Ambiente de suporte ao ensino de processamento de imagens usando a linguagem Python

Alexandre Gonçalves Silva[1], Roberto de Alencar Lotufo[1],
Rubens Campos Machado[2]

[1]DCA-FEEC – Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6101 – 13081-970 – Campinas – SP – Brasil
{alexgs, lotufo}@dca.fee.unicamp.br

[2]Centro de Pesquisas Renato Archer (CenPRA)
Caixa Postal 6162 – 13082-120 – Campinas – SP – Brasil
rubens.machado@cenpra.gov.br

Resumo. Este trabalho consiste no estudo, desenvolvimento e implementação de uma caixa de ferramentas de processamento de imagens a partir da linguagem *Python* e pacote *Numerical Python*. Este conjunto segue a linha "Open Source" de distribuição sendo bastante adequado ao processamento matemático multidimensional. Python é uma linguagem moderna, interpretada, de "altíssimo nível" e excelente portabilidade, orientada a objetos, bem projetada e apropriada ao desenvolvimento rápido de aplicações (RAD). O ambiente de ensino é gerado, em grande parte, usando a metodologia de construção de *software* de computação científica do projeto *Adesso*. Além de útil ao ensino, este trabalho pode também ser utilizado tanto em pesquisa como no desenvolvimento de aplicações finais.

Abstract. This work consist in the study, development and implementation of a toolbox for image processing with the *Python* language and the *Numerical Python* package. This set is based on "Open Source" disbribuition license and is suitable for multidimensional mathematical processing. Python is a modern language, interpreted, "very high level" and extremely portable, object oriented, well projected and suitable for rapid application development (RAD). The system was generated using the methodology of the Adesso project. This environment is useful in education, research and development of final applications.

Palavras Chave: Processamento de imagens, caixa de ferramentas, linguagem Python.

1. Introdução

A análise de imagens digitais de forma automática ou semi-automática apresenta considerável importância na solução de problemas em diversas áreas como geologia, medicina, robótica, química, artes, entre outras. Exemplificando, poderíamos ter a caracterização de forma de um tumor a partir do processamento de imagens de uma tomografia computadorizada.

Existem diversas ferramentas computacionais apropriadas ao processamento de imagens, tanto comerciais como gratuitas, como *Khoros*, *MATLAB* [Eddi94], *Mathematica*, *VisiLog*, *PV-WAVE*, *AVS*, *IDL*, *ImageMagik*, e o esforço nacional *PhotoPixJ* [Alme96], entre outras [Robi00]. O DCA-FEEC-Unicamp adotou o Khoros, entre 1991 e 1997, como uma ferramenta de programação visual aos cursos de processamento de imagens. Neste período, foi desenvolvido o *DIPCOURSE* [Jord96, Lotu96], primeiro curso de processamento de imagens disponível na internet. Porém, o grupo de desenvolvimento do Khoros não foi capaz de acompanhar o avanço tecnológico de software

e, ao mesmo tempo, deixou de ser gratuito. Como conseqüência, o MATLAB[1] passou a ser usado preferencialmente nos cursos de processamento de imagens. O MATLAB é um ambiente genérico de computação científica muito difundido nos cursos de Engenharia, tem uma linguagem matricial extremamente simples, compacta, muito próxima da linguagem matemática e possui uma grande comunidade de usuários. Tem como maior desvantagem, entretanto, o seu alto custo. Poucas instituições têm condições de adquiri-lo. Uma outra desvantagem é a falta de ferramentas de autoria de caixas de ferramentas. Apesar de ser muito fácil criar um conjunto de funções voltadas a um problema, não existe nenhum suporte para a geração de documentação ou gerenciamento das implementações.

É intenção do grupo desenvolver, em longo prazo, um ambiente computacional que contenha o melhor das duas plataformas. Neste sentido já existe o esforço associado ao projeto Adesso [Mach02,Mach00], em cooperação com o CenPRA que consiste de um ambiente de autoria de software de computação científica. O estágio atual do projeto, todo baseado na tecnologia XML [w3c00] contempla o suporte para geração de documentação e de código de interface de bibliotecas computacionais. Existe hoje o suporte para interface automática para as plataformas Python, MATLAB e *Tcl/Tk* [Welc99].

Recentemente, a linguagem Python [Ross02,Lutz98] vem mostrando um crescimento muito grande, principalmente na comunidade acadêmica, como sendo uma linguagem de *script* bastante moderna, com os conceitos de orientação a objetos e extensibilidade muito apurados. Paralelamente, há o pacote *Numerical* [Asch01] que apresenta um conjunto de módulos facilmente incorporados ao Python, oferecendo um suporte de computação científica dentro de um modelo de matriz multidimensional. Estes módulos foram projetados sob forte influência das funções básicas do MATLAB. Desta forma, está se tornando muito atrativo o uso do Python, associado a este pacote numérico e ao ambiente Adesso de autoria de software de processamento científico. Acreditamos que seja possível dar um grande passo na direção de conseguir um sistema que agregue as vantagens dos sistemas Khoros e MATLAB.

[1] http://www.mathworks.com

O objetivo principal deste trabalho é desenvolver um ambiente de suporte à implementação de algoritmos para um curso de processamento de imagens. A intenção é a de se ter um conjunto de ferramentas totalmente gratuito, de fácil instalação, multiplataforma e, sobretudo, de rápida aprendizagem e com programação simplificada. E ainda, baseado em Python, uma linguagem genérica bem projetada que oferece suporte nativo às mais diversas aplicações. O ambiente implementado não se restringe ao ensino e pode ser muito bem aproveitado em pesquisa e no desenvolvimento de aplicações finais.

Este trabalho está organizado da seguinte forma. A Seção 0 ilustra modelos de programação de algoritmos de processamento de imagens. A Seção 0 descreve as ferramentas de desenvolvimento, a forma como se dá a produção de software e, enfim, os sistemas gerados para o suporte ao ensino de processamento de imagens. A Seção 0 mostra os principais resultados obtidos, e as conclusões são apresentadas na Seção 0.

2. Modelo de programação

Para cada linguagem de programação, é natural que haja uma série de vantagens e desvantagens [Prec00]. Devemos analisar os propósitos de uma dada aplicação para adoção de uma ferramenta em particular. Uma linguagem de sistema pode ser suficientemente eficiente conforme o grau de otimização que os programas sofrem na compilação. Considerando eficiência, a programação em linguagem assembly seria uma ótima opção, mas demanda muito tempo e paciência na codificação. Por outro lado, programas extremamente simples, em poucas linhas e com eficiência computacional compatível ao assembly podem ser elaborados em linguagens de mais alto nível. Linguagens interpretadas como Python ou MATLAB têm excelente suporte a operações matriciais (condição esperada em um sistema de processamento de imagens) e são apropriadas para construção de protótipos, pois nelas, em geral, o tempo de desenvolvimento de programas é consideravelmente mais curto. Porém, muitas vezes, não se ajustam a implementações que necessitam de iteração excessiva. A Tabela 1 compara os tempos para o cálculo do fatorial, em C, Python e MATLAB, de cem mil elementos.

O Teste 1 ilustra a ineficiência em se usar laços iterativos em linguagens interpretadas. No Teste 2, tanto em Python como em MATLAB, para cada elemento de valor *v*, é criado um vetor com inteiros entre *2* e *v* (*Numeric.arange(2,v+1)* em Python; *2:v* em MATLAB) que são multiplicados entre si por um operador próprio da linguagem (*Numeric.product* em Python; *prod* em MATLAB). Observa-se que o tempo resultante é consideravelmente reduzido neste caso.

	Teste 1	Teste 2
C	1,0 s	1,5 s
Python	33,4 s	8,6 s
MATLAB	151,1 s	2,6 s

Tabela 1 - Comparação de tempos médios (Sun Ultra 60) para o cálculo do fatorial de 10^5 números. O Teste 1 utiliza-se de laços iterativos acumulando o valor do fatorial, enquanto o Teste 2 implementa a multiplicação de todos elementos ordenados em um vetor.

Uma ferramenta computacional para processamento de imagens genérica usualmente oferece dois níveis de programação, associando facilidade de implementação e desempenho. Procura-se ter então uma linguagem interpretada de *script* servindo de interface a uma biblioteca de alto desempenho escrita em uma linguagem de mais baixo nível como C ou Fortran. Porém, normalmente a demanda de ferramentas diversas de processamento de imagens é muito grande, sendo praticamente impossível oferecer um sistema completo ao usuário.

Dada a facilidade de programação, poderia ser interessante que a maioria dos algoritmos solicitados em uma disciplina de processamento de imagens fosse implementada diretamente em linguagens interpretadas de forma eficiente. Devemos, para isto, evitar, sobretudo, programas com iteração explícita. A idéia é proceder indiretamente toda iteração necessária, sempre que possível, através de manipulações matriciais. A Figura 7 mostra um exercício inicial, neste sentido, para se gerar uma imagem em forma de xadrez. Observe que tanto o operador "+" (soma) como o operador "%" (resto da divisão inteira) devem estar presentes na linguagem e se encarregar de fazer o cálculo elemento a elemento (pixel a pixel). As matrizes **x** e **y** podem ser facilmente e eficientemente construídas Python (*Numeric.indices* em Python; *meshgrid* em MATLAB). Desta mesma forma poderíamos sintetizar uma imagem de círculo, logaritmo, ou cossenóide 2-D, entre outros exemplos.

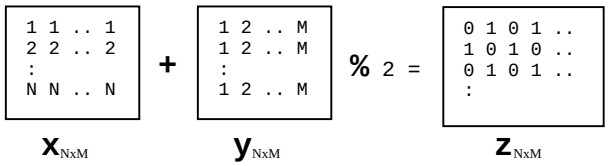


Figura 7 - Exemplo para se gerar uma imagem em forma de xadrez por manipulação matricial: **z=(x+y)%2**.

Outra abordagem poderia ser uma certa operação de vizinhança dada uma máscara de convolução. Neste caso, é mais eficiente transladar a imagem em todas as posições da máscara e calcular a soma acumulada do produto destas translações por cada pixel da máscara. A Tabela 2 mostra tempos calculados para duas implementações, em Python, de um filtro por convolução. No Teste 1, é feita uma varredura pixel a pixel com a utilização de quatro laços iterativos (dois para a imagem e dois para a máscara). Já, no Teste 2, são feitas translações na imagem e são usados dois laços iterativos apenas para a máscara. Observa-se a diferença expressiva de tempo entre os métodos. Em se tratando do uso de linguagens interpretadas, é importante ter em mente este modelo de programação, no qual um operador, sempre que houver, possa ser aplicado a um grande conjunto de dados em uma única vez. Na verdade, este ganho em eficiência vem do fato destes operadores serem implementados em linguagens de mais baixo nível.

Máscara	Teste 1	Teste 2
3x3	22,41 s	0,05 s
5x5	61,23 s	0,13 s
7x7	119,07 s	0,26 s

Tabela 2 - Tempos calculados (Pentium 4-1.5GHz) para duas implementações diferentes de um filtro de média sobre uma imagem de 256x256.

3. Ambiente de desenvolvimento

Nesta seção, apresentamos a linguagem de programação Python, um de seus pacotes de processamento matemático e aspectos de exibição de imagens e gráficos. Em seguida, é visto o sistema Adesso de autoria de *software* e a caixa de ferramentas implementada a partir do mesmo. E, por último, é apresentado um ambiente de correção automática de *scripts*.

3.1 A linguagem Python

Python é uma linguagem interpretada moderna, orientada a objetos, de muito alto nível, de rápida prototipação, com sintaxe simples (quase como um "pseudo-código"), extensível (*C/C++*), e extremamente portável. A linguagem é bastante flexível (assim como *Perl*) e é livremente distribuída, além de ter seu código-fonte aberto. Pode ser facilmente instalada a partir de seu *site*[2]. É suficientemente eficiente para utilização em processamento de imagens quando associado ao pacote Numerical. Este também segue a linha "Open Source" e pode ser adquirido livremente[3]. A combinação Python e Numerical permite um poder numérico e facilidade de uso similar ao MATLAB.

A exibição de imagens é possível através do uso do módulo *Tkinter*, uma interface, orientada a objetos, para as funcionalidades do pacote de interfaces gráficas Tk. A Figura 8 mostra um *script* e saídas resultantes da chamada da função "iashow" para exemplificar a exibição de imagens[4].

```
>>> import Numeric
>>> from ia636 import *
>>> f =
iaread('cameraman.pgm')
>>> (g,a) = iasobel(f)
>>> iashow(f)
>>> iashow(Numeric.log(g+1))
```

(a)

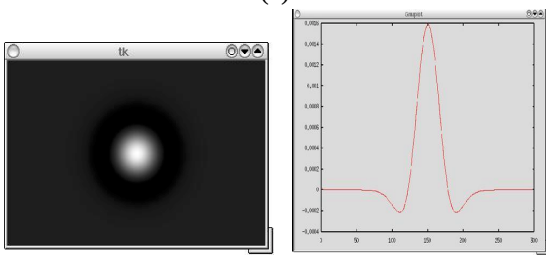
(b) (c)

Figura 8 - (a) *Script* de teste de exibição de imagens. (b) Imagem `f` mostrada pelo primeiro "iashow" do *script*. (c) Imagem do logaritmo de `g+1` do segundo "iashow".

Quanto à exibição de gráficos, existem várias ferramentas que podem ser localizadas a partir do *site* do Python. Foi escolhido o *gnuplot* por ser bastante poderoso. A interface com o Python, o *Gnuplot.py*[5], é simples de ser instalada e pode ser utilizada em várias plataformas. A Figura 9 mostra um *script* e suas saídas correspondentes. Nota-se, no *script* da Figura 9a, a exibição do gráfico da Figura 9c pelo uso da função "iaplot" sobre uma linha horizontal da matriz da imagem "h" (mostrada pelo "iashow" na Figura 9b).

```
>>> h = ialog([200, 300], [100, 150],
20)
>>> iashow(h)
>>> iaplot(h[100, :])
```

(a)



(b) (c)

Figura 9 - (a) *Script* de teste de exibição de imagem e gráfico. (b) Imagem `h` mostrada pelo "iashow". (c) Gráfico da linha 101 da imagem `h` mostrada pelo "iaplot".

3.2 O sistema Adesso

O Adesso é um projeto conjunto entre a Fundação CenPRA e a FEEC-Unicamp que explora o modelo de programação baseado em componentes reutilizáveis. Esta abordagem fornece suporte ao desenvolvimento de componentes e sua integração a diversas plataformas de programação científica. O Adesso configura-se em uma base de dados de componentes (algoritmos) representada em XML e em um conjunto de ferramentas de transformação (*stylesheets*) para a geração de código, documentação e empacotamento. A

Figura **10** mostra o esquema de como esta transformação ocorre. Há uma base de dados XML, de entrada, contendo informações quaisquer tais como algoritmos C, Python e meta-informações. Há um conjunto de folhas de estilo, cada qual para uma saída específica que pode ser código-fonte C ou Python, documentação LaTeX, HTML, entre inúmeras outras possibilidades. O processador de estilos do Adesso se encarrega de ler as duas entradas (base de dados da caixa de ferramentas e a folha de estilo) e gerar a saída solicitada. Este processador de estilos é implementado através do tDOM [Joch99], um eficiente pacote para Tcl que facilita o processamento XML/DOM.

[2] http://www.python.org

[3] http://www.pfdubois.com/numpy

[4] O prefixo "ia" das funções vem da sigla da disciplina de Visão Computacional da Unicamp (IA636).

[5] http://gnuplot-py.sourceforge.net

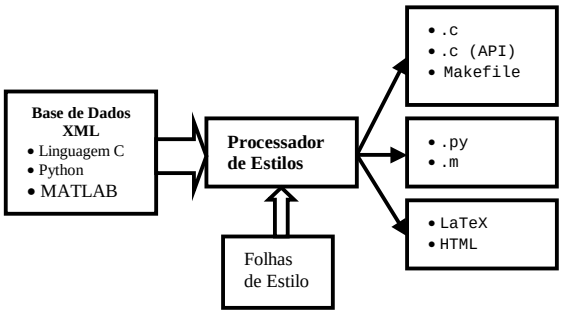


Figura 10 - Esquema de transformação do sistema Adesso.

3.2.1 Organização XML

Os arquivos XML formam uma base de dados contendo algoritmos e suas descrições, parâmetros de entrada e saída e suas descrições, exemplos, equações, testes, indicações de dependências, entre outros campos. Todas estas informações são armazenadas em formato texto plano. As equações, por exemplo, seguem a notação LaTeX. A Figura 11 mostra um trecho de código XML.

```
<AdFunction lang="python matlab">
<Documentation>
  <Descr name="Description">Generate a cosenosoid image of size s with
  amplitude 1, period T, phase phi and wave direction of theta. The
  output image is a double array.</Descr>
  <Descr name="Examples">
    <PYcode>import Numeric
    f = <Self/>([128,256], 100, Numeric.pi/4, 0)
    iashow(ianormalize(f, [0,255]))</PYcode>
  </Descr>
  <Descr name="Equation">
  <Eq>f(x,y) &amp;amp; = &amp;amp; sin( 2\pi (f_x x + f_y y) + \phi) \\
  ...
  </Eq>
  </Descr>
</Documentation>
<Return name="f" type="mmIMAGE" output="yes">
<Args>
  <Arg name="s" type="mmIMAGE" dir="in" optional="no" default="">
    <Descr>size: [rows cols].</Descr>
  </Arg>
  ...
</Args>
<Source lang="python">
  <Code>cols, rows = s[1], s[0]
  x, y = iameshgrid(range(cols),range(rows))
  freq = 1./t
  fcols = freq * Numeric.cos(theta)
  frows = freq * Numeric.sin(theta)
  f = Numeric.cos(2*Numeric.pi*(fcols*x + frows*y) + phi)</Code>
</Source>
…
<Platforms>windows linux sunos</Platforms>

</AdFunction>
```

Figura 11 - Exemplo de estrutura XML.

3.2.2 Geração de código

Uma das folhas de estilo implementada foi a de geração de código-fonte Python. Esta se encarrega de percorrer toda a base de dados XML, reconhecer os campos necessários, transformando-os em código Python. Por exemplo, assim que todos os parâmetros de entrada de uma determinada função são reconhecidos o protótipo (ou cabeçalho) desta função é montado. No momento em que toda a descrição e exemplos são lidos, pode-se montar um trecho de comentários no código-fonte que, por sua vez, constituirá uma ajuda quando no uso da caixa de ferramentas. O código-fonte propriamente dito é simplesmente copiado e, finalmente, o retorno de cada função é feito através do reconhecimento dos parâmetros de saída. A Figura 12 mostra um exemplo de código-fonte gerado automaticamente a partir da estrutura XML da Figura 11.

```
def iacos(s,t,theta,phi):
    """  o Purpose
      Create a cossenoidal image.
  o Synopsis
      f = iacos(s,t,theta,phi)
  o Input
      s: size: [rows cols].
      t: Period: in pixels.
      theta: spatial direction...
      phi: Phase
  ...
  o Examples
      f = iacos([128,256], 100, Numeric.pi/4, 0)
      iashow(ianormalize(f, [0,255]))
    """
    cols, rows = s[1], s[0]
    x, y = iameshgrid(range(cols),range(rows))
    freq = 1./t
    fcols = freq * Numeric.cos(theta)
    frows = freq * Numeric.sin(theta)
    f    =    Numeric.cos(2*Numeric.pi*(fcols*x    +
 frows*y) + phi)
    return f
```

Figura 12 - Exemplo de código-fonte Python gerado automaticamente.

3.2.3 Geração de documentação

A folha de estilo para geração de documentação segue o mesmo princípio descrito na subseção anterior. As novidades aqui vêm do fato da necessidade em se gerar figuras e gráficos. Por exemplo, quando há uma demonstração na descrição de uma função, o Adesso invoca o interpretador Python que executa o *script* contendo o exemplo, gerando a saídas correspondentes aproveitadas no conteúdo da documentação. Há basicamente quatro tipos de saídas: em forma de texto simples (*string*); saídas numéricas; imagens; e gráficos. Há também o campo de equações, onde é a vez do interpretador LaTeX ser invocado, gerando uma imagem de fórmulas. Observe que é necessário manter apenas as informações textuais para produzir toda uma documentação ilustrada. Evidentemente que também é mantido um conjunto básico de imagens fotográficas, mas todas as ilustrações resultantes da aplicação de alguma função são criadas quando da geração da documentação. A Figura 7 mostra um exemplo de documentação HTML gerada automaticamente. A função “iagaussian” foi escolhida, desta vez, por apresentar quase todos os elementos possíveis de saída (numérica, gráfica, imagem, equação e textos).

[Top] [Up] [Prev] [Next] [Up] [Image Processing Co

iagaussian - Generate a 2D Gaussian image.

Synopsis `g = iagaussian( s, mu, sigma )`

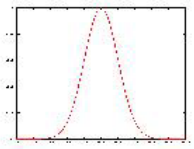
Input `s`: [rows columns]
`mu`: Mean vector. 2D point (x;y). Point of maximum value.
`sigma`: Covariance matrix (square). [Sx^2 Sxy; Syx Sy^2]

Output `g`: Gray-scale (uint8 or uint16).

Description A 2D Gaussian image is an image with a Gaussian distribution. It can be used to generate test patterns or Gaussian filters both for spatial and frequency domain. The integral of the gaussian function is 1.0.

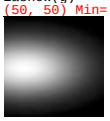
Examples

```
import Numeric
f = iagaussian([8,4], [3,1], [[1,0],[0,1]])
g = ianormalize(f, [0,255]).astype(Numeric.UnsignedInt8
print g
[[  1   2   1   0]
 [ 20  34  20   4]
 [ 93 154  93  20]
 [154 255 154  34]
 [ 93 154  93  20]
 [ 20  34  20   4]
 [  1   2   1   0]
 [  0   0   0   0]]
f = iagaussian(100, 50, 10*10)
g = ianormalize(f, [0,1])
g,d = iaplot(g)
```



(g)

```
f = iagaussian([50,50], [25,10], [[10*10,0],[0,20*20]])
g = ianormalize(f, [0,255]).astype(Numeric.UnsignedInt8
iashow(g)
(50, 50) Min= 0 Max= 255 Mean=83.974 Std=72.81
```



(g)

Equation

1D

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Multidimensional

$$f(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} exp\left[-\frac{1}{2}(\mathbf{x}-\mu)^t \Sigma^{-1}(\mathbf{x}-\mu)\right]$$

See Also iacos - Create a cossenoidal image.
ialog - Laplacian of Gaussian image.

Source code iagaussian.py

Figura 13 - Documentação HTML gerada automaticamente.

A importância da geração das figuras desta forma é manter uma consistência entre a documentação e a implementação das funções. É comum encontrarmos documentações completamente defasadas da última versão das implementações das funções do pacote computacional. Além da documentação padrão para cada função também é possível gerar lições ou demonstrações bastando, para isto, definir uma estrutura XML particular.

3.2.4 Geração de testes (*testsuites*)

A última tarefa do gerador de código é a de criar *scripts* Python de teste de toda caixa de ferramentas implementada. As informações de teste também estão armazenadas em XML juntamente com as funções. Os *testsuites* são exemplos que comparam as saídas das funções com valores corretos, gerado na maioria das vezes, analiticamente. Com isto, espera-se cobrir o máximo de casos de erros de implementações e detectar qualquer alteração de resultados quando há uma atualização da caixa de ferramentas.

3.3 Caixa de Ferramentas

De posse do sistema Adesso, ajustado ao desenvolvimento em Python, foi possível criar todas as funções, lições, demonstrações e testes da caixa de ferramentas [Silv01,ia636], além de gerar toda a documentação correspondente. São mais de 60 algoritmos implementados. A Figura 14 mostra o ambiente Python e um exemplo simples de uso da caixa de ferramentas implementada (ia636).

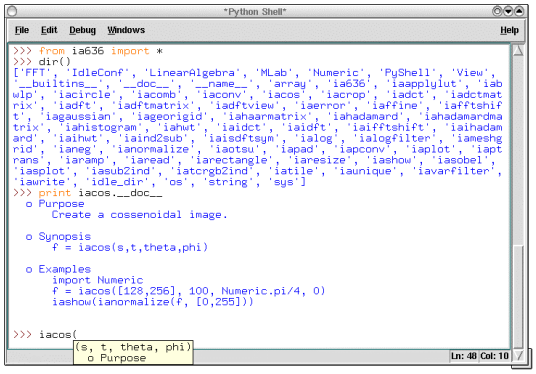


Figura 14 - Exemplo de uso da caixa de ferramentas dentro do interpretador Python.

3.4 Corretor automático

Com auxílio da caixa de ferramentas construída e das funcionalidades nativas do Python para implementação de CGI (Common Gateway Interface) foi possível desenvolver um sistema para processar entregas de exercícios, em Python, dos alunos pela internet. Em uma página de submissão, é solicitada a entrega de um módulo (um arquivo) contendo uma ou mais funções, sendo os protótipos de tais funções precisamente definidos por um enunciado. Optativamente também pode ser entregue um relatório (em formato *txt*, *ps*, *pdf* ou *html*). Este sistema se divide em duas modalidades:

1) Correção. Vários testes, dado por um *script* elaborado pelo professor, são feitos sobre o módulo entregue e os resultados obtidos são comparados com os resultados do processamento do "módulo gabarito". A Figura 15 ilustra este processo. A caixa com o símbolo "=" se encarrega de comparar todas as *n* variáveis (*n* testes) em memória, V_1, V_2, ... V_n, definidas pelo *script* de testes, entre os

resultados da submissão $V_i^{(a)}$ e do gabarito $V_i^{(b)}$ (para cada *i* entre *0* e *n*). As saídas são os resultados das comparações. É exibido, para cada variável, se o valor está correto, incorreto ou se houve falha. O interesse aqui é verificar a exatidão do algoritmo do aluno.

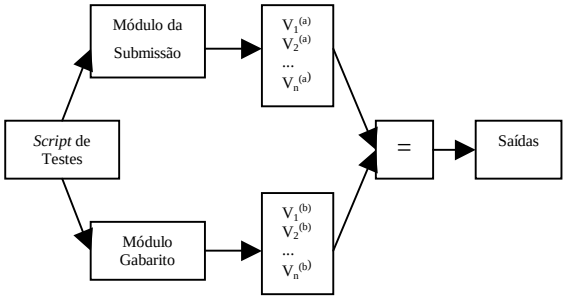


Figura 15 - Esquema do corretor automático.

2) Visualização. Nesta modalidade, não é necessária a implementação de um gabarito. São feitos testes sobre o módulo submetido, da mesma forma, e uma página HTML, com saídas numéricas ou imagens, é gerada automaticamente e armazenada como resultados de cada aluno, conforme especificações feitas no *script* de testes. A Figura 16 ilustra o ambiente de interação inicial de submissão dos exercícios e alguns resultados gerados neste sentido. O interesse aqui poderia ser, entre outros, o de verificar a qualidade ou forma de uma imagem produzida por um algoritmo, um índice de correlação ou de erro.

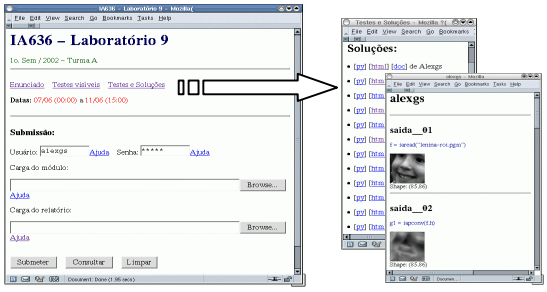


Figura 16 - Ambiente de submissão (à esquerda) e visualização de alguns resultados (à direita).

Para o professor há ainda uma opção de consulta detalhada dos resultados que pode ser acessada a qualquer tempo (os alunos não têm esta permissão antes do prazo de entrega). Além do código-fonte, relatório, resultados ilustrativos em HTML ou histórico de testes realizados, o professor também pode visualizar os tempos de execução de cada função submetida, aluno por aluno, permitindo, desta forma, uma análise de eficiência das implementações.

4. Resultados

A caixa de ferramentas IA636 construída está sempre incorporando novas funcionalidades. Seu estado atual é descrito a seguir.

Foram implementadas diversas lições:

- Geração de imagens sintéticas diversas (iagenimages);
- Função de transformação de contraste (iait);
- Equalização de histograma (iahisteq);
- Técnica de casamento de padrão (iacorrdemo);
- Demonstração do espectro de Fourier para imagens sintéticas simples (iadftexamples);
- Propriedade de escala da Transformada de Discreta de Fourier e caracterização da forma da matriz núcleo desta transformação (iadftscaleproperty e iadftmatrixexamples);
- Decomposição de imagens em ondas primitivas 2D (iadftdecompose);
- Interpolação de imagens aumentadas (iamagnify);
- Restauração pela filtragem inversa (iainversefiltering);
- Transformada de Hotelling (iahotelling);
- Ilustração do método de seleção de *thresholding* de Otsu (iaotsudemo).

Foram implementadas diversas funções divididas nos seguintes tópicos:

- Criação de imagem: filtro passa-baixo de Butterworth (iabwlp), círculo (iacircle), impulsos (iacomb), cossenoidal 2-D (iacos), gaussiana 2-D (iagaussian), Laplaciano da Gaussiana (ialog), bandas verticais com intensidade crescente (iaramp), retângulo (iarectangle);
- Manipulação e informação: recorte do retângulo mínimo de uma imagem (iacrop), conversão de índices lineares para x-y e vice-versa (iaind2sub e iasub2ind), criação de malha 2-D de índices (iameshgrid), negação (ianeg), normalização de intensidades (ianormalize), acréscimo de borda (iapad), replicação matricial até uma

nova dimensão (iatile);

- Arquivo de imagens: leitura (iaread) e gravação (iawrite);
- Manipulação de contraste: transformação de intensidades (iaapplylut);
- Processamento de cor: *true color* RGB para imagem de índices e tabela de cores (iatcrgb2ind);
- Manipulações geométricas: transformação afim (iaffine), transformação e escala de corpo rígido 2-D (iageorigid), translação periódica (iaptrans), redimensionamento (iaresize);
- Transformações: Transformada Discreta de Cossenos (iadctmatrix e iadct), Transformada Discreta de Fourier (iadftmatrix e iadft), Transformada Wavelet de Haar (iahaarmatrix e iahwt), Transformada de Hadamard (iahadamardmatrix e iahadamard) e, respectivamente, suas inversas (iaidct, iaidft, iaihwt e iaihadamard), além de funções de translação de espectro da componente zero (iafftshift e iaifftshift), e verificação de simetria conjugada (iaisdftsym);
- Filtragem: filtro do Laplaciano da Gaussiana (ialogfilter), convolução linear e periódica 2-D (iaconv e iapconv), detecção de borda Sobel (iasobel), filtro de variância (iavarfilter);
- Técnica de *thresholding* automático: Thresholding de Otsu (iaotsu);
- Medidas: histograma (iahistogram);
- Visualização: da transformada de Fourier (iadftview), exibição de gráficos (iaplot), exibição de imagens (iashow), exibição de superfícies 3-D (iasplot);
- Funções de suporte: devolução do conjunto único ordenado, sem repetição, a partir de um vetor ou matriz de entrada (iaunique) e exibição de mensagens de erro (iaerror).

Dada uma base de dados XML no Adesso, acrescentar o suporte à outra linguagem significa basicamente incluir as informações de sintaxe em trechos onde há código-fonte, além de implementar as folhas de estilo para a interpretação do novo contexto. As demais informações como parâmetros de entrada e saída, descrições gerais das funções e de seus parâmetros, são compartilhadas. Esta característica facilita a manutenção de uma mesma caixa de ferramentas para múltiplas linguagens. Um resultado importante é que as mesmas funcionalidades apresentadas aqui, para a caixa de ferramentas IA636, estão disponíveis também para a linguagem MATLAB. Python e MATLAB seguem, via de regra, uma certa sistemática de intercambeamento das sintaxes tornando esta manutenção perfeitamente plausível.

Toda a caixa de ferramentas, inclusive código-fonte, pode ser obtida a partir da página do projeto do curso de processamento de imagens em Python [ia636]. Neste *site* também podem ser conferidos: o programa do curso, os pontos levantados em uma lista de discussão, referências bibliográficas, manual de instalação do ambiente de programação, e todas as atividades recentes entregues pelos alunos.

5. Conclusões

Python é uma linguagem de programação genérica podendo ser usada em vários domínios de aplicação. Existem diversos módulos que vêm junto à sua distribuição (de funções matemáticas, de manipulação de *strings* e tempo, de utilização de sistemas operacionais e de arquivos, de compactação, criptografia, de suporte SGML, CGI, HTTP, FTP, Telnet, apenas para citar alguns) e inúmeros outros que podem ser incorporados. Como exemplo, podemos destacar o poderoso *PyOpenGL* (interface encapsulada da implementação *OpenGL*). Este módulo vem sendo utilizado com sucesso por grande parte dos alunos do curso de pós-graduação de Computação Gráfica da FEEC-Unicamp. O pacote Numerical, em especial, se adapta perfeitamente ao nosso propósito de manipulação matricial de grandes conjuntos de dados, de forma eficiente, para o processamento de imagens.

O sistema Adesso separa nitidamente conteúdo de apresentação simplificando consideravelmente o esforço de manutenção e gerenciamento da informação de uma caixa de ferramentas, um processo bastante trabalhoso devido ao elevado número de componentes (que tende a aumentar). A adoção da metodologia de estruturar a informação e usar geração de código e documentos de forma automatizada traz inúmeros benefícios sendo os principais: diminuição do número de linhas de informação mestre, responsável tanto pelo programa como

sua documentação gerada, aumentando imunidade a erros (tipicamente introduzidos em situações de "copiar e colar"), diminuindo espaço de armazenamento de cópias de segurança; consistência na interface com o usuário, visto que o código resultante é gerado automaticamente, sendo inerentemente sistemático por todo o sistema; informação facilmente adaptável para gerar código e documentação de acordo com os avanços tecnológicos exigidos pelo mercado; maior imunidade contra erros na criação de um empacotamento e distribuição do aplicativo, permitindo uma consistência na configuração desejada.

A caixa de ferramentas desenvolvida é capaz de suprir boa parte das necessidades básicas quando na implementação de um exercício de processamento de imagens. A grande quantidade de exemplos da documentação gerada auxilia fortemente o aprendizado da linguagem Python em si e, principalmente, dos algoritmos tradicionais de tratamento de imagens. A sintaxe, tanto do Numerical Python quanto do MATLAB, são implementações diretas das fórmulas estudadas nos livros. O corretor automático revelou-se uma ferramenta eficaz no auxílio à avaliação de exercícios de programação, além de permitir a visualização e comparação das soluções pelos códigos entregues e resultados gerados.

A folha de estilo para geração automática de interface entre código C/C++ e Python está em fase de testes. Já é possível aproveitar, no ambiente Python, funções implementadas na linguagem C, sem custo adicional de programação. A caixa de ferramentas de Morfologia Matemática [Barre98], por exemplo, utilizada inicialmente nas plataformas Khoros e MATLAB, já funciona também em Python. O SWIG[6] é uma outra opção de encapsulamento automático, para Python, neste sentido.

Como trabalho futuro, poderíamos citar a criação de folhas de estilo, dentro do sistema Adesso, para facilitar a geração de páginas dinâmicas na internet. Poder-se-ia ter todo um ambiente *on-line* de testes de funções, onde, a priori, é desnecessária a instalação da caixa de ferramentas (o interpretador seria transparente do lado do servidor). A

Figura 17 exibe um protótipo neste sentido construído manualmente para o cálculo do filtro de média, onde o aluno pode escolher a imagem de entrada e a dimensão da máscara. A saída é então visualizada ao lado da imagem original. Todo este processo depende simplesmente de um navegador de internet.

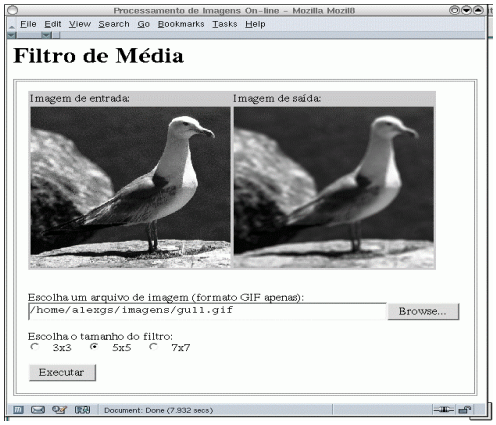


Figura 17 - Cálculo *on-line* do filtro de média usando CGI/Python e a caixa de ferramentas IA636 implementada.

6. Agradecimentos

Agradecemos aos alunos que cursaram IA636 no primeiro semestre de 2002 por toda contribuição dada à implementação da caixa de ferramentas e à FAPESP pelo suporte dado ao Alexandre G. Silva (processo número 00/13671-0) para a realização deste trabalho.

7. Referências

[Jord96] R. Jordan, R. A. Lotufo, *Interactive Digital Image Processing Course on the World-Wide Web*, Proccedings of the 1996 International Conference on Image Processing. IEEE Signal Processing Society. pp. 433-436. September 1996, Lausanne, Switzerland.

[Lotu96] R. A. Lotufo, R. Jordan, *Hands-On Digital Image Processing*, IEEE Frontiers in Education – 26th Annual Conference, FIE-96 pp. 1199-1202, Vol. 3, November 1996. Salt Lake City, Utah, USA.

[Barre98] J. Barrera, G. J. F. Banon, R. A. Lotufo, and R. Hirata Jr., *Mmach: a Mathematical Morphology Toolbox for the Khoros System*, Journal of Electronic Imaging, Vol. 7, No. 1, pp. 174-210, January 1998.

[Alme96] A. C. R. Almeida, A. A. S. Sol, A. A. Araújo, *PhotoPixJ: uma Plataforma para Processamento Digital de Imagens em Java*, SIBGRAPI, October 1998, Rio de Janeiro, Brazil.

[6] http://www.swig.org

[Ross02] G. Rossum, *Python Tutorial*, F. L. Drake Jr., editor, April 10, 2002. http://www.python.org/doc/current/tut/tut.html

[Lutz98] M. Lutz, D. Ascher, *Learning Python*, O'Reilly & Associates, April 1998.

[Asch01] D. Ascher, P. F. Dubois, K. Hinsen, J. Hugunin, and T. Oliphant, *Numerical Python*, LLNLL, University of California, September 7, 2001. http://www.pfdubois.com/numpy/html2/numpy.html

[Mach02] R. C. Machado, *Adesso: Ambiente para Desenvolvimento de Software Científico*, Dissertação de Mestrado, Faculdade de Engenharia Elétrica e de Computação – Universidade Estadual de Campinas. June 2002.

[Mach00] R. C. Machado, R. A. Lotufo, *Adesso: Ambiente Computacional para Desenvolvimento Rápido de Aplicaçõe*s. DCA-FEEC-Unicamp/FCTI Activities Report, May, 2000, Campinas.

[Eddi94] S. L. Eddins, M. T. Orchard, *Using MATLAB and C in an Image Processing Lab Course*, Proceedings of ICIP-94. pp. 515-519, Vol. 1. November 1994, Austin, USA.

[Prec00] L. Prechelt, *An Empirical Comparison of Seven Programming Languages*, Computing Pratices IEEE. pp. 23-29. October 2000.

[Robi00] J. A. Robinson, *A Software System for Laboratory Experiments in Image Processing*, IEEE Transactions on Education. pp. 455-459, Vol. 43, No. 4. November 2000.

[Welc99] B. B. Welch, *Pratical Programming in Tcl and Tk*, 3rd ed., Prentice Hall PTR, 1999.

[Joch99] L. Jochen, tDOM - A fast XML/DOM/XPath package for Tcl written in C. http://www.tuharburg.de/skf/tcltk/papers2000/tDOM3.pdf

[w3c00] W3C Recommendation 6 October 2000, Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/2000/REC-xml-20001006

[Silv01] A. G. Silva, R. A. Lotufo, R C. Machado, *Toolbox of Image Processing for Numerical Python*, Proceedings of XIV SIBGRAPI. p. 402. October 2001, Florianópolis, Brazil.

[ia636] Course of Image Processing in Python. http://www.dca.fee.unicamp.br/ alexgs/curso