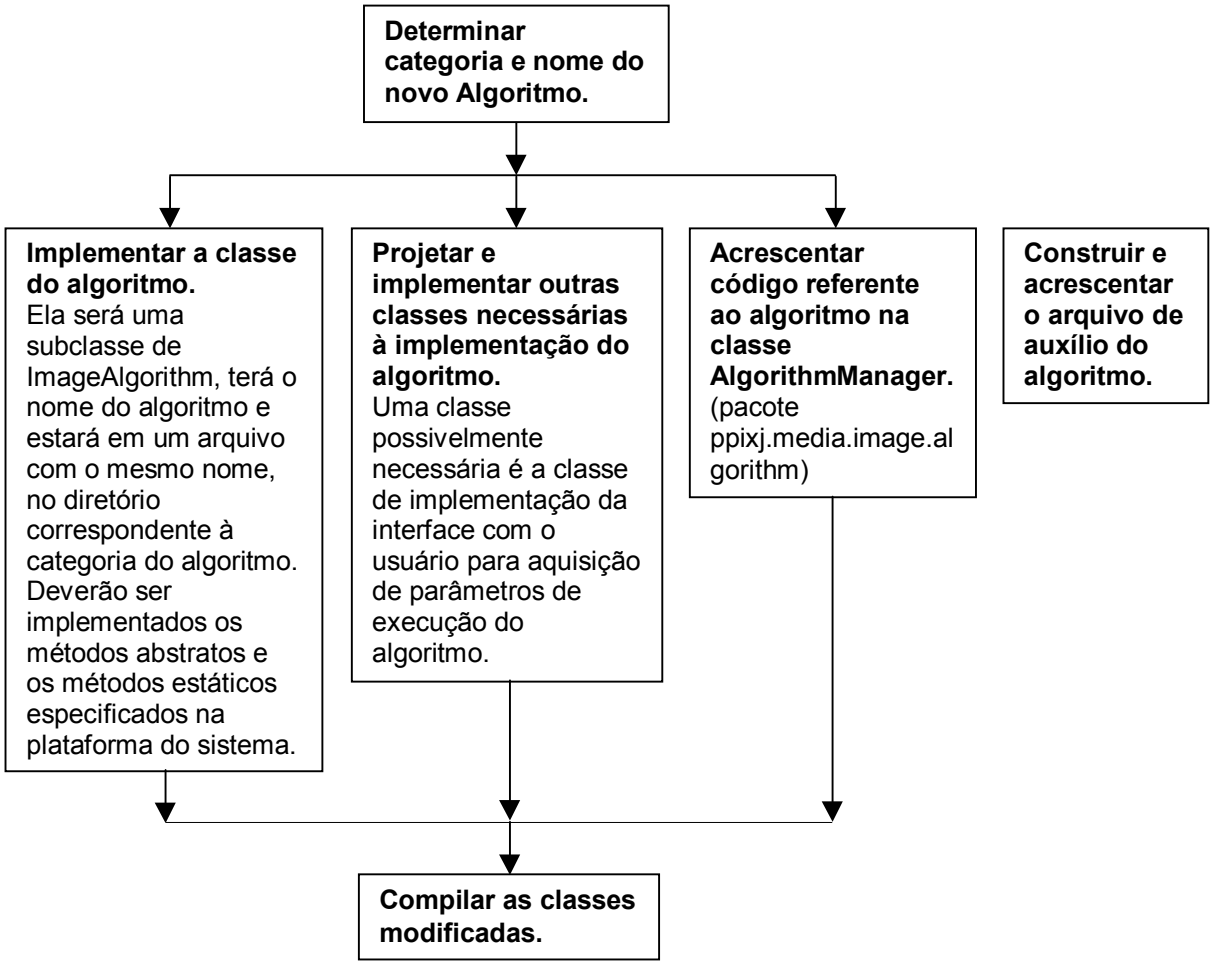

Figura 5.1: Diagrama com os passos do processo de adição de um algoritmo no PhotoPixJ.

Devido a algumas soluções utilizadas no PhotoPixJ, deverão ser implementados alguns elementos adicionais. Deve ser implementado um atributo estático de controle, o hasHelp, que é constante (palavra reservada final em Java 1.0.2) que indica se existe uma descrição (ajuda ou *help*) disponível daquele algoritmo. Deverá haver outro atributo estático de controle, enabled que indica se, em um determinado estado do sistema, aquele algoritmo está ou não habilitado. Juntamente com esse atributo, deverão ser definidos os métodos isEnabled, disable e enable. O método isEnabled retorna o valor de Enabled e os métodos disable e enable alteram seu valor. Deverá também ser definido o nome do algoritmo, o atributo estático name, que corresponde ao nome da classe incluindo o nome da categoria – para a classe Average_Filter, o nome seria {diretório base dos algoritmos + "Spatial_Smoothing/Average_Filter"}. O nome será usado, dentre outras coisas, para que o algoritmo seja incluído corretamente na interface com o usuário para acesso aos algoritmos.

Se o algoritmo apresenta parâmetros de execução que deverão ser obtidos do usuário, além da classe de algoritmo, deverá ser implementada a interface com o usuário de aquisição desses parâmetros. No exemplo do Filtro da Média, o parâmetro requisitado é a dimensão da máscara a ser usada (veja Seção 5.3).

Deverá ser acrescentado código relativo ao novo algoritmo no método AlgorithmManager (do pacote ppixj.media.image.algorithm), que inicia e controla a execução dos algoritmos.

Finalmente, se existe um arquivo de ajuda para a execução do algoritmo ele deverá ser inserido no caminho (diretório) de ajuda relativo àquele algoritmo (veja Seção 5.4).

5.3 Interface com o Usuário para os Algoritmos

No exemplo do Filtro da Média, um parâmetro requisitado do usuário é a dimensão da máscara a ser usada. Uma possível interface com o usuário para a aquisição do parâmetro do Filtro da Média pode ser vista na Figura 5.2.

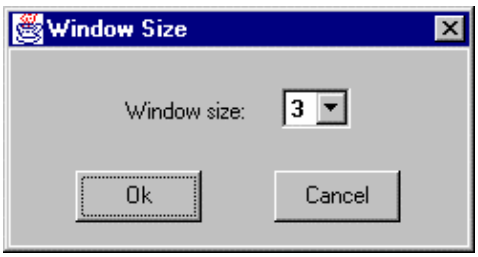

Figura 5.2: Janela de aquisição do parâmetro para a execução do Filtro da Média.

A classe de implementação da janela de aquisição do parâmetro para o Filtro da Média poderia se chamar AverageFilterDialog e ela seria chamada dentro do método setParameters da classe Average_Filter.

As classes de interface com o usuário dos algoritmos deverão pertencer ao pacote ppixj.media.image.algorithms_ui.

Interfaces com o usuário para algoritmos já implementadas e disponíveis podem ser vistas no Apêndice A.

5.4 Arquivos de Ajuda dos Algoritmos

Para se adicionar a descrição de um algoritmo, deve-se criar um arquivo no formato HTML. Esse arquivo tem um nome padrão, Principal.html. Ele deve ser inserido em um subdiretório específico para o algoritmo, sob o subdiretório de ajuda dos algoritmos, help/algorithms. O subdiretório específico do algoritmo é o mesmo que o do pacote onde o algoritmo é implementado acrescido do nome da classe do algoritmo. Por exemplo, no caso da classe de algoritmo Average_Filter, de implementação do Filtro da Média, o arquivo Principal.html ficaria no subdiretório help/algorithms/Spatial_Smoothing/Average_Filter.

5.5 Observações

No processo de adição de um algoritmo, na criação da classe de algoritmo, na maioria das vezes, é aconselhável tomar uma classe de algoritmo já implementada como ponto de partida e modificá-la de acordo com o algoritmo específico a ser implementado.

5.6 Conclusões

As experiências obtidas com a adição de algoritmos mostraram que essa tarefa no PhotoPixJ é muito simples. Foram adicionados ao sistema vários algoritmos por diferentes estudantes e pesquisadores de PDI que tiveram facilidade, tanto no uso da plataforma do sistema como no uso da linguagem Java.

A solução de interface com o usuário para acesso aos algoritmos facilitou a automatização da inserção de elementos de interface para acesso a novos algoritmos inseridos: a construção da árvore de algoritmos é feita com base na informação do nome de cada algoritmo que inclui a informação de sua posição na hierarquia.

A plataforma PhotoPixJ serve como uma ferramenta para estudantes e pesquisadores de computação implementarem e testarem algoritmos de PDI de forma fácil e rápida.

Capítulo 6

Conclusões e Trabalhos Futuros

6.1 Conclusões

Um objetivo neste trabalho foi a concepção de uma sistema para execução de algoritmos de PDI, independente de plataforma de *hardware* e que apresentasse alta portabilidade. Nesse contexto, o uso da linguagem Java apresentou vantagens na implementação do PhotoPixJ. Java, sendo uma linguagem independente de plataforma de *hardware*, faz com que o sistema seja altamente portátil. Para a inserção de novas operações, só é necessário usar a API de Java, tanto para o código relativo aos algoritmos como para o código relativo à interface com o usuário. Portanto, para que o sistema se mantenha multiplataforma, com a adição de uma nova função, é necessário implementar apenas uma versão de código.

Outro objetivo foi disponibilizar um conjunto de classes que proporcionasse fácil implementação e integração de novos algoritmos, além de outras funções, como de interface com o usuário. A plataforma para sistemas multimídia adotada se apresentou adequada para a fácil adição de novas classes de mídia, novas classes de algoritmos (uso de objetos funcionais) e novos formatos de mídia. Além disso, o desenvolvimento dos demais aspectos do sistema com base em um projeto orientado para objetos permite que os conjuntos de funções, como funções de interface com o usuário, sejam facilmente estendidos.

Java apresenta, em sua API, classes e métodos que proporcionam fácil representação e manipulação de imagens. Deve-se utilizar o que já existe disponível na API de Java e, à medida em que for necessário, devem-se criar outras classes para representação e processamento das imagens, como no caso da plataforma PhotoPixJ.

As experiências obtidas com a adição de algoritmos mostraram que a tarefa de se acrescentar um novo algoritmo no PhotoPixJ é muito simples. Foram adicionados ao sistema vários algoritmos por diferentes estudantes e pesquisadores de PDI que tiveram facilidade, tanto no uso da plataforma como no uso da linguagem Java.

A solução de interface com o usuário adotada se mostrou bastante adequada para acesso fácil aos algoritmos e para automatização de inserção de elementos de interface para acesso a um novo algoritmo adicionado ao sistema.

Java, sendo uma linguagem semi-interpretada, pode produzir programas bem mais lentos do que programas em uma linguagem como C. Isso pode ser crítico quando se trata de processamento de imagens e de mídias em geral. Devem ser usados compiladores JIT para se obter execução eficiente do programa.

A plataforma PhotoPixJ serve como uma ferramenta para estudantes e pesquisadores de computação implementarem e testarem algoritmos de PDI de forma fácil e rápida.

6.2 Trabalhos Futuros

O PhotoPixJ é um sistema em constante extensão pelo próprio objetivo deste trabalho. Além de algoritmos de processamento, novas mídias e novos formatos, o sistema provê forma fácil de extensões de outros recursos.

Devem ser estudados e inseridos novos recursos de interface com o usuário. Devem ser criadas formas mais automáticas para o processamento como teclas de atalho para comandos. Outro recurso desejável constitui prover automatização para execução de um conjunto de algoritmos sobre imagens através, por exemplo, da disponibilidade de linguagens *script* ou de um ambiente de programação visual.

Além da utilização de compiladores JIT para obtenção de execução eficiente, devem ser estudados outros recursos como paralelismo e processamento distribuído: um ponto de partida é o fato de Java ser uma linguagem “de vários processos” (*Multithreaded*).

O sistema foi implementado em Java 1.0.2 e deve ser atualizado para a versão mais recente para que se possa fazer uso de novos recursos da linguagem.

A plataforma PhotoPixJ pode ser estendida a fim de que ela possa processar outros tipos de mídia, como áudio, vídeo ou animação.

Apêndice A

Interfaces com o Usuário para Algoritmos Já Implementadas

A.1 Janela de Aquisição de Opções de Execução para Algoritmos de Detecção de Bordas

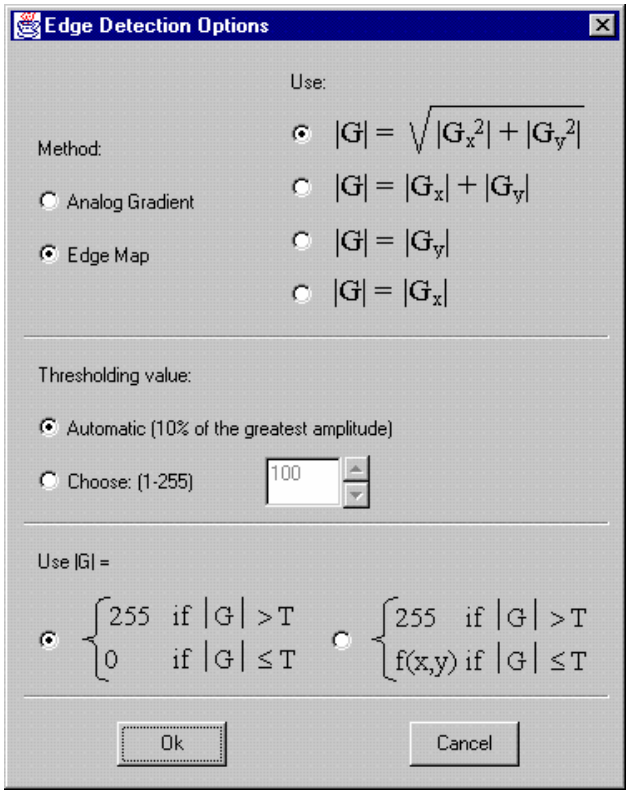

Figura A.1: Janela de opções de execução para algoritmos de detecção de bordas.

A.2 Janela de Aquisição de Dimensão de Máscaras

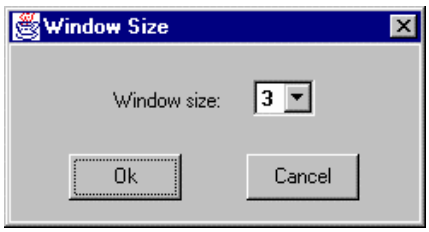

Figura A.2: Janela de aquisição de dimensão de máscaras (ou janelas) para execução de alguns algoritmos.

Apêndice B

Exemplo de Interface do PhotoPixJ no Unix®

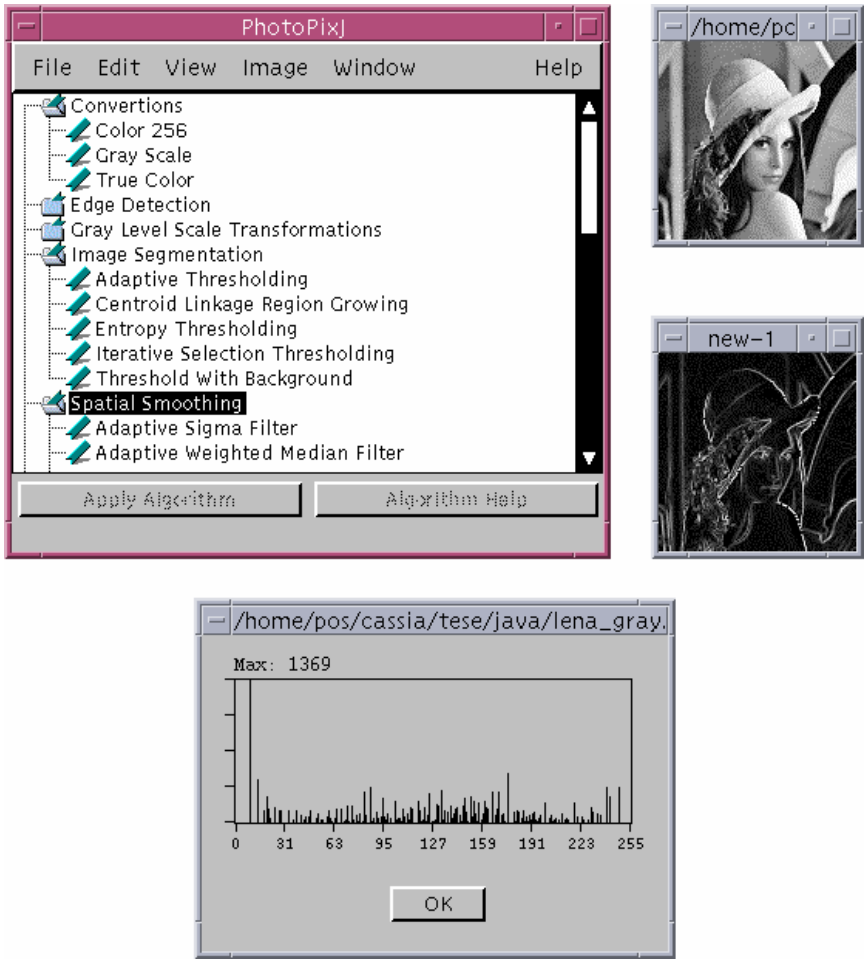

Figura B.1: Interface com o usuário do PhotoPixJ no sistema Solaris versão 2.5, ambiente CDE (*Common Desktop Environment*) versão 1.0.2.

Bibliografia

[Arnold96] Arnold, K. & Gosling, J., *The Java Programming Language*, Addison-Wesley Publishing Company, Inc., 1996

[Blattn92] Blattner, M.M. & Dannenber, R.B. [Editors], *Multimedia Interface Design*, Addison-Wesley Publishing Company, Inc., 1992

[Booch94] Booch, G., *Object-Oriented Analysis and Design – with Applications*, 2nd. Ed., The Benjamin/Cummings Publishing Company, Inc., 1994

[Buford94] Buford, J.F.K. [Editor], *Multimedia Systems*, Addison-Wesley Publishing Company, Inc., 1994

[Campio96] Campione, M. & Walrath, M., *The Java Tutorial – Object Oriented Programming for the Internet*, Addison-Wesley Publishing Company, Inc., 1996

[Coad95] Coad, P., North, D. & Mayfield, M., *Object Models: Strategies, Patterns, and Applications*, Prentice-Hall, Inc., 1995

[Collin95] Collins, D., *Designing Object-Oriented User Interfaces*, The Benjamin/Cummings Publishing Company, Inc., 1995

[Cotter95] Cotter, S. & Potel, M., *Inside Taligent Technology*, Addison-Wesley Publishing Company, Inc., 1995

[Dacont96] Daconta, M.C., *Java(tm) for C/C++ Programmers*, John Wiley & Sons, Inc., 1996

[Espese97] Espeset, T., *Kick Ass Java Programming*, The Coriolis Group, Inc., 1996

[Flanag96] Flanagan, D., *Java in a Nutshell*, O'Reilly & Associates, Inc., 1996

[Fowler97] Fowler, M. & Scott, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley Publishing Company, Inc., 1997

[Gamma95] Gamma, E., Helm, R., Johnson, R. & Vlissides, J., *Design Patterns – Elements of Reusable Object-Oriented Software*, Addison-Wesley Publishing Company, Inc., 1995

[Geary97] Geary, D. M. & McClellan, A. L., *Graphic Java – Mastering the AWT*, Prentice Hall, Inc., 1997

[Geary97] Geary, D. M. & McClellan, A. L., *Graphic Java – Mastering the AWT*, Prentice Hall, Inc., 1997

[Gibbs94] Gibbs, S.J. & Tsichritzis, D.C., *Multimedia Programming – Objects, Environments and Frameworks*, Addison-Wesley Publishing Company, Inc., 1994

[Gonza93] Gonzalez, R. C. & Woods, R. E., *Digital Image Processing*, Second Edition, Addison-Wesley Publishing Company, Inc., 1993

[Gosli96a] Gosling, J. & McGilton, H., *The Java Language Environment* - A White Paper, Sun Java Documention, 1996

[Gosli96b] Gosling, J. & Joy, B. & Steele, G., *The Java Language Specification*, Addison-Wesley Longman, Inc., 1996

[Gosli96c] Gosling, J. & Yellin, F. & The Java Team, *The Java Application Programming Interface*, volume 1 e 2, Addison-Wesley Longman, Inc., 1996

[HTML97] http://www.w3.org/TR/REC-html40, *HTML 4.0 Specification*, W3C Recommendation, 18-Dec-1997

[Jacobs92] Jacobson, I., Christerson, M., Jonsson, P. & Overgaard, G., *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley Publishing Company, Inc., 1992

[Jacobs94] Jacobson, I., Ericsson, M. & Jacobson, A., *The Object Advantage – Business Process Reengineering with Object Technology*, Addison-Wesley Publishing Company, Inc., 1994

[Jain89] Jain, A. K., *Fundamentals of Digital Image Processing*, Prentice-Hall, Inc., 1989

[JavaSoft] JavaSoft, http://java.sun.com

[Kay92] Kay, D.C. and Levine, J.R., *Graphics File Formats*, Windcrest Books, 1992

[Khoral] Khoral Research, Inc., http://www.khoral.com

[Kramer96] Kramer, D., *The Java™ Platform - a White Paper*, Sun Java Documention, 1996

[Martin95] Martin, R.C., *Designing Object-Oriented C++ Applications – Using the Booch Method*, Prentice-Hall, Inc., 1995

[Murray94] Murray, J. & van Ryper, W., *Encyclopedia of graphics File Formats*, O'Reilly & Associates, 1994

[Pree94] Pree, W., *Design Patterns for Object-Oriented Software Development*, Addison-Wesley Publishing Company, Inc., 1994

[Rational] Rational *Software* Corporation, *Rational Rose/C++ 3.0.2 manuals*, 1996

[Shneid92] Shneiderman, B., Designing the User Interface - Strategies for Effective Human-Computer Interaction, Addison-Wesley Publishing Company, Inc., 1992

[Sol93] Sol, A.A.S., *PhotoPix: Uma Plataforma para Processamento Digital de Imagens Orientada para Objetos*, Dissertação de mestrado, DCC/UFMG, Belo Horizonte, MG, Brasil, 1993

[Sol93b] Sol, A.A.S. & Araújo, A. de A., *PHOTOPIX: Uma plataforma para sistemas de processamento digital de imagens orientada para objetos*, Anais do VI Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens - SIBGRAPI, Recife-PE, 1993, pp 65-73

[Sol95] Sol, A.A.S., *PhotoPix: An Object-Oriented Framework for Digital Image Processing Systems*, in: Lecture Notes in Computer Science; Vol. 974 - Image Analysis and Processing, pp. 109-114, Springer-Verlag, 1995

[Sulliv91] Sullivan, J.W. & Tyler, S.W. [Editors], *Intelligent User Interfaces*, Addison-Wesley Publishing Company, Inc., 1991

[SunJIT] *The JIT Compiler Interface Specification*, http://java.sun.com/docs/jit_interface.html

[Symantec] Symantec Corporation, Symantec Café manual, 1996