PhotoPixJ: uma Plataforma para Processamento Digital de Imagens em Java

ADRIANA CÁSSIA ROSSI DE ALMEIDA[1]
ALISSON AUGUSTO SOUZA SOL[2]
ARNALDO DE ALBUQUERQUE ARAÚJO[3]

DCC - Departamento de Ciência da Computação
UFMG - Universidade Federal de Minas Gerais
Caixa Postal 702
30.161-970 Belo Horizonte, MG, Brasil
http://www.npdi.dcc.ufmg.br
[1]cassia@dcc.ufmg.br, [2]sol@acm.org, [3]arnaldo@dcc.ufmg.br

Abstract. This work presents the PhotoPixJ, a framework to ease the implementation of image processing algorithms, supporting several image types and formats. The system was implemented in Java™, with application and applet versions. The system is multiplatform, relying only on the broadly available Java Virtual Machine 1.0.2. New algorithms and other extensions will automatically be usable in all platforms for which a compatible JVM is implemented. Use of the PhotoPixJ enables a great productivity gain for developers working with digital image processing, due to the high abstraction level of the classes that need to be implemented or modified to add new functionality to the basic framework.

Keywords: Digital Image Processing, Algorithms, Java, Framework, Software Engineering

1 Introdução

A área de processamento digital de imagens [Gonza93, Jain89] pode ser dividida em duas categorias de funcionalidade: uma consiste na melhoria da qualidade das imagens digitais e outra 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 desenvolvedores de sistemas de PDI (Processamento Digital de Imagens) deparam-se 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 APIs (*Application Programming Interfaces*, i.e., Interfaces para Programação de Aplicativos) disponíveis, que dificulta a portabilidade dos sistemas, em especial o código de interface com o usuário.

Com o propósito de solucionar o problema exposto, analisou-se a filosofia da linguagem de programação Java™ [Arnold96, Campi96, Flanag96, 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 para o qual exista uma implementação da *Máquina Virtual Java*.

Este trabalho apresenta o PhotoPixJ, um ambiente independente de plataforma de hardware para execução de algoritmos de PDI, que constitui uma plataforma que permite fácil implementação e integração de novos algoritmos.

Os exemplos de código mostrados neste trabalho estão na linguagem Java 1.0.2 [Arnold96]. Diagramas de classes são apresentados na notação UML (*Unified Modeling Language*) [Fowler97].

A seguir, serão descritos os trabalhos relacionados, a hierarquia de classes para aplicativos multimídia projetada para o sistema PhotoPixJ, aspectos de implementação, a interface com o usuário, o procedimento para a inserção de um algoritmo, o sistema de ajuda, ambientes de uso e as conclusões e trabalhos futuros.

2 Trabalhos relacionados

Este trabalho partiu do estudo do sistema PhotoPix [Sol93, Sol95]. O trabalho em [Sol93, Sol95] descreve como o uso de técnicas de análise e projeto orientados para objetos [Booch94, Coad95, Jacobs92, Pree94] podem ajudar a isolar a implementação de algoritmos em sistemas de PDI da codificação de funcionalidade "não essencial". Com essa diretiva, o sistema PhotoPix foi construído visando ser posteriormente estendido, através da adição de 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 Microsoft Windows®. A adição de algoritmos nesse sistema depende do aprendizado da linguagem C++, que apresenta um grau de dificuldade relativamente alto. Além disso, para se implementar interfaces gráficas específicas dos algoritmos (por exemplo: solicitação de parâmetros) é necessário o aprendizado e uso 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 sempre e somente a biblioteca de classes pré-disponível na linguagem Java. Além disso, o grau de dificuldade de aprendizado de Java é relativamente baixo.

Outras plataformas, 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. A abstração das classes no 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.

3 Plataforma para sistemas multimídia

Uma plataforma (*framework*) [Booch94, Gibbs94] pode ser definida como uma coleção de classes abstratas relacionadas entre si, cooperando para produzir uma solução de projeto reusável para um dado problema.

Geralmente, classes de plataformas especificam interfaces (classes abstratas), deixando em aberto a implementação. Adicionando-se novas classes, através da especialização, as interfaces podem ser estendidas. Por exemplo, uma plataforma para multimídia poderia conter duas classes abstratas: `Imagem`, para representação de imagens, e `ProcessamentoDeImagens`, para implementação de algoritmos de PDI. As interfaces para essas classes podem ser especificadas sem preocupação com os detalhes de representação de uma imagem em particular ou detalhes de um algoritmo específico. A seguir, será descrita a hierarquia de classes para aplicativos multimídia utilizada no sistema PhotoPixJ.

3.1 Visão geral da plataforma

Para definição da hierarquia de classes de aplicações multimídia é usual a divisão em quatro categorias [Gibbs94]: classes de mídia, classes de processamento, classes de formato e classes de componentes. Em síntese, classes de mídia correspondem a áudios, vídeos, imagens e outros tipos de mídia; classes de processamento representam operações sobre mídias (algoritmos); classes de formato tratam das representações internas e externas dos dados de mídia; classes de componentes permitem a interface do sistema com dispositivos externos para aquisição e apresentação de mídias, como câmeras, microfones, conversores analógico-digitais, etc. (tais classes não serão tratadas neste trabalho).

Para o sistema PhotoPixJ, foram projetadas e construídas classes de mídia, classes de processamento e classes de formato. Para produção e obtenção de imagens, são utilizadas classes pré-disponíveis na biblioteca básica da linguagem Java.

3.2 Classes de Mídia

Mídias são usualmente divididas em tipos (áudio, vídeo, etc.), sendo cada tipo representado por uma classe. Essas são chamadas classes de mídia e têm a estrutura hierárquica mostrada na Figura 1.

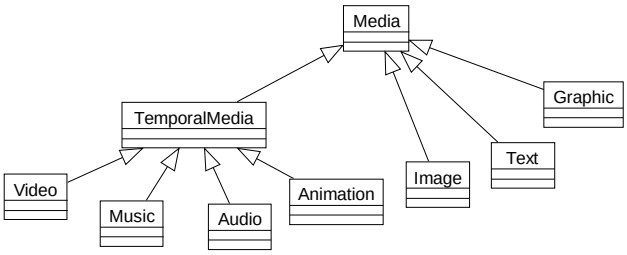

Figura 1. Hierarquia de classes de mídia

Ocorrências das classes de mídia são os objetos de mídia. Um objeto de mídia consiste em um descritor e um valor de mídia. O descritor contém os atributos do objeto de mídia, como o seu tamanho e formato, enquanto o valor de mídia consiste no dado usado para representar digitalmente o objeto de mídia, no formato adequado.

A classe de mídia mais genérica será chamada `Media`. Ela é uma classe abstrata que especifica tanto métodos que deverão ser implementados por uma classe de mídia concreta quanto métodos já implementados, pré-disponíveis para qualquer classe de mídia. A interface (aqui no sentido do conjunto das funções) para essa classe pode ser vista no diagrama da Figura 2.

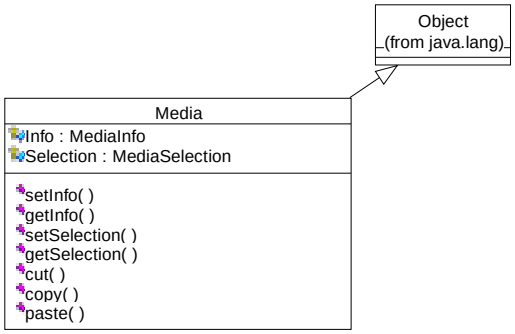

Figura 2. Classe abstrata de uma mídia genérica.

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 alterar e obter os atributos `Info` e `Selection` e os métodos abstratos de edição (`cut`, `copy`, `paste`), que deverão ser implementados em cada subclasse concreta de `Media`. Um exemplo de subclasse de `Media` seria a classe `Image`, que apresenta atributos e métodos específicos para uma imagem.

As subclasses imediatas de `Media` são a classe `Image`, `Graphic`, `Text` e `TemporalMedia`. O propósito de uma subclasse de `Media` é agrupar as operações específicas daquele tipo de 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, para imagens, alguns atributos ou métodos são específicos de imagens de tons de cinza, enquanto outros são específicos de imagens coloridas. Portanto, subclasses de `Image` poderiam incluir a classe `GrayScaleImage` e `ColorImage`. Seguindo tal raciocínio, a classe `ColorImage` deveria ter subclasses como `IndexedColorImage` e `TrueColorImage`.

Uma dificuldade inerente ao projeto de qualquer hierarquia de classes é o posicionamento de métodos. Na definição de métodos dentro da hierarquia das classes de mídia, pode não ser claro qual o posicionamento adequado, sendo necessária uma análise a cada caso, evitando uma sobrecarga de métodos desnecessários nas classes mais genéricas.

3.3 Classes de Processamento

Uma classe de processamento implementa o que é usualmente denominado "objeto funcional" [Gibbs94], que encapsula conjuntos de funções, seus argumentos (também chamados parâmetros de entrada/saída) e parâmetros de execução.

Cada objeto de uma classe de processamento contém atributos para armazenar os argumentos adequados para a função de processamento, implementada em um método padronizado da classe, denominado `execute`. Na utilização de um objeto funcional é invocado este método (`execute`), que atua sobre os argumentos de entrada, modificando ou gerando os argumentos de saída, de acordo com o algoritmo implementado e seus parâmetros de execução.

A interface para objetos funcionais é especificada pela classe abstrata `MediaAlgorithm` e pode ser vista no diagrama da Figura 3. A classe `MediaAlgorithm` possui várias subclasses abstratas com o propósito de agrupar famílias de algoritmos similares, como filtros de imagens, efeitos de vídeo. Essas classes têm efetivamente subclasses concretas, representando algoritmos específicos.

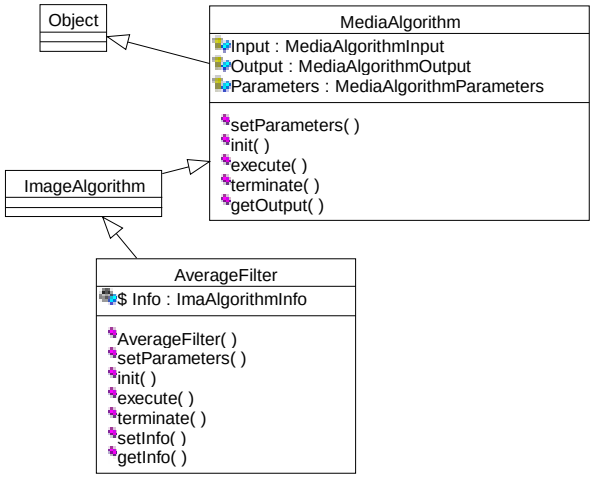

Figura 3. Hierarquia de classes de processamento

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. Considerando, por exemplo, um algoritmo para imagens, como o filtro da média, os argumentos de entrada e saída seriam imagens, tendo-se como parâmetro o tamanho da máscara. O método abstrato `setParameters` permite definir os valores dos parâmetros. O método abstrato `execute` deverá ser redefinido, implementado-se o algoritmo propriamente dito. Os métodos abstratos `init` e `terminate` são invocados no início e no final do método `execute`, permitindo às subclasses concretas iniciar e finalizar adequadamente mecanismos e estruturas de dados auxiliares.

Um exemplo simplificado de uso de objeto funcional pode ser visto no Código 1, onde existe um argumento de entrada e um de saída (ambos objetos da classe imagem), sem parâmetros de execução.

```
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("i.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 1. Implementação de um objeto funcional.

A alternativa ao uso de objetos funcionais seria fazer uma série de declarações de métodos em uma classe de mídia ou processamento. Algumas vantagens do uso de objetos funcionais são:

- Filtros de imagem, filtros de áudio, efeitos de vídeo e transições de vídeo são grandes em número e sempre passíveis de novas extensões. Assim, não são conhecidas previamente todas as operações para definição completa de métodos em uma só classe. Com objetos funcionais, novas operações podem ser adicionadas sem a necessidade de modificar as classes existentes;

- 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). Como exemplo, a separação de cores produz uma imagem (ou mais) 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;

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

Além de implementar todos os métodos abstratos da(s) superclasse(s), cada classe concreta de algoritmo deverá ter um atributo estático `Info`, que conterá 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: o `setInfo`, que permite alterar o objeto `Info`, e o `getInfo`, que permite consultar o valor atual (como, em Java 1.0.2, não é possível definir um método abstrato e estático ao mesmo tempo, a exigência de que métodos estáticos sejam implementados nas classes concretas é feita por documentação, ao invés de via declaração no código).

Seguindo a hierarquia de classes de algoritmos, poderia ser definida para, por exemplo, processamento de imagens, a classe abstrata `ImageAlgorithm`. Um algoritmo específico de processamento de imagens, por exemplo, o Filtro da Média, seria uma subclasse concreta da classe `ImageAlgorithm` (Figura 3).

3.4 Classes de Formato

Padrões para representação, principalmente externa, de valores de mídia são denominados formatos de mídia [Kay92, Murr94]. 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*). Como as soluções para as duas categorias são bastante semelhantes, será descrita apenas a solução para formatos de arquivo.

Para suporte aos diversos formatos de arquivos de mídia, são necessários métodos para decodificar arquivos nos diferentes formatos e representá-los no valor de mídia, além de métodos para fazer a codificação no sentido contrário, quando necessário. Uma solução para o posicionamento de tais métodos é adicioná-los às classes de mídia. Contudo, tal solução apresenta problemas. Não são conhecidos todos os formatos quando as classes de mídia são definidas, além da possibilidade de aparecimento de formatos adicionais em aplicações futuras. Suportar novos formatos ou versões irá requerer a adição de novos métodos ou a modificação de métodos existentes, o que implica na modificação das classes, tornando mais difícil a integração e manutenção do sistema.

Uma solução melhor é o uso de classes de formato. Cada formato é representado por uma classe que implementa métodos de importação (decodificação) e exportação (codificação). Classes de formatos são organizadas em uma hierarquia, com estrutura similar à hierarquia das classes de mídia. A raiz da hierarquia será a classe abstrata `MediaFormat`. Algumas subclasses, também abstratas, são recomendáveis para cada tipo principal de mídia. Os formatos reais são suportados em subclasses concretas, aparecendo como folhas na hierarquia.

A declaração para a classe `MediaFormat` pode ser vista no diagrama da Figura 4, sendo todos os seus métodos abstratos. O método `canDecode` verifica se o objeto da classe de formato pode importar um arquivo

específico. O método `decode` lê os dados de mídia em um arquivo e os representa em uma instância de alguma subclasse concreta de `Media`. 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.

Toda subclasse concreta de uma classe de formato deverá implementar os métodos abstratos das superclasses e também os métodos estáticos `guessFormat`, `isFormatOf`, `setInfo`, `getInfo`, que podem ser vistos no diagrama da Figura 4.

O método estático `guessFormat` tenta identificar o tipo de formato de um arquivo específico. Ele deve ser implementado nas classes de formato abstratas para cada tipo principal de mídia (por exemplo, na classe `ImageFormat`, que pode ser vista na Figura 4).

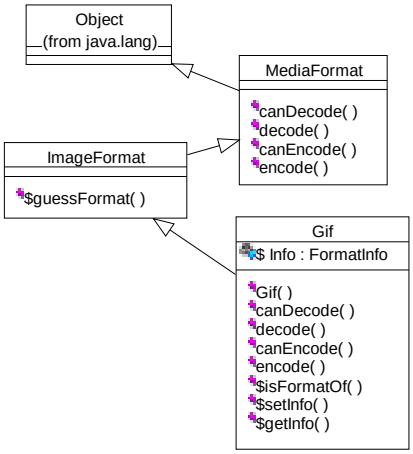

Figura 4. Classes de Formato.

O atributo estático `Info` contém informação que descreve o formato. Os métodos estáticos `getInfo` e `setInfo` permitem consultar e alterar o valor de `Info`, respectivamente. O método estático `isFormatOf` verifica se um arquivo usa o formato representado na classe de formato específica.

4 A implementação do PhotoPixJ

O PhotoPixJ pode ser executado como aplicação ou como *applet* [Arnold96, Campi96]. Algumas particularidades ocorrem durante a operação no modo *applet*, devido às restrições de segurança na implementação de *applets* no JDK 1.0.2.

O sistema compreende quatro pacotes lógicos principais:

- `ppixj.media.image`;
- `ppixj.media.image.algorithms`;
- `ppixj.media.image.formats`;
- `ppixj.ui`.

O pacote `ppixj.media.image` contém as classes de representação e manipulação de imagens. O pacote `ppixj.media.image.algorithms` apresenta todos os algoritmos de processamento de imagens. O pacote `ppixj.media.image.formats` apresenta todas as classes de formatos. O pacote `ppixj.ui` apresenta todas as classes de implementação de interface com o usuário.

4.1 Pacote `ppixj.media.image`

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 biblioteca básica a classe `Image` (pacote `java.awt`) e o pacote `java.awt.image` para representação e manipulação de imagens [Espese96, Geary97]. Para a implementação da plataforma proposta, foi necessário criar uma classe `Image` específica do PhotoPixJ. Seguindo a especificação da plataforma, foi implementado 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 5.

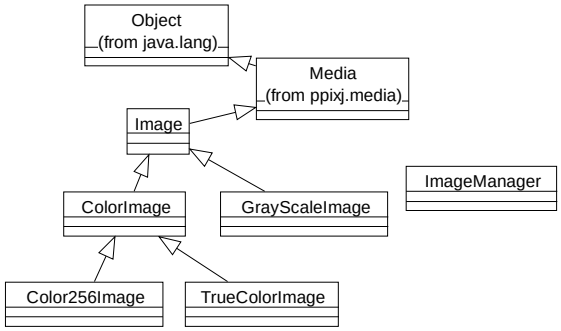

Figura 5. Classes do pacote `ppixj.media.image`.

A classe `Image` do PhotoPixJ é a superclasse de todas as classes de imagens no sistema. Ela apresenta um atributo, `Img`, objeto da classe `java.awt.Image`, pois a produção e obtenção de imagens reutiliza as classes da biblioteca padrão da linguagem Java.

A classe `ImageManager` é uma classe com métodos e atributos estáticos para o controle de criação e destruição de imagens e para a gerência das imagens no sistema.

No PhotoPixJ, pode-se trabalhar com imagens de tons de cinza, 256 cores ou cores reais. Para que o sistema possa trabalhar com outros tipos de imagens, é necessário que se acrescente o suporte para o novo tipo, através da codificação de uma subclasse concreta de `Image` (ou de uma de suas descendentes) na hierarquia de classes de imagens do sistema.

4.2 Pacote `ppixj.media.image.algorithms`

A plataforma para processamento de dados multimídia constitui a base para implementação e adição de novos algoritmos no PhotoPixJ. A relação das classes pode ser vista na Figura 6.

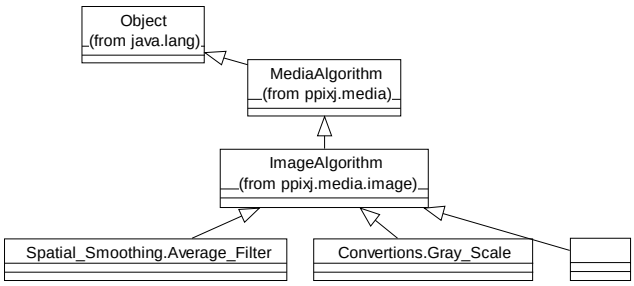

Figura 6. Classes de algoritmos de processamento

Todo algoritmo de processamento de imagens do sistema será uma classe concreta, subclasse de `ImageAlgorithm` e deverá pertencer ao pacote `ppixj.media.image.algorithms`.

4.3 Pacote `ppixj.media.image.formats`

Toda classe de formato pertencerá ao pacote `ppixj.media.image.formats`. A relação das classes do pacote pode ser vista na Figura 7.

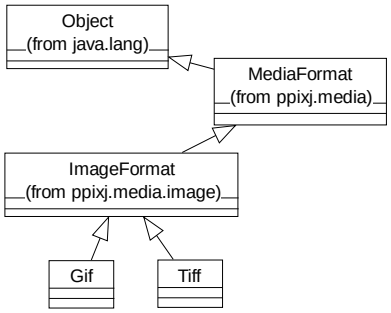

Figura 7. Classes de formatos de mídia no PhotoPixJ.

Toda classe de formato de imagens será uma subclasse de `ImageFormat`. No PhotoPixJ, foi implementado inicialmente o suporte a dois formatos: GIF e TIFF [Kay92, Murr94].

5 Interface com o usuário

A janela principal do PhotoPixJ (Figura 8) apresenta um menu com várias funções, tais como carregamento e salvamento de imagens, edição de imagens, operações de visualização, operações de conversões de tipo de imagens, dentre outras (algumas funções não estão presentes na versão *applet*). A interface principal apresenta também um painel de visualização em árvore, para acesso aos algoritmos do sistema.

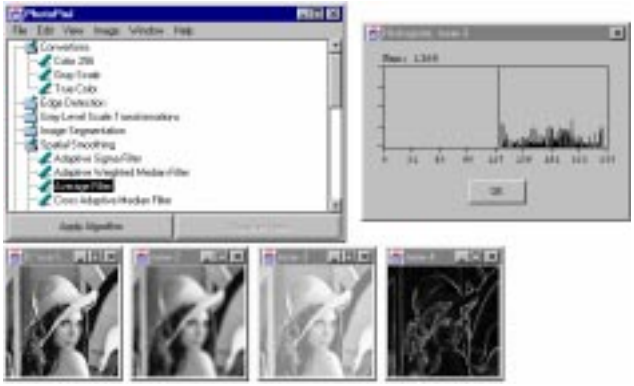

Figura 8. Interface com o usuário do PhotoPixJ.

Como o sistema poderá ter um número grande de algoritmos, classificados dentro de uma hierarquia flexível de categorias, foi importante obter uma solução de interface com o usuário em que fosse fácil ter acesso aos algoritmos. Além disso, na introdução de novos procedimentos, 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.

A interface com o usuário, assim como diversos outros aspectos do sistema, foi feita com base em um projeto orientado para objetos [Collin95].

6 Adição de algoritmos

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`. Tal algoritmo será inserido na categoria dos filtros de Suavização Espacial, no pacote `ppixj.media.algorithms.Spatial_Smoothing`. A classe será implementada em um arquivo com o mesmo nome (`Average_Filter`), que ficará no subdiretório `ppixj/media/algorithms/Spatial_Smoothing`, ao qual corresponde o pacote de suavização espacial.

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). Deverão ser implementados também alguns elementos adicionais, que não foram mencionados na descrição anterior das classes de processamento da plataforma.

Deve ser definido um atributo estático de controle, o `hasHelp`, que é constante e indica se existe uma descrição de auxílio (*help*) disponível para o 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 o atributo `enabled`, 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, no 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 árvore de visualização dos algoritmos.

Se o algoritmo apresenta parâmetros de execução que deverão ser obtidos do usuário, além da classe de algoritmo deverá ser implementada interface para aquisição desses parâmetros. No exemplo do Filtro da Média, o parâmetro requisitado é a dimensão da máscara a ser usada. Esta implementação pode utilizar classes de interface com o usuário disponíveis na biblioteca da linguagem Java.

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.

7 Sistema de ajuda

Como ferramenta de auxílio no estudo e implementação de algoritmos de PDI, é essencial prover acesso a uma descrição dos algoritmos presentes no sistema. Para isso foi projetado um sistema de auxílio aos algoritmos.

Para continuidade da qualidade no sistema de auxílio, é desejável que seja fácil a sua extensão, pela adição de informações sobre novos algoritmos, e manutenção, através da mudança do conteúdo dos arquivos. 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` [HTML97], associados a um navegador para apresentação.

Foi elaborado um modelo de formatação para a descrição de um algoritmo, que visa proporcionar o fácil entendimento de seu funcionamento e discriminar elementos para pesquisa, como referências bibliográficas. A partir do modelo, qualquer editor `HTML` pode ser utilizado para compor o arquivo de auxílio para um algoritmo do sistema.

8 Plataformas de Uso

O PhotoPixJ foi desenvolvido em Java 1.0.2 e testado no Windows 95 e no Solaris no ambiente CDE (*Common Desktop Environment*). O sistema funciona nos ambientes testados e não apresentou problemas durante o desenvolvimento. Alguns aspectos de interface gráfica funcionam de forma diferente nos ambientes testados, o que é natural devido às diferenças nas soluções em cada ambiente.

Java, como linguagem 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 (*Just in Time*) [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 pode se equiparar a de programas em linguagens como C++ [Mangi98].

9 Conclusões e trabalhos futuros

A plataforma multimídia projetada para o sistema PhotoPixJ foi considerada mais adequada que a biblioteca de classes pré-disponível na linguagem Java para adição de novas classes de mídia, novas classes de algoritmos (uso de objetos funcionais) e novos formatos de mídia. Apenas para a produção e obtenção de imagens optou-se por utilizar as classes da biblioteca da linguagem Java.

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

O uso de Java apresentou vantagens para a 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 própria linguagem Java, tanto para código relativo aos algoritmos como para eventual código relativo à interface de obtenção de parâmetros do usuário. Portanto, obteve-se um sistema multiplataforma que assim permanece após as extensões previstas para adição de novos algoritmos.

A plataforma PhotoPixJ serve como uma ferramenta para estudantes e pesquisadores de computação implementarem e testarem algoritmos de PDI de forma fácil e rápida. A continuidade na implementação do sistema de auxílio, que provê a descrição dos algoritmos, é essencial para que o PhotoPixJ cumpra sua função.

As experiências realizadas para adição de algoritmos ao sistema PhotoPixJ mostraram bons resultados. Foram adicionados vários algoritmos por estudantes de Ciência da Computação, que tiveram facilidade tanto no uso do sistema quanto no uso da linguagem Java, sendo capazes de integrar o primeiro algoritmo ao sistema em menos de duas semanas, na média, com produtividade posterior crescente.

Em trabalhos futuros, devem ser consideradas técnicas para melhoria de performance com uso de múltiplos processadores locais ou remotos, como paralelismo ou processamento distribuído. Isto será facilitado pelo fato de Java ser uma linguagem com suporte nativo a várias linhas de execução (*multithreaded*), além de conter, em sua biblioteca básica, diversas classes para comunicação entre processos em uma rede. Com essa diretiva, o sistema poderia ser estendido a fim de servir como plataforma para teste e implementação de algoritmos paralelos de PDI. A plataforma PhotoPixJ pode também ser facilmente estendida para processar outros tipos de mídia, como áudio, vídeo ou animação.

10 Agradecimentos

Os autores gostariam de agradecer ao CNPQ, à CAPES e à FAPEMIG, pelo suporte financeiro deste trabalho. Agradecem aos revisores pelas críticas apresentadas e pela atenção dedicada ao trabalho.

11 Referências

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

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

[Campi96] Campione, M. & Walrath, M., *The Java Tutorial – Object Oriented Programming for the Internet*, Addison-Wesley Publishing Co., 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 Co., Inc., 1995

[Espese96] Espeset, T., *Kick Ass Java Programming*, The Coriolis Group, Inc., 1996

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

[Fowler97] Fowler, M. & Scott, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley Publishing Co., 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 Co., 1994

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

[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 Co., 1992

[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

[Mangi98] Mangione, C., *Just In Time for Java vs. C++*, http://www.ncworldmag.com/ncw-01-1998/ncw-01-jperf.html?0112

[Murr94] 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 Co., 1994

[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

[Sol95] Sol, A.A.S. & Araújo, A. A., *PhotoPix: an Object-Oriented Framework for Digital Image Processing Systems*, In: Lecture Notes in Computer Science; Vol. 974 – Image Analysis and Processing, pp. 109-115, 1995

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