PhotoPix: uma plataforma para sistemas de processamento digital de imagens orientada para objetos

ALISSON AUGUSTO SOUZA SOL
ARNALDO DE ALBUQUERQUE ARAÚJO

DCC - Departamento de Ciência da Computação
UFMG - Universidade Federal de Minas Gerais
Caixa Postal 702
30.161-970 Belo Horizonte, MG, Brasil
alisson@dcc.ufmg.br
arnaldo@dcc.ufmg.br

Abstract. This work describes how the use of object-oriented programming techniques can help to separate the implementation of algorithms in digital image processing systems from the coding of "non-essential" functionality. With that directive it was conceived and implemented a system, named **PHOTOPIX**, wich was coded in the C++ language, using the application programming interface of the *Microsoft Windows* environment. Methods for converting the spectral information between images of different spectral resolution were also investigated.

1 Introdução

A contínua evolução das tecnologias de fabricação de circuitos integrados torna cada vez maior a capacidade de processamento dos computadores. Como resultado disto, é crescente o número de plataformas de *hardware* capazes de efetuar de maneira eficiente o processamento digital de imagens (PDI).

A finalidade dos sistemas de PDI é permitir a melhoria na qualidade de imagens digitais e automatizar procedimentos de extração de informação da imagem, como a segmentação e o reconhecimento de padrões [Jain89, Prat78].

Entretanto, dois grandes problemas ainda impedem que programadores de sistemas de PDI concentrem-se unicamente na implementação da numerosa quantidade de algoritmos para tratamento de imagens sendo pesquisados na atualidade:

- a falta de padronização para o formato de arquivos armazenando imagens digitais;
- a incompatibilidade entre as diversas plataformas de hardware e APIs (*Application Programming Interfaces*, isto é, interfaces para programação de aplicativos) disponíveis, dificultando a portabilidade de código-fonte, em especial o relativo à interface aplicativo-usuário.

A impossibilidade de implementação de um sistema fornecendo suporte (leitura/escrita) a todos os tipos de arquivos de imagens existentes, assim como a todas as plataformas de *hardware* e APIs disponíveis, obriga o desenvolvedor a concentrar-se em um número limitado de escolhas.

Infelizmente, não apenas méritos técnicos podem ser utilizados para decidir dentre as diversas possíveis opções. Pressões comerciais têm freqüentemente levado ao completo desuso tanto formatos de arquivo que já foram muito utilizados quanto plataformas de *hardware* e APIs que poderiam até mesmo ser consideradas avançadas demais para sua época de lançamento.

2 Definição do sistema PHOTOPIX

O objetivo do sistema **PHOTOPIX** é fornecer uma base, tanto conceitual quanto prática, para sistemas de processamento de imagens. Tal sistema deve poder migrar não apenas entre diversas plataformas de *hardware* e APIs da atualidade, mas também ao longo do tempo, implementando plenamente o conceito de reusabilidade de *software* [Cox86, Mart93].

Muitos esforços têm sido duplicados na codificação de algoritmos de PDI, simplesmente porque não se consegue reaproveitar a maior parte do código já existente. Isto ocorre porque, na maioria das implementações, é difícil separar o que seja código relativo ao algoritmo do que vem a ser código para manutenção da interface aplicativo-usuário, leitura/escrita de arquivos ou tratamento de restrições da API e do *hardware* utilizados.

A principal diretriz do sistema **PHOTOPIX** é isolar a implementação de algoritmos de PDI da codificação de outras funcionalidades "não essenciais".

A contínua evolução das tecnologias usadas na Informática, aproxima as plataformas de *hardware* atuais cada vez mais das concebidas teoricamente segundo o princípio de tecnologia perfeita da Análise Essencial [McPa84]. Portanto, um sistema que pretenda

portabilidade ao longo do tempo não deve conter, ou deve restringir ao mínimo possível, a sua dependência de configurações de *hardware* específicas.

Diversas APIs disponíveis atualmente permitem a codificação completa de um sistema sem qualquer (ou com um mínimo de) referência à plataforma de *hardware* em uso. Dentre estas, a API do ambiente operacional *Microsoft Windows* [MSDK92, Msft92a, NoYa90, Petz92, Sol93] foi escolhida como base para a implementação do sistema **PHOTOPIX**.

O uso de técnicas de programação orientada para objetos é a melhor opção atualmente disponível para implementação de sistemas onde se deseja "encapsular" detalhes e "isolar" diversos módulos. Para codificação do sistema **PHOTOPIX**, foi escolhida a mais popular das linguagens orientadas para objetos: a linguagem **C++** [AnAn91, Chri92, DeSt89, MSVC93, Sol93].

A solução mais difundida para lidar com o problema da multiplicidade de formatos para arquivos contendo imagens digitais consiste em padronizar apenas um formato para os dados sobre os quais atuam os algoritmos de PDI implementados em um sistema. São então codificados diversos conversores entre este "formato interno" das imagens em um sistema de PDI e os inúmeros formatos existentes para arquivos contendo imagens [KaLe92]. O sistema **PHOTOPIX** adota um formato interno para as imagens que é denominado *Packed-DIB* [Petz91]. Tal formato foi escolhido por ser compatível com algumas primitivas gráficas do ambiente *Microsoft Windows* [MSDK92].

Atualmente, considera-se que uma resolução espectral de 24 *bits/pixel* é suficiente para a maior parte das aplicações de processamento digital de imagens. É comum utilizar o sistema RGB [FVFH90] para especificação das cores em uma imagem digital, designando 8 *bits/pixel* para cada componente. Contudo, diversos formatos populares para arquivos contendo imagens não possibilitam o armazenamento de 24 *bits* de informação espectral por *pixel*. Além disto, há ocasiões onde são necessárias representações de uma imagem em escala de tons de cinza [FVFH90, HeBa86] ou com baixa resolução espectral, para impressão, transmissão ou armazenamento em dispositivos de baixa capacidade (como disquetes).

É consenso que uma plataforma para sistemas de PDI não deve obrigar o programador de um algoritmo a implementar versões específicas para cada uma das resoluções espectrais aceitas pelo sistema ou ter de optar por um código genérico e ineficiente. A maioria dos sistemas de PDI permite que determinados algoritmos sejam implementados apenas para imagens com uma resolução espectral específica, geralmente a mais alta disponível. Desabilita-se a seleção destes algoritmos quando a imagem "ativa" não tiver a resolução espectral adequada, sendo implementados algoritmos para converter uma imagem entre as diversas resoluções espectrais possíveis. Esta "solução de consenso" foi adotada no sistema **PHOTOPIX**.

Diversos algoritmos para conversão da informação espectral entre imagens com diferentes resoluções espectrais foram pesquisados. Um problema particularmente difícil é a redução do número de *bits* utilizados para armazenar a informação espectral sobre cada *pixel*. São implementados no sistema **PHOTOPIX** os algoritmos de redução da informação espectral para uma palheta uniforme, o algoritmo da popularidade e o algoritmo do corte da mediana [Heck82], sendo consideradas as possibilidades de conversão para escala de tons de cinza e espalhamento do erro local [Sol93, Ulic88].

3 Programação orientada para objetos

O paradigma da programação orientada para objetos é a adição à linguagem em uso de novos tipos de dados, pelo programador de aplicativos. Numa linguagem orientada para objetos, o desenvolvedor de um sistema de processamento de imagens deve ter possibilidade de definir um tipo de dados "Imagem", declarar duas variáveis daquele tipo e usar funções como a adição ou a subtração de imagens numa sintaxe idêntica à utilizada para os tipos de dados pré-definidos.

Embora a idéia básica seja bastante simples, diversas questões são problemáticas em relação à implementação de sistemas orientados para objetos:

- como definir os novos tipos de dados básicos?
- em que linguagem implementar o sistema?
- quais as limitações e problemas decorrentes da linguagem escolhida?
- quais tipos básicos de dados podem ser aproveitados de "bibliotecas" desenvolvidas por empresas especializadas?

Princípios da programação orientada para objetos

São três as principais diferenças de uma linguagem orientada para objetos em relação às outras linguagens de programação: a maior capacidade de "encapsulamento" de detalhes no código-fonte, o "polimorfismo" das operações e a possibilidade de "herança" de funcionalidades de tipos pré-definidos [AnAn91, Chri92, Cox86, DeSt89, Mart93].

Na terminologia da programação orientada para objetos, os tipos de dados, tanto básicos como definidos pelo programador de aplicativos, constituem "classes". A declaração de uma variável de determinada classe gera o aparecimento de um "objeto" (na linguagem Pascal orientada para objetos, o que aqui é denominado

classe recebe o nome de objeto, sendo uma variável de determinado tipo denominada "instância").

Por "encapsulamento" entende-se a possibilidade de adicionar novas classes a uma linguagem sem tornar públicos detalhes da implementação. De certa forma, o encapsulamento já está presente em quase todas as linguagens de programação da atualidade. Diversos tipos básicos de dados, como números inteiros e reais, estão usualmente disponíveis, sem que o programador saiba como são armazenados seus valores ou como são implementadas as operações disponíveis nestas classes pré-definidas.

O "polimorfismo", que significa literalmente "muitas formas", é uma característica que expressa a capacidade de uma mesma operação ter seu comportamento modificado, de acordo com as classes dos objetos sobre os quais irá atuar. Um certo grau de polimorfismo também já está presente na maioria das linguagens da atualidade. É comum que a adição seja sempre representada por um único símbolo, usualmente o "+", independente da classe dos números a serem adicionados. Assim, a expressão "x + y" pode estar representando a soma de dois números inteiros, dois números reais ou, em algumas linguagens, a soma de um inteiro a um real. Neste caso de polimorfismo, a escolha dentre todas as diferentes formas de um operador é feita em "tempo de compilação". A declaração das variáveis permite ao compilador determinar a classe correta que implementa o operador que atua sobre os objetos em uma expressão.

Através do mecanismo de "herança", torna-se possível criar novos tipos de objetos aproveitando as classes já existentes, pela adição de dados ou funções, ou a modificação do comportamento das funções já definidas. A classe original, denominada "classe base", é especializada na "classe derivada". A relação entre as classes deve ser estabelecida de forma que objetos da classe derivada sejam casos particulares da classe base, e não partes dela [Mart93].

O polimorfismo, no contexto da programação orientada para objetos, deve aproveitar também a existência de hierarquias de classes, que formam uma "família de classes". Uma operação definida para atuar sobre objetos de determinada classe deverá ter um comportamento previsível quando atuar sobre objetos de classes derivadas. Contudo, é útil que seja possível especificar novas versões da operação nas classes derivadas, deixando para o momento de execução a determinação da versão adequada, de acordo com a verdadeira classe dos objetos sobre os quais a operação irá atuar (tal funcionalidade é implementada através das funções virtuais na linguagem C++ [MSVC93]).

Claramente, a implementação de uma linguagem orientada para objetos exige a definição de diversas regras de sintaxe e semântica. Uma complicação adicional aparece quando se deseja que a tecnologia de orientação para objetos seja adicionada a uma linguagem já existente. Apesar disto, as mais populares linguagens orientadas para objetos da atualidade tiveram como "base" linguagens já existentes.

Uma linguagem de programação deve auxiliar no projeto, implementação, extensão, correção e otimização de um sistema [Joyn92].

Uma linguagem orientada para objetos auxilia no projeto de um sistema, pois direciona o projetista para a identificação de classes, que podem então ser implementadas reproduzindo fielmente o especificado. A extensão da funcionalidade das classes de um sistema também é facilitada, pelo encapsulamento dos detalhes e pela possibilidade de herança de funcionalidades. Em um sistema orientado para objetos, corretamente definido, espera-se que a correção de erros seja bem localizada, não perturbando todo o código-fonte. Por fim, uma linguagem orientada para objetos auxilia na otimização do sistema, facilitando a troca de segmentos de código ineficientes por outros otimizados e permitindo a especialização de funções, com o uso do polimorfismo.

A linguagem C++

A linguagem C++ é, atualmente, a mais utilizada na codificação de sistemas orientados para objetos. Em grande parte, sua aceitação decorre da popularidade da sua linguagem base: a linguagem C.

É longa uma discussão dos detalhes de sintaxe e semântica da linguagem C++. É grande também o número de críticas possíveis à forma como a linguagem C++ implementa os princípios da programação orientada para objetos [Joyn92]. Apesar disto, a diversidade e a qualidade das ferramentas de programação atualmente disponíveis para a implementação de programas em C++ supera o existente para todas as outras linguagens orientadas para objetos.

A correta compreensão da sintaxe e semântica da linguagem C++ é facilitada com o conhecimento prévio destes temas em relação à linguagem C. Uma análise da linguagem C, objetivando principalmente a portabilidade de código-fonte, pode ser encontrada em [RaSc90]. Uma apresentação da linguagem C++ pode ser encontrada em [AnAn91, Chri92, DeSt89, Joyn92, MSVC93, Sol93, Stev90].

Bibliotecas de classes

Um conjunto de novos tipos definidos em uma linguagem orientada para objetos pode ser desenvolvido

especialmente para comercialização, sendo este pacote denominado uma "biblioteca de classes".

Para programadores especialistas em linguagens orientadas para objetos, esta possibilidade abre um novo campo de trabalho. Enquanto isto, programadores que ainda não dominam os conceitos da programação orientada para objetos podem usar as classes de uma biblioteca como se fossem tipos pré-definidos da linguagem.

O próprio sistema **PHOTOPIX** pode ser considerado como uma biblioteca de classes. Procurou-se implementar uma classe que represente as imagens digitais, sendo que as funções implementadas para atuar sobre objetos desta classe formam uma interface para programação de algoritmos de processamento digital de imagens.

4 O ambiente operacional *Microsoft Windows*

O ambiente operacional *Microsoft Windows*, atualmente na sua versão 3.1 para o MS-DOS [Dunc86], pode ser classificado como um sistema gerenciador de janelas.

Os sistemas gerenciadores de janelas não têm, geralmente, a pretensão, nem a necessidade, de constituir um sistema operacional completo. Originalmente, sua única função era direcionar corretamente os fluxos de comunicação entre um usuário e diversos processos sendo executados concorrentemente por uma ou mais CPUs, locais ou remotas [HDF+86, Myer88, ScGe86]. Para isto, é necessário controlar o acesso de cada processo aos dispositivos de vídeo, áudio, *mouse*, teclado, impressora, caneta óptica, etc.

Diversas classificações são possíveis para os sistemas gerenciadores de janelas. Contudo, é comum que cada taxonomia considere apenas uma das duas interfaces que todos estes sistemas devem possuir:

- a interface com o usuário, que consiste no conjunto de escolhas relativas à aparência e ao *modus operandi* do ambiente operacional e seus aplicativos [Amar92, Msft92a, Msft92b, Myer88];
- a interface com o programador, onde são detalhadas as primitivas oferecidas pelo sistema aos desenvolvedores de programas e o conjunto de regras que devem ser seguidas na codificação de um aplicativo, para garantir o funcionamente correto de todo o sistema [Chri92, KoRa88, NoYa90, NyOR90, Petz92, Sun90].

A comparação de gerenciadores de janelas, considerando-se principalmente a interface com o usuário, baseia-se em critérios subjetivos, não podendo nortear a escolha de uma plataforma para desenvolvimento de aplicativos. Deve-se considerar ainda o fato de que há uma evolução contínua na interface com o usuário dos gerenciadores de janelas mais populares.

A escolha do *Microsoft Windows* como ambiente para o sistema **PHOTOPIX** deveu-se, principalmente, aos seguinte fatores:

- interface com o programador estável e amplamente documentada;
- funcionalidades gráficas pré-disponíveis;
- diversidade e disponibilidade de plataformas de *hardware* com as quais o *Microsoft Windows* é compatível, considerando também as possibilidade futuras [Cust93, Yao91];
- suporte internacional embutido no ambiente operacional;
- disponibilidade e diversidade de ferramentas de desenvolvimento.

Considerou-se também a evolução da aceitação do ambiente *Microsoft Windows*, que é o gerenciador de janelas que conquista o maior número de novos adeptos na atualidade.

A interface com o usuário do ambiente *Microsoft Windows* é plenamente apresentada em [Msft92a, Msft92b]. Conceitos e regras necessários à compreensão da API do sistema são apresentados em [Chri92, MSDK92, NoYa90, Petz92, Sol93].

A biblioteca de classes MFC

Para implementação do sistema **PHOTOPIX**, diversas funcionalidades teriam de ser codificadas, para atender a requisições que não são parte da "finalidade essencial" do programa [McPa84]. Optou-se pelo uso de uma biblioteca de classes comercial, tendo sido escolhida a biblioteca MFC (*Microsoft Foundation Class Library*) [Chri92, MSVC93].

Comparada a outras bibliotecas de classes, a biblioteca MFC apresenta os seguintes pontos positivos:

- além das classes básicas, implementando funcionalidades geralmente utilizadas em aplicativos comuns, há um subconjunto de classes específicas para suporte ao ambiente *Microsoft Windows*;
- portabilidade do código-fonte, que não depende de nenhuma extensão à sintaxe e semântica da linguagem C++.

5 Especificação do sistema PHOTOPIX

Para especificar o sistema **PHOTOPIX**, foi utilizada uma das mais modernas metodologias concebidas para especificação de sistemas informatizados: a "Análise Essencial" [McPa84, Your90]. Numa apresentação resumida, a Análise Essencial introduz poucos conceitos e regras a serem seguidas para obter a especificação de um sistema.

Análise essencial

No seu atual estágio de desenvolvimento, as plataformas de *hardware* utilizadas para implementação de sistemas informatizados contém dois componentes básicos: os "processadores", que executam atividades, e a "memória", onde são armazenados dados para uso dos processadores. A Análise Essencial define conceitos e regras que auxiliam o projetista de um sistema a evitar uma especificação repleta de referências às limitações das plataformas de *hardware* atualmente disponíveis.

Tecnologia perfeita

O conceito básico definido pela Análise Essencial é o de "tecnologia perfeita". Uma plataforma de *hardware* implementada com tecnologia perfeita teria um processador perfeito e uma memória perfeita.

Um processador perfeito daria à plataforma de *hardware* que o utilizasse a habilidade de realizar qualquer atividade, instantaneamente e sem nenhum gasto. Não haveria consumo de energia, erros, falhas, quebras, demoras ou qualquer outra restrição imposta devido à imperfeição das atuais tecnologias de construção de computadores.

As mesmas virtudes seriam válidas para uma memória perfeita, que não teria custo e daria à plataforma de *hardware* onde estivesse instalada a capacidade de armazenar uma quantidade infinita de dados, que poderiam ser acessados pelo processador randomicamente, sem qualquer atraso.

Deve-se atentar para o fato de que, numa plataforma de *hardware* implementada com tecnologia perfeita, poderiam ser executados algoritmos que ainda não possuem larga utilização prática justamente devido à sua ineficiência ou necessidade de enormes bases de dados. Exemplos seriam algoritmos para análise de imagens, reconhecimento de comandos de voz, inteligência artificial, etc. Assim, o uso de tecnologia perfeita poderia alterar profundamente a interface homem-máquina [Amar92] a que estão acostumados os atuais usuários da Informática.

Essência de um sistema

A diretriz a ser seguida na etapa de especificação, segundo os princípios da Análise Essencial, é a de encontrar a "essência" do sistema. A essência de um sistema que se utilize das tecnologias da Informática consiste no conjunto das suas "atividades essenciais" e na sua "memória essencial".

As atividades essenciais são os procedimentos que o sistema teria de realizar, mesmo que fosse possível implementá-lo utilizando tecnologia perfeita. A memória essencial consiste nos dados que teriam de ser obrigatoriamente armazenados, mesmo que o sistema executasse apenas as suas atividades essenciais.

Não se impõe limite à complexidade de cada atividade essencial de um sistema, visto que, com tecnologia perfeita, sua execução seria sempre instantânea. Não há também limite para o volume de dados que pode ser necessário armazenar na memória essencial, pois a plataforma de *hardware* que executaria o "modelo essencial" do sistema possuiria uma memória perfeita.

É possível distinguir dois tipos de atividades essenciais em um sistema: as atividades fundamentais e as custodiais.

Uma atividade essencial é dita fundamental quando ajuda a justificar a existência do sistema. Já as atividades custodiais são necessárias para estabelecer e manter a memória essencial.

Atividades custodiais usualmente dizem respeito à coleta de dados a serem armazenados na memória essencial. Contudo, deve-se observar que procedimentos como "ler arquivo para memória principal" não são atividades custodiais, visto que não são sequer atividades necessárias em um sistema implementado com tecnologia perfeita.

Essência do sistema PHOTOPIX

A memória essencial do sistema **PHOTOPIX** é constituída unicamente por "imagens". A sua única atividade essencial fundamental é executar algoritmos de PDI sobre estas imagens, gerando novas imagens que são também armazenadas na memória essencial. A Figura 1 apresenta o modelo essencial do sistema **PHOTOPIX**.

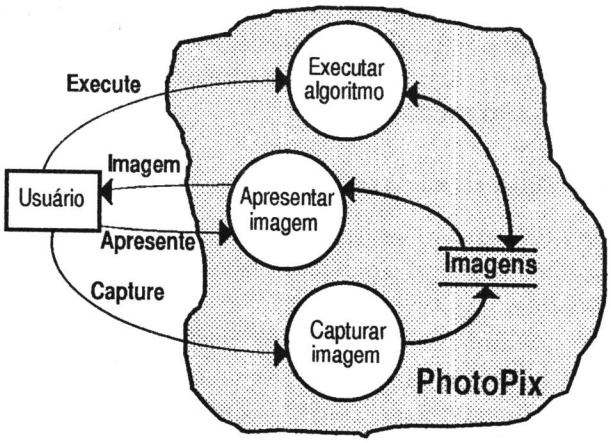

Figura 1 Modelo essencial do sistema PhotoPix

Duas atividades custodiais aparecem no modelo essencial. A atividade de "captura de imagens" é necessária para alimentar a memória essencial com imagens digitais. Idealmente, o sistema **PHOTOPIX**

obteria imagens diretamente de *scanners*, câmeras e outros aparelhos para aquisição de imagens digitais. Uma vez adquirida, uma imagem estaria indefinidamente disponível na memória essencial.

A outra atividade custodial necessária para o sistema é a que possibilita levar as imagens da memória essencial ao mundo externo. Atendendo a comandos do usuário, o sistema deve apresentar uma ou mais imagens em qualquer dos dispositivos de apresentação disponíveis na configuração da plataforma de *hardware* em uso (vídeo, impressora, etc.).

É digno de nota o fato de que, trocando-se o termo "imagem" pelo termo "dado" na Figura 1, obtém-se o primeiro nível de um modelo essencial genérico, válido para qualquer sistema de processamento eletrônico de dados [McPa84].

6 Codificação do sistema PHOTOPIX

O modelo essencial de um sistema permite um enorme número de implementações diferentes. Contudo, o uso de linguagens de programação inadequadas e o impacto da tecnologia imperfeita disponível nas atuais plataformas de *hardware*, pode fazer desaparecer no código-fonte do sistema a clareza das relações estabelecidas no modelo essencial.

Sistemas para PDI são particularmente susceptíveis aos impactos causados pela tecnologia imperfeita. Referências a configurações de *hardware* específicas impossibilitam o reaproveitamento da maior parte do código-fonte já desenvolvido. O uso da API do ambiente *Microsoft Windows* para implementação do sistema **PHOTOPIX** teve como intuito minimizar as referências ao *hardware*. A API do ambiente *Microsoft Windows* simula para os aplicativos uma "máquina virtual", ao custo da adição de uma camada a mais de código a ser executado.

Restrições tecnológicas

Obtido o modelo essencial de um sistema, devem ser determinadas as restrições tecnológicas a serem atendidas, permitindo assim que se obtenha um resultado útil em prazo pré-determinado.

As principais restrições tecnológicas presentes na implementação atual do sistema **PHOTOPIX** são:

- suporte apenas a imagens digitais codificadas no modelo *raster* [FVFH90, HeBa86]. Embora imagens digitais possam ser codificadas como uma série de vetores, este tipo de representação não é suportado no sistema **PHOTOPIX**, que armazena as imagens em um DIB (*Device Independent Bitmap*) [Petz92, Petz91];
- impossibilidade de aquisição de imagens diretamente do ambiente externo, assim como inviabilidade do armazenamento por tempo indefinido na memória essencial. O sistema obterá imagens digitais em arquivos, em uma limitada gama de formatos.

Identificando classes

A implementação de um sistema orientado para objetos inicia-se com a identificação dos novos tipos de dados (classes) a serem definidos, codificados isoladamente e depois utilizados no(s) programa(s) [Cox86, Mart93].

A partir do modelo essencial do sistema **PHOTOPIX** (Figura 1) é possível imediatamente identificar uma classe: a das imagens. As imagens são a matéria-prima e também o produto final do processamento digital de imagens. Portanto, constituem uma classe básica para qualquer sistema de PDI.

Também a partir do modelo essencial, é possível identificar que os algoritmos devem constituir uma classe. Os diferentes algoritmos seriam implementados como "derivações" da classe básica, sendo que, em "tempo de execução", seria usado o polimorfismo via funções virtuais para garantir a chamada ao código correto que implementa um algoritmo específico [Chri92, DeSt89, Sol93].

O uso da interface para múltiplos documentos (MDI: *Multiple Document Interface*) [Msft92b] é considerado vital para o sistema **PHOTOPIX**, pois a capacidade de armazenamento e processamento de um sistema de PDI é mal aproveitada quando o usuário não pode ver e comparar mais de uma imagem simultaneamente. Com o uso da biblioteca de classes MFC, um mínimo de codificação é necessário para implementar tal funcionalidade no sistema.

Na Figura 2, é apresentada a hierarquia das classes definidas para o sistema **PHOTOPIX**, seguindo-se a simbologia definida em [Mart93]. As classes específicas para lidar com a interface aplicativo-usuário no ambiente *Microsoft Windows* são derivadas de suas equivalentes funcionais na biblioteca MFC (alguns níveis na hierarquia da biblioteca MFC foram suprimidos para maior clareza do diagrama).

A classe **CPPixApp** controla a execução do aplicativo. A classe **CPPixFrame** fornece as funcionalidades da janela principal de um aplicativo MDI. A classe **CPPixView** implementa as funcionalidades das janelas-filhas de um aplicativo MDI, atuando como um visor através do qual é possível visualizar as imagens armazenadas na memória essencial, que são da classe **CImage**. A classe **CDIPAlgorithm** serve como base para derivação de classes que implementam algoritmos de PDI. A classe

CConverter serve como base para derivação de classes que implementam conversores entre a representação interna das imagens e os inúmeros formatos existentes para arquivos contendo imagens.

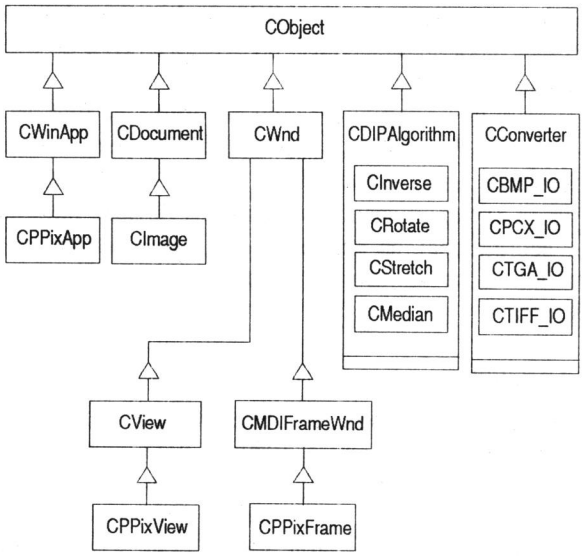

Figura 2 Hierarquia de classes do sistema **PHOTOPIX**

7 Interface com o usuário

Uma interface com o usuário de boa qualidade minimiza a ocorrência de erros, sejam eles devidos a modos de operação, representação, inconsistência, captura ou ativação [Amar92].

O ambiente *Microsoft Windows*, como a maioria dos gerenciadores de janelas da atualidade, segue os princípios de projeto de interface com o usuário estabelecidos no guia CUA (*Common User Access: Advanced Interface Design Guide*) [IBM89]. O sistema **PHOTOPIX** segue as recomendações de projeto para a interface com o usuário de aplicativos desenvolvidos para o ambiente *Microsoft Windows*, definidas em [Msft92b].

Janela principal

A janela principal do sistema **PHOTOPIX** possui todos os elementos comuns nos aplicativos que seguem o padrão MDI (Figura 3).

A janela principal pertence à classe **CPPixFrame**, enquanto as janelas-filhas pertencem à classe **CPPixView** (Figura 2). As janelas-filhas são recortadas (*clipping*) [Petz92] nos limites da janela principal.

Quadros de diálogo

Os quadros de diálogo do sistema **PHOTOPIX** foram elaborados segundo as regras de projeto de aplicativos para o sistema *Microsoft Windows*, especificadas em [Msft92b]. Um usuário proficiente do ambiente operacional *Microsoft Windows* está apto a utilizar qualquer dos quadros de diálogo do sistema **PHOTOPIX**.

Caixa de ferramentas (*ToolBox*)

Para facilitar a futura extensão das capacidades do sistema **PHOTOPIX**, foi implementada uma caixa de ferramentas (*ToolBox* [Msft92b]), controle não pré-definido no ambiente *Microsoft Windows* mas que tem recebido cada vez maior aprovação por parte de usuários de programas de "pintura" e PDI (Figura 3). Estando a parte relativa à interface aplicativo-usuário já implementada, os programadores que pretendam ampliar as funcionalidades disponíveis no sistema **PHOTOPIX** têm apenas de "interceptar" mensagens geradas pela caixa de ferramentas, "respondendo" de maneira adequada [Sol93].

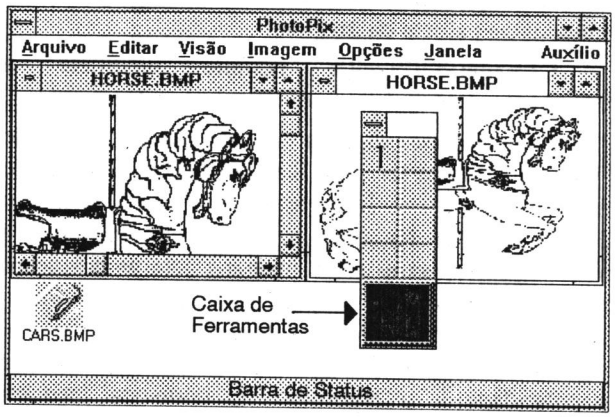

Figura 3 Janela principal do sistema PHOTOPIX

8 DIPAPI

O sistema **PHOTOPIX** define uma interface para programação de algoritmos de processamento digital de imagens, denominada DIPAPI (*Digital Image Processing Algorithm Programming Interface*).

A DIPAPI disponível no sistema **PHOTOPIX** é constituída pelas funções públicas da classe "CImage". O programador de algoritmos de PDI deverá familiarizar-se apenas com este número reduzido de funções, que implementam funcionalidades básicas necessárias em algoritmos de PDI:

CImage::GetBitsPerPixel(): retorna o número de *bits* utilizados para representar a informação espectral sobre cada *pixel* na imagem.

CImage::GetHeight(): retorna a dimensão vertical da imagem, sempre em *pixels*.

CImage::GetNearestPaletteIndex(): tendo como entrada uma cor no sistema RGB, retorna o índice na palheta da cor com menor distância euclidiana.

CImage::GetPaletteEntries(): possibilita obter as entradas na palheta da imagem.

CImage::GetPaletteSize(): retorna o número de entradas na palheta da imagem.

CImage::GetPixel(): retorna informação espectral sobre um *pixel* na imagem.

CImage::GetRGBHistogram(): retorna o histograma da imagem, separado em canais R, G e B.

CImage::GetWidth(): retorna a dimensão horizontal da imagem.

CImage::IsPaletted(): infoma se a imagem usa ou não uma palheta.

CImage::ResizeImage(): permite redimensionar a imagem, tanto em número de *pixels* na horizontal ou vertical, quanto no número de *bits* utilizados para representar a informação espectral.

CImage::SetPaletteEntries(): permite alterar as entradas na palheta da imagem.

CImage::SetPixel(): permite modificar a informação espectral sobre um *pixel* na imagem.

As imagens digitais armazenadas na classe "CImage" podem ter 1, 4, 8 ou 24 *bits* de informação espectral por *pixel*. Imagens de 1, 4 ou 8 *bits/pixel* podem ou não ter uma palheta. Para converter a informação espectral entre imagens com as diferentes resoluções espectrais aceitas pelo sistema, foram implementados os algoritmos de redução da informação espectral para uma palheta uniforme, o algoritmo da popularidade e o algoritmo do corte da mediana [Heck82], sendo consideradas as possibilidades de conversão para escala de tons de cinza e espalhamento do erro local. A ampliação do número de *bits/pixel* utilizados para representar a informação espectral também exige cuidado, principalmente em casos particulares que podem introduzir erros consideráveis na representação da imagem [Sol93].

No sistema **PHOTOPIX**, um algoritmo de PDI é classificado quanto ao seu número de entradas e metodologia de execução da seguinte forma:

- um operador local, que atua sobre uma ou mais imagens de entrada, gerando uma única imagem de saída, podendo ser invocado múltiplas vezes para gerar apenas determinada região da imagem destino (exemplos seriam os algoritmos da média, da mediana, dos k-vizinhos mais próximos, etc. [Jain89, Prat78]);
- um operador global, que atua sobre uma ou mais imagens de entrada, gerando também uma única imagem de saída mas atuando sobre toda a área das imagens de entrada (exemplos seriam algoritmos que dependem de medidas sobre a imagem original, como transformações no histograma, etc.);
- um macro-algoritmo, que pode ser considerado como uma seqüência de operadores locais e globais, cada um atuando sobre a(s) saída(s) de etapas anteriores. Através do mecanismo de macro-algoritmos torna-se possível que o usuário componha novos algoritmos a partir da base já implementada, além de permitir uma automatização do processamento de uma seqüência de imagens.

Para implementar um novo algoritmo de PDI para o sistema **PHOTOPIX**, o programador deve apenas "derivar" uma nova classe de **CDIPAlgorithm** (Figura 2), modificando uma ou mais de suas funções, de acordo com as regras de comunicação entre o sistema e os algoritmos de PDI, especificadas em [Sol93].

9 Conclusões

O sistema **PHOTOPIX** foi desenvolvido para servir como base, conceitual e prática, para sistemas de processamento digital de imagens. Sua especificação segue os princípios da "Análise Essencial" e sua arquitetura interna foi projetada segundo o paradigma da programação orientada para objetos.

Foi definida no trabalho uma API específica para a codificação de algoritmos de PDI, constituída pelas funções públicas da classe "CImage". Implementando-se a classe "CImage" em outros ambientes operacionais, garante-se o reaproveitamento dos algoritmos de PDI implementados utilizando esta DIPAPI.

O uso de técnicas de programação orientada para objetos permite que atualmente sejam adicionados ao sistema **PHOTOPIX**, simultaneamente e por equipes diferentes, novos algoritmos de PDI e funcionalidades na interface com o usuário. A utilização desta técnica impõe o acréscimo de uma camada a mais de *software* a ser executado, para implementar a comunicação entre as diversas classes que compõe o sistema. Contudo, a produtividade final alcançada, tanto pelo usuário do sistema como pelos programadores de algoritmos de PDI, justifica plenamente o uso da programação orientada para objetos, que não impõe aumento na complexidade dos algoritmos de PDI implementados no sistema, embora o tempo de execução seja multiplicado por uma constante.

10 Agradecimentos

Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq (processos 400.190/90-7, 500.908/91-5 e 130.990/90-6) e à Fundação de Amparo à Pesquisa do Estado de Minas Gerais - FAPEMIG (processo TEC 1113-90) pelo apoio financeiro a este trabalho.

11 Referências

[AnAn91] Anderson, G. & Anderson, P., Moving on to C++, *UnixWorld*, Jan. 1991

[Amar92] Amaral, L.M., *Interface Homem-Máquina do Ambiente de Definição de Semântica LDS*, Dissertação de Mestrado, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, 1992

[Chri92] Christian, K., *The Microsoft Guide to C++ Programming*, Microsoft Press, 1992

[Cox86] Cox, B.J., *Object-Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Inc., 1986

[Cust93] Custer, H., *Inside Windows NT*, Microsoft Press, 1993

[DeSt89] Dewhutst, S.C. & Stark, K.T., *Programming in C++*, Prentice-Hall, Inc., 1989

[Dunc86] Duncan, R., *Advanced MS-DOS*, Microsoft Press, 1986

[FVFH90] Foley, J.D., Van Dam, A., Feiner, S.K. & Hughes, J.F., *Computer Graphics: Principles and Practice, 2nd. Ed.*, Addison-Wesley, Inc., 1990

[HDF+86] Hopgood, F.R.A., Duce, D.A., Fielding, E.V.C., Robinson, K. & Williams, A.S., *Methodology of Window Management*, Springer-Verlag, 1986

[HeBa86] Hearn, D. & Baker, M.P., *Computer Graphics*, Prentice-Hall, Inc., 1986

[Heck82] Heckbert, P., Color image quantization for frame buffer display, *Computer Graphics*, Vol. 16, No. 3, Jul. 1982

[IBM89] IBM Co., *Common User Access: Advanced Interface Design Guide*, IBM Co. (SC26-4582-0), 1989

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

[Joyn92] Joyner, I., *A C++ Critique*, e-mail correspondence, 1992

[KaLe92] Kay, D.C. & Levine, J.R., *Graphics File Formats*, Windcrest Books, 1992

[KoRa88] Kogan, M.S. & Rawson III, F.L., The design of Operating System/2, *IBM Systems Journal*, Vol. 27, No. 2, 1988

[Mart93] Martin, J., *Principles of Object-Oriented Analysis and Design*, Prentice-Hall, Inc., 1993

[McPa84] McMenamin, S.M. & Palmer, J.F., *Essential Systems Analysis*, Prentice-Hall, Inc., 1984

[MSDK92] Microsoft Co., *Microsoft Windows 3.1 Software Development Kit*, Microsoft Press, 1992

[Msft92a] Microsoft Co., *Microsoft Windows 3.1 - User's Guide*, Microsoft Co., 1992

[Msft92b] Microsoft Co., *The Windows Interface - An Application Design Guide*, Microsoft Press, 1992

[MSVC93] Microsoft Co., *Microsoft Visual C++ compiler manuals*, Microsoft Co., 1993

[Myer88] Myers, B.A., A Taxonomy of Window Manager User Interfaces, *IEEE Computer Graphics & Applications*, Sep. 1988

[NoYa90] Norton, P. & Yao, P., *Windows 3.0 Power Programming Techniques*, Bantam Books, 1990

[NyOR90] Nye, A. & O'Reilly, T., *X Toolkit Intrinsics Programming Manual - OSF/Motif Edition*, O'Reilly & Associates, 1990

[Petz91] Petzold, C., Preserving a Device-Independent Bitmap: The Packed-DIB Format, *PC Magazine*, Vol. 10, No. 13, Jul. 1991

[Petz92] Petzold, C., *Programming Windows 3.1*, Microsoft Press, 1992

[Prat78] Pratt, W., *Digital Image Processing*, John Wiley & Sons, 1978

[RaSc90] Rabinowitz, H. & Schaap, C., *Portable C*, Prentice-Hall, Inc., 1990

[ScGe86] Scheifler, R.W. & Gettys, J., The X Window system, *ACM Trans. on Graphics*, Vol. 5, No. 2, Apr. 1986

[Sol93] Sol, A.A.S., *PhotoPix: uma plataforma para sistemas de processamento digital de imagens orientada para objetos*, Dissertação de Mestrado, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, 1993

[Stev90] Stevens, R.T., *Fractal Programming and Ray Tracing with C++*, M&T Books, 1990

[Sun90] Sun Microsystems, Inc., *Writing Applications for Sun Systems - A Guide for Macintosh Programmers*, Addison-Wesley, Inc., 1990

[Ulic88] Ulichney, R., *Digital Halftoning*, MIT Press, 1988

[Yao91] Yao, P., Windows 32-bit API gives developers advanced operating system capabilities, *Microsoft Systems Journal*, Nov-Dec. 1991

[Your90] Yourdon, E., *Análise Estruturada Moderna*, Editora Campus, 1990