



DIOGO RAPHAEL CRAVO  
JOAO LUIZ GRAVE GROSS

## Crop and Drop

Trabalho Final da disciplina de  
Fundamentos de Processamento de  
Imagens

[Prof. Dr. Manuel Menezes de Oliveira Neto](#)

Porto Alegre, 19 de dezembro de 2011.

# SUMÁRIO

- [1. Propostas Inicial e Final](#)
- [2. Material de Apoio](#)
- [3. Objetivo](#)
- [4. Algoritmo](#)
  - [4.1 Algoritmo - Exemplo](#)
- [5. Problemas de Implementação](#)
- [6. Estrutura do Projeto](#)
- [7. Interface e Uso](#)
- [8. Resultados](#)
- [9. Pontos Positivos](#)
- [10. Pontos Negativos](#)
- [11. Conclusão](#)

## 1. Propostas Inicial e Final

Inicialmente o grupo optou pela implementação da ideia apresentada pelo artigo [Drag and Drop Pasting](#) por Jia, et al. de 2009. Porém a proposta deste artigo era implementar uma otimização à borda do contorno da área de seleção, tornando a seleção mais eficiente.

A ideia era interessante, contudo bastante complexa e optamos por fazer a implementação do artigo [Poisson Image Editing](#) de Pérez, et al. de 2003. Este artigo consiste na seleção de um trecho de uma imagem de origem, colando-a sobre outra e unindo as duas imagens através da resolução de um sistema do tipo  $Ax = b$ .

## 2. Material de Apoio

De modo a auxiliar a compreensão do algoritmo apresentado no artigo do Pérez, utilizamos alguns materiais de apoio:

- Artigo Poisson Image Editing - Pérez, et al. – 2003
- [Projetos sobre Poisson Image Editing](#) do Instituto de Ciência da Computação da Brown University - USA
  - materiais utilizados para compreender o algoritmo de Pérez
- Algoritmo/Implementação de um [Linear System Solver](#)

## 3. Objetivo

Este trabalho tem por objetivo implementar a composição de duas imagens de modo que a imagem final apresente um resultado bom o suficiente para que as alterações na imagem sejam minimamente perceptíveis por um usuário destreinado. O resultado esperado é o

resultado apresentado a seguir:

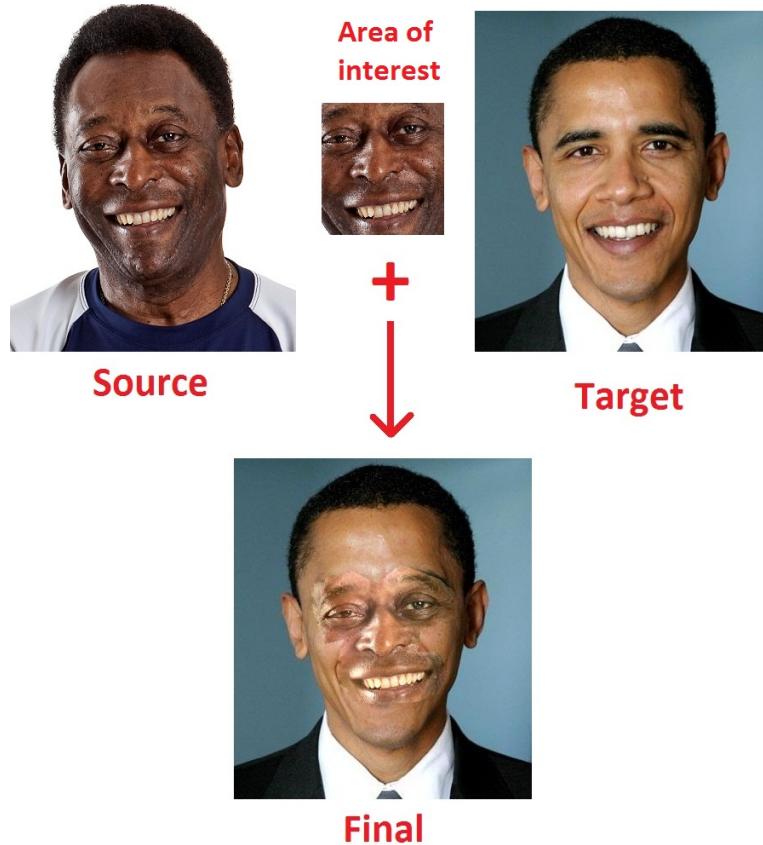


Figura 3.1. Exemplo do resultado esperado para o programa.

## 4. Algoritmo

Com base no artigo de Pérez e nas leituras realizadas, construímos o seguinte algoritmo:

- Selecionar área de interesse
- Colar área de interesse sobre a imagem de destino
- Iniciar composição resolvendo sistema de equações lineares do tipo  $Ax = b$ , onde:

Matriz A

- matriz esparsa (com vários coeficientes zero)
- para pixels  $p$  fora da máscara, as linhas em  $b$  que correspondem aquelas linhas em  $A$  terão a diagonal igual a 1, para indicar que o valor desse pixel já está definido
- para pixels  $p$  dentro na máscara, a diagonal da linha em  $A$  correspondente a linha em  $b$  terá valor 4 e para os pixels  $q$  vizinhos a  $p$  é atribuído -1
- se  $p$  está na posição  $(x,y)$ , seus vizinhos serão os pixels  $q$   $(x-1,y)$ ,  $(x+1,y)$ ,  $(x,y-1)$

### 1) e $(x,y+1)$

Vetor x

- valores finais dos pixels, ou seja, a imagem final
- quantidade de linhas igual a quantidade de pixels da imagem final
- pixels fora da zona de interesse são conhecidos (estão prontos)
- pixels dentro da zona de interesse são desconhecidos (é necessário resolver equações de poisson)

Vetor b

- quantidade de linhas igual a quantidade de pixels da imagem final
- para os pixels  $p_i$  conhecidos as correspondentes linhas  $b_i$  tem o valor copiado da imagem de destino
- para os pixels  $p_i$  desconhecidos (dentro da área de interesse - imagem recortada) as correspondentes linha  $b_i$  possui a soma dos gradientes dos  $n$  vizinhos de  $p$  sendo  $n$  igual a 2, 3 ou 4

## 4.1 Algoritmo - Exemplo

Antes de começar o exemplo, faz-se necessária algumas convenções:

Image coordinates model

$(0,0) | (0,1) | (0,2)$

$(1,0) | (1,1) | (1,2)$

$(2,0) | (2,1) | (2,2)$

RGB

	red	green	blue
0x	xx	xx	xx

Gradient equation

$$4 * p(x,y) - p(x-1,y) - p(x+1,y) - p(x,y-1) - p(x,y+1)$$

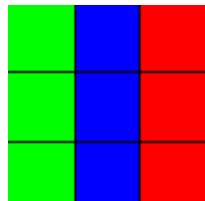
Para a execução do algoritmo precisamos das imagens de origem e destino e também de seus respectivos gradientes.

Source

0x00FF00 | 0x0000FF | 0xFF0000

0x00FF00 | 0x0000FF | 0xFF0000

0x00FF00 | 0x0000FF | 0xFF0000



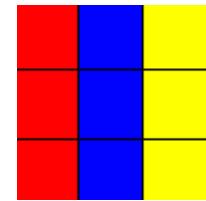
Source Gradient

0x00FE01 | 0x0002FD | 0xFEFF01

=>

0x00FE01 | 0x0002FD | 0xFEFF01

0x00FE01 | 0x0002FD | 0xFEFF01

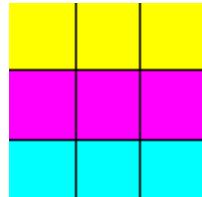


Target

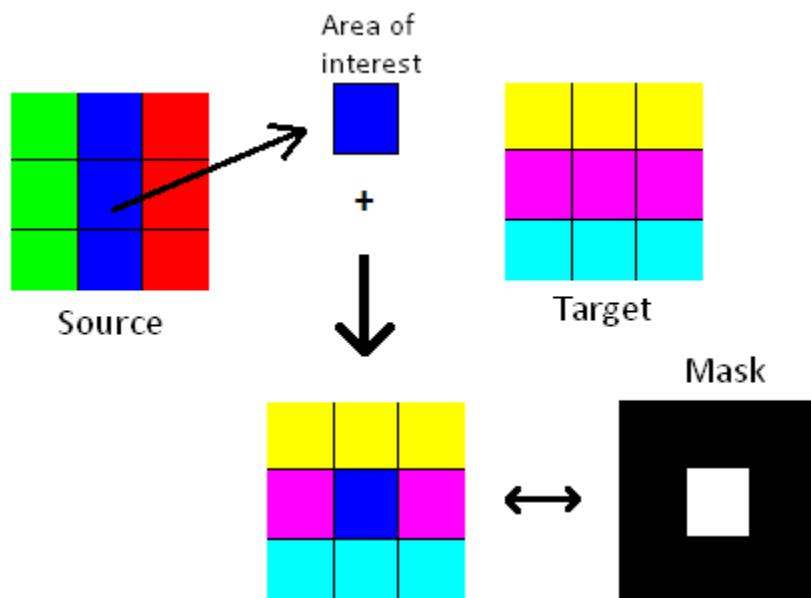
0xFFFF00 | 0xFFFF00 | 0xFFFF00

0xFF00FF | 0xFF00FF | 0xFF00FF

0x00FFFF | 0x00FFFF | 0x00FFFF



Também é necessário selecionar um trecho da imagem de origem e posicioná-la sobre a imagem de destino.



Preparando o sistema linear de forma  $Ax = b$ :

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -1 & 4 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$x = [x_{00} \ x_{01} \ x_{02} \ x_{10} \ x_{11} \ x_{12} \ x_{20} \ x_{21} \ x_{22}]^T$$

```

b = [
    0xFFFF00 Final pixel
    0xFFFF00
    0xFFFF00
    0xFF00FF
    4 * 0x0002FD - 0xFEFF01 - 0x00FE01 - 2 * 0x0002FD
    0xFF00FF Source Gradient
    0x00FFFF
    0x00FFFF
    0x00FFFF
]

```

Resolvendo as equações:

```

//Equação completa
0*x00 - x01 + 0*x02 - x10 + 4*x11 - x12 + 0*x20 - x21
+ 0*x22 = 4 * 0x0002FD - 0xFEFF01 - 0x00FE01 - 2 * 0x0002FD

```

```

//Equação simplificada
- x01 - x10 + 4*x11 - x12 - x21 = 0x0008F7

```

```

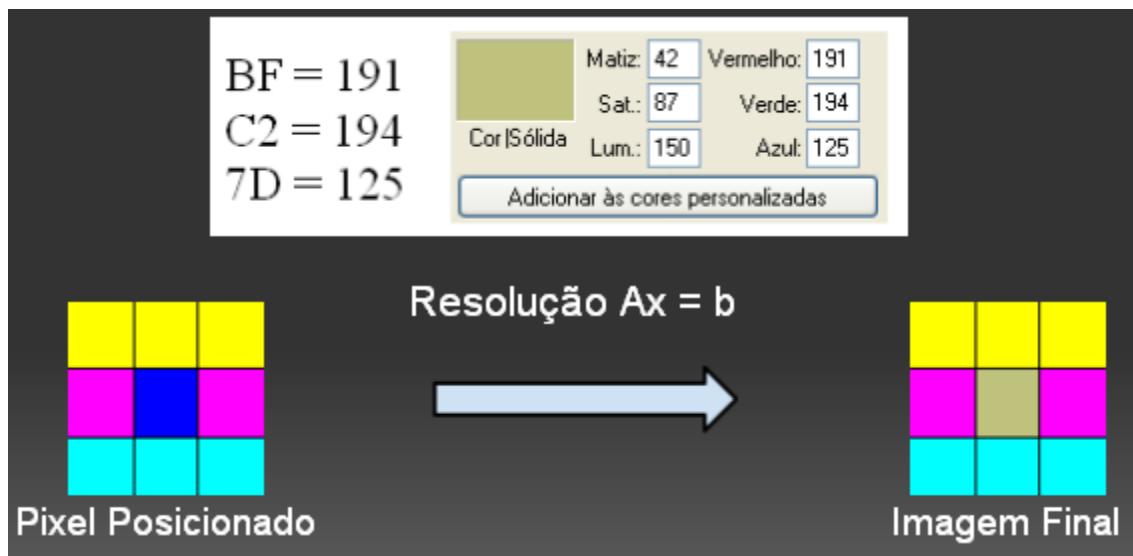
//Substituição de variáveis conhecidas
- 0xFFFF00 - 0xFF00FF + 4x11 - 0xFF00FF - 0x00FFFF = 0x0008F7

```

```

//Obtendo o pixels desconhecido
x11 = (0x0008F7 + 0xFFFF00 + 0xFF00FF + 0xFF00FF + 0x00FFFF) / 4
      = 0xBFC27D

```



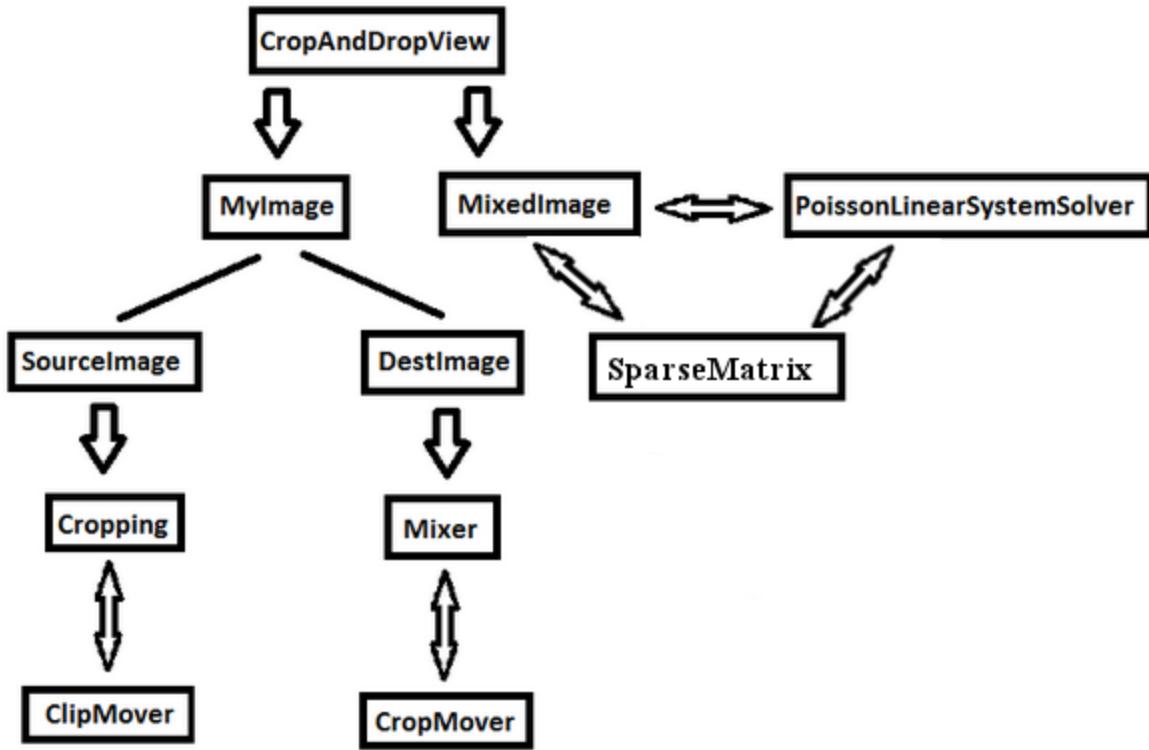
## 5. Problemas de Implementação

Durante a implementação, encontramos muitos problemas como:

- Implementar seleção da imagem de origem com área de seleção móvel (com o mouse)
- Matriz esparsa é muito grande
- Implementar solver de sistema linear de equações para uma matriz **GIGANTE**, como é a nossa
- Procurar uma forma otimizada de representar uma matriz esparsa (matriz cheia de coeficientes zero)

Tentou-se fazer com que o programa inteiro ficasse em Java, o que não foi possível devido aos problemas com matrizes esparsas e equação de Poisson. Decidiu-se manter as classes que estavam destinadas a resolver estes problemas, mas criou-se novo código em MATLAB que o faz.

## 6. Estrutura do Projeto



**Figura 6.1. Estrutura de classes do programa em Java.**

A classe `CropAndDropView` é a main da aplicação. Nela são definidos os elementos da interface e todas as funções de callback dos botões e check boxes. A classe `MyImage` é a classe mãe das classes `SourceImage` e `DestImage`. Elas implementam manipulações sobre as imagens de origem (`sourceImage`) e de destino (`DestImage`), já a classe `MyImage` contém métodos usados por estas duas outras classes.

As classes `Cropping` e `ClipMover` são responsáveis respectivamente por capturar a região selecionado da imagem de origem (o recorte, crop) e por posicionar o clip de recorte sobre a imagem de origem. Ou seja, primeiro o clip de recorte é posicionado e a imagem contida dentro deste clip de recorte constituirá o recorte da classe `Cropping`.

As classes `Mixer` e `CropMover` funcionam de forma semelhante às classes `Cropping` e `ClipMover`. `CropMover` é usado para mover o recorte feito pela classe `Cropping` sobre a imagem de destino. A classe `Mixer` é responsável por unir as duas imagens (imagem de destino e recorte).

A classe `MixedImage` tem o objetivo de misturar duas imagens passadas como parâmetro ao construtor (sendo que a imagem de destino precisa já conter o crop da imagem fonte). Ela faz uso das classes `SparseMatrix` e `PoissonLinearEquationSolver` para achar a solução do sistema linear que leva à imagem final, conforme descrito no algoritmo. Infelizmente, estas classes levaram a muitos problemas e tiveram que ser deixadas de lado.

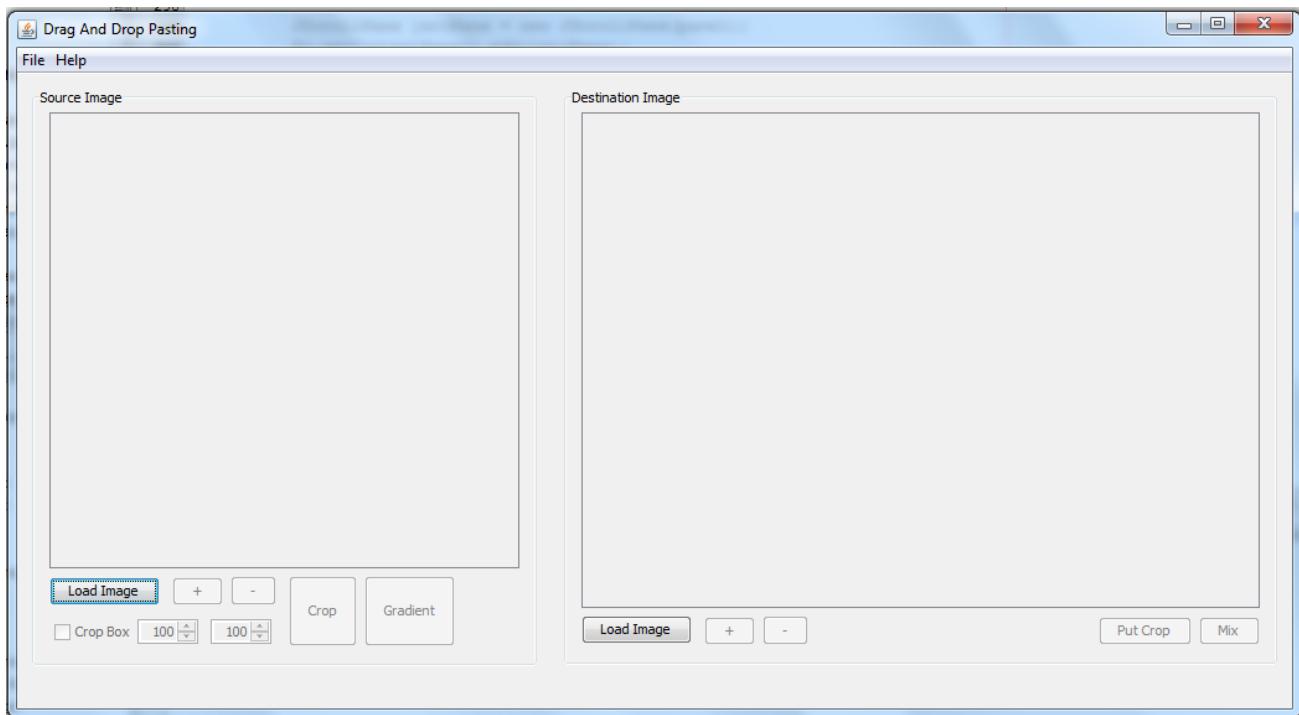
## 7. Interface e Uso

Para usar o programa, é necessário começar pela parte implementada em Java. Usa-se

a linha de comando para iniciar o programa, chamando-a com o comando a seguir à partir do diretório em que encontra-se o arquivo cropAndDrop.jar do projeto:

```
$ java -jar cropAndDrop.jar
```

É necessário uma JRE de versão propriamente atualizada para tal, bem como os arquivos ‘appframework-1.0.3.jar’ e ‘swing-worker-1.1.jar’ dentro do diretório lib, sendo que lib está no mesmo diretório do cropAndDrop.jar. Ao iniciar o programa, o usuário verá a seguinte tela:

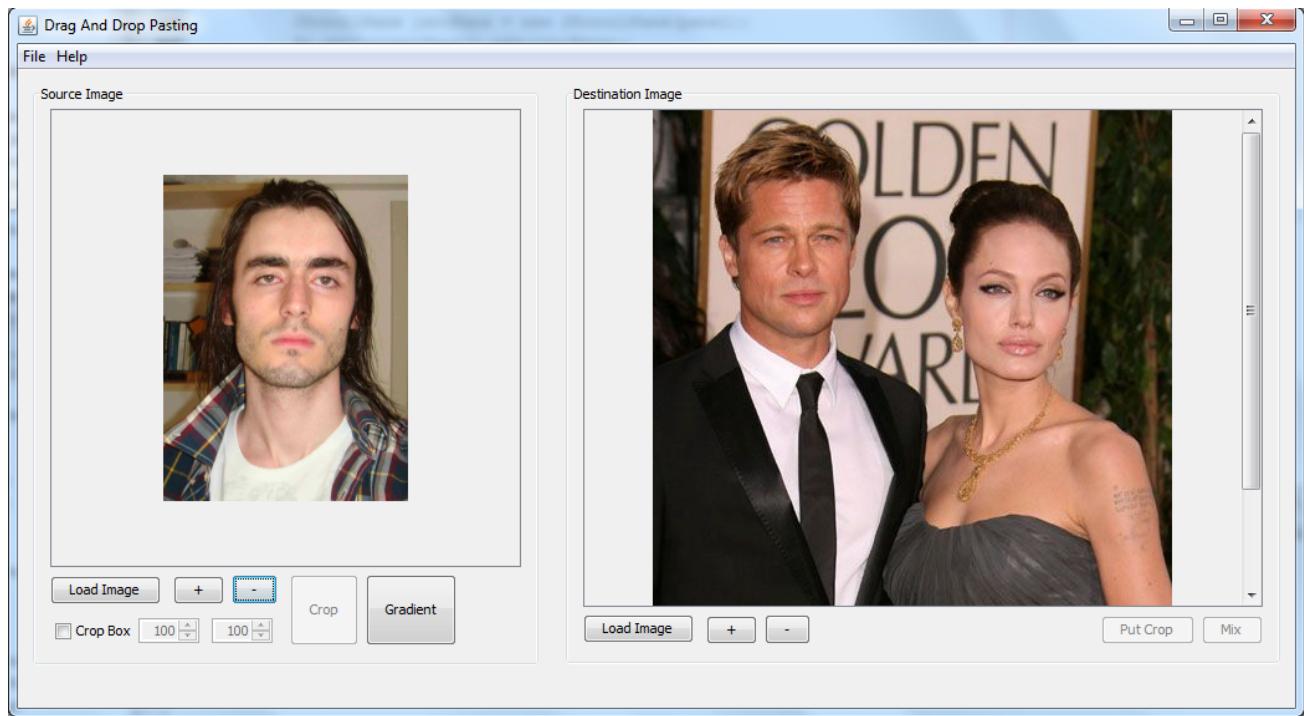


**Figura 7.1. Tela de início do programa.**

Ela oferece a opção de carregar uma imagem fonte e uma imagem destino. A imagem fonte é aquela que será recortada para ser colada na imagem destino.

A barra de menu (na parte superior) oferece as opções File e Help. Em File, há a opção de terminar o programa. Help oferece a opção About, que traz uma janela informando dados sobre o programa. Os botões desabilitados possuem funções que só podem ser usadas após carregamento das imagens sobre as quais operam.

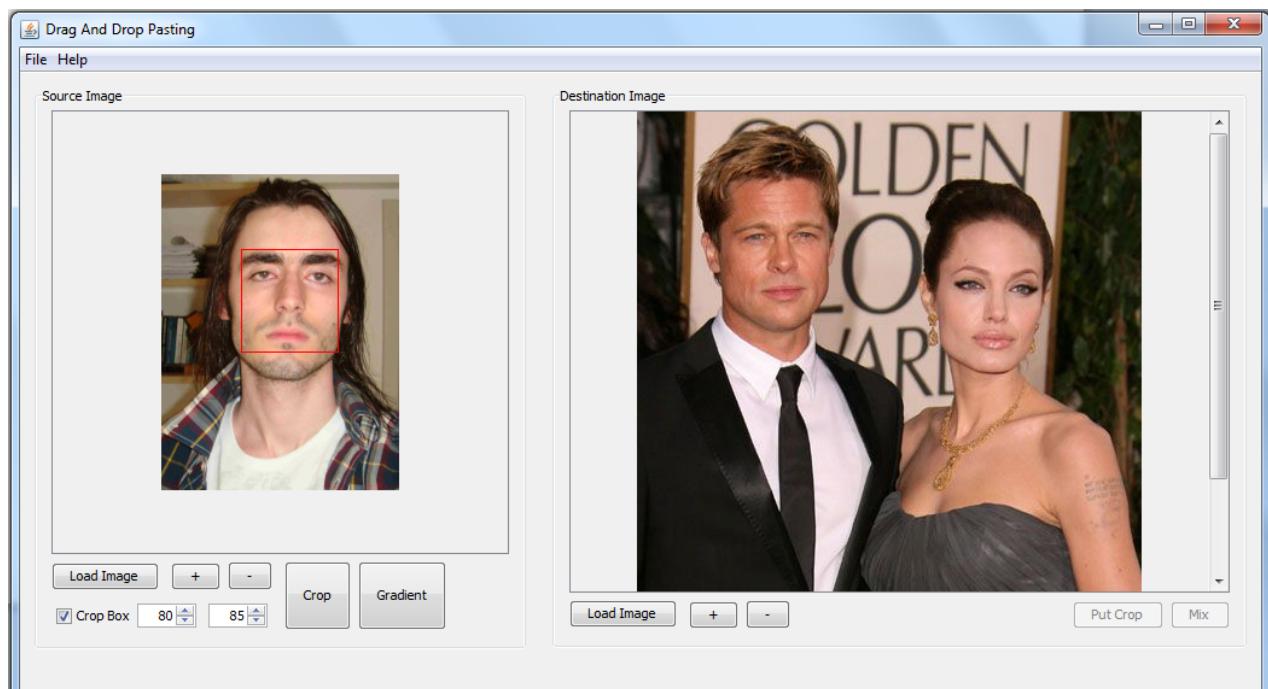
A figura à seguir exemplifica o carregamento de imagens fonte e destino. Observa-se que os botões da interface tornam-se clicáveis.



**Figura 7.2. Exemplo de imagens carregadas na interface.**

Imagens carregadas são exibidas em painéis com barras de scroll para fácil visualização, sendo mantido seu tamanho original.

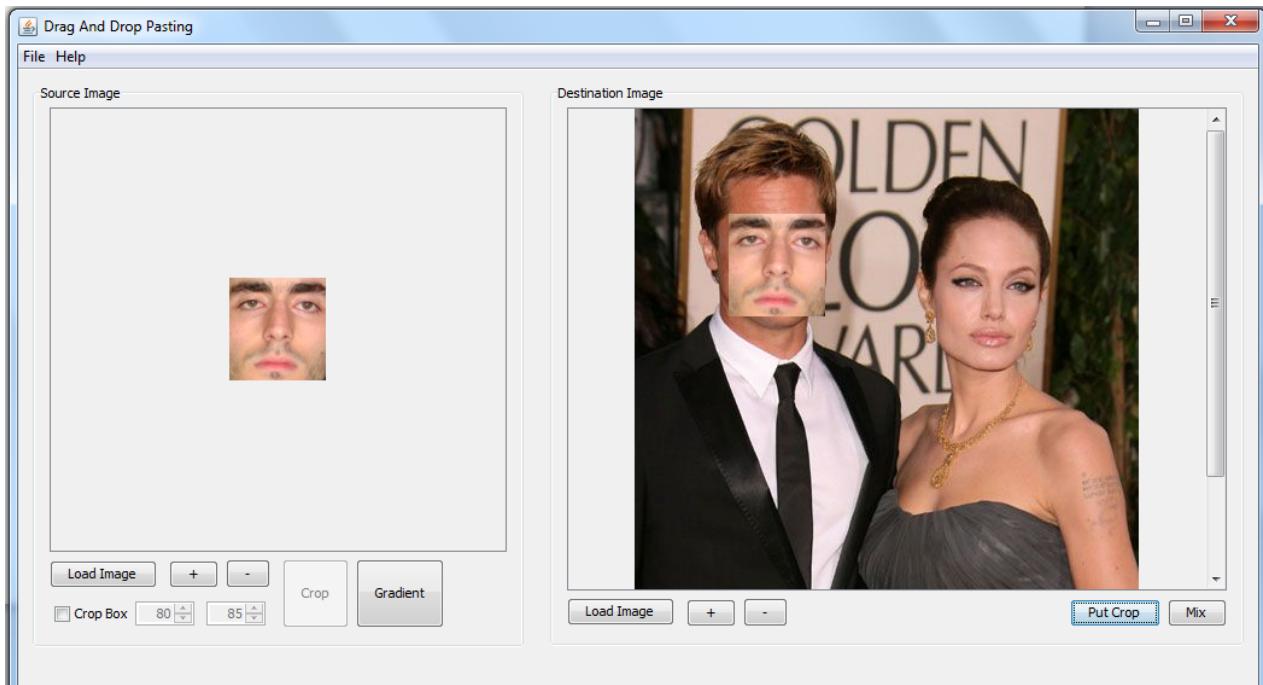
A interface oferece opções de *zoom in* (+) e *zoom out* (-) para ambas imagens. A primeira imagem pode ter seu gradiente calculado e exibido (pelo botão Gradient) e oferece a opção de mostrar a caixa de recorte (Crop Box) com uma checkbox. Ela é usada na imagem à seguir.



**Figura 7.3. Uso da Crop Box.**

A Crop Box é uma caixa retangular vermelha, que delimita o espaço de corte da imagem fonte. Ela pode ser livremente habilitada e desabilitada por sua checkbox. Seu tamanho é especificado nos spinners que antecedem o botão Crop.

Ao clicar em Crop, a imagem fonte será recortada segundo a região delimitada pela Crop Box.

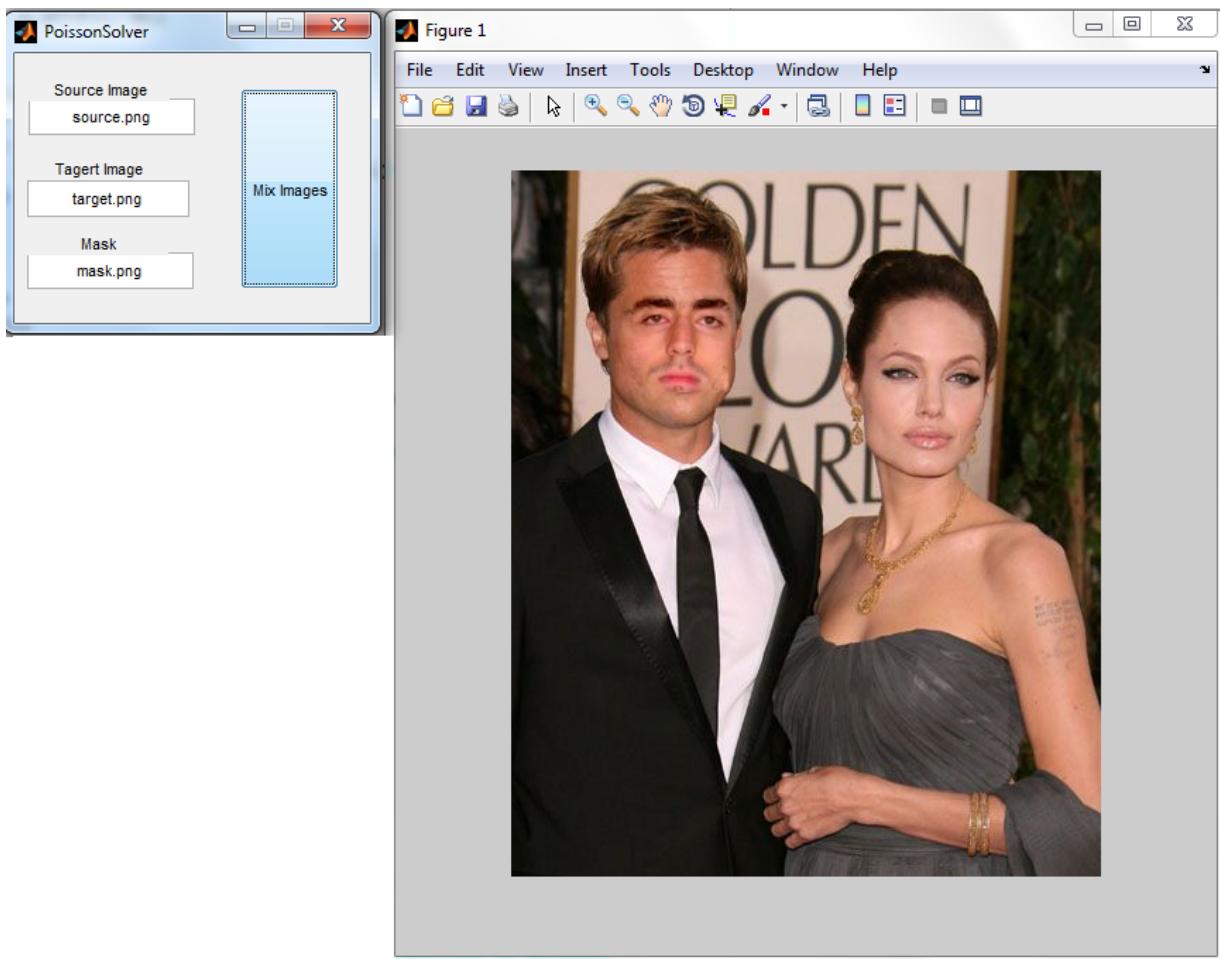


**Figura 7.4. Recorte (Crop) de imagem e colagem na imagem destino (Put Crop).**

Uma vez selecionada a área da imagem fonte que deseja-se misturar à imagem destino, deve-se usar a função Put Crop para colá-la na imagem destino. Ela será inserida no canto superior esquerdo desta e será possível arrastá-la para onde o usuário desejar.

A função Mix, de fato, é aquela que não está implementada em Java. Ela não mistura as imagens, mas salva-as como "source.jpg", "mask.jpg" e "target.jpg" na mesma pasta em que encontrar-se o arquivo cropAndDrop.jar.

Estas imagens devem ser usadas de entrada para o script em MATLAB. Deve-se colocá-las na mesma diretório do script e executá-lo pela interface do MATLAB.

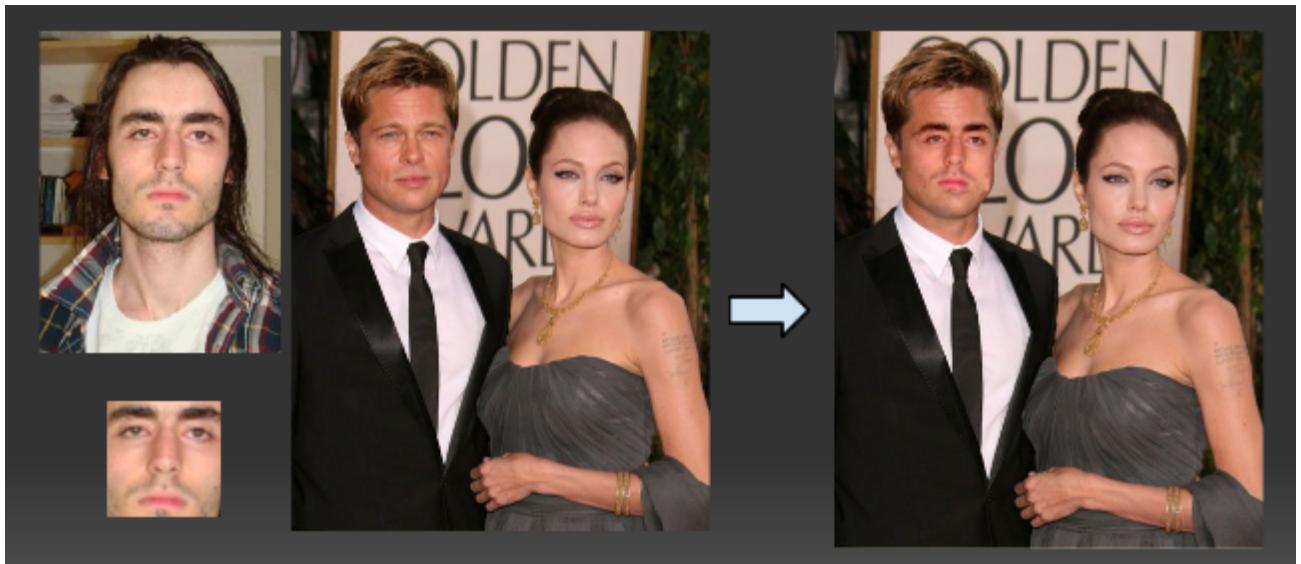
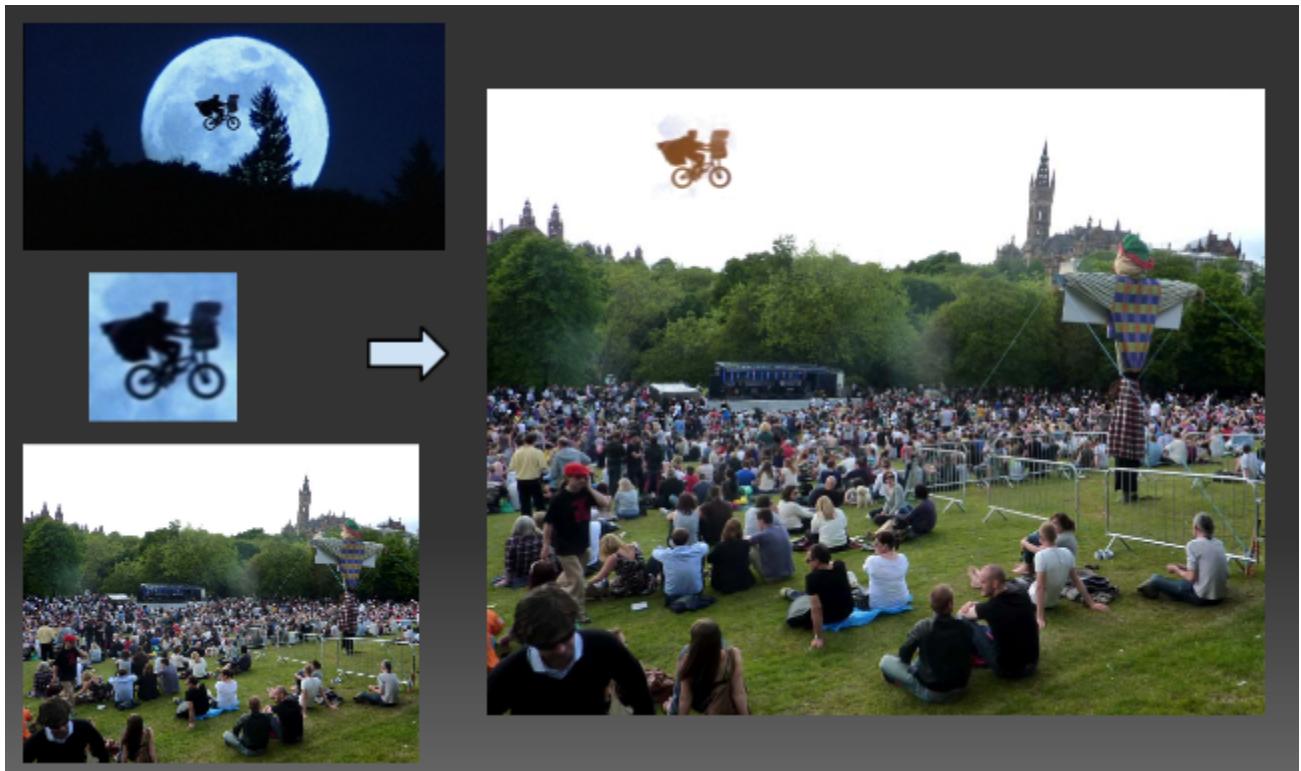


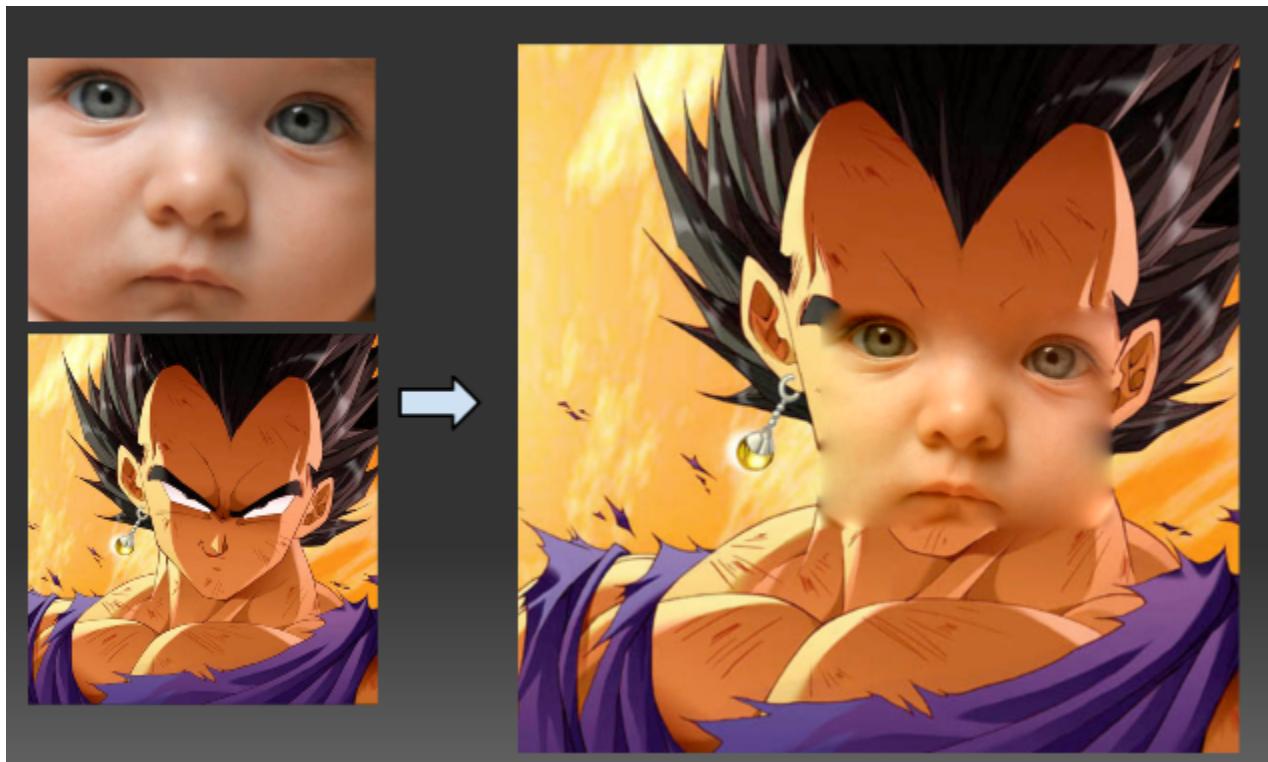
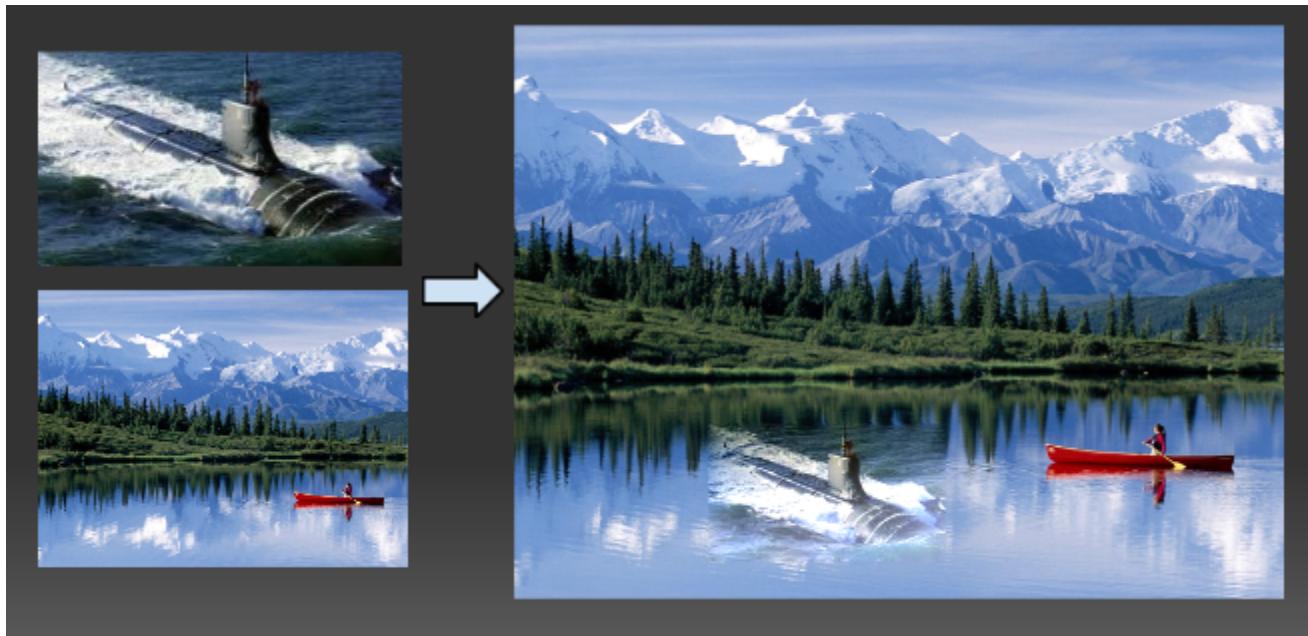
**Figura 7.5. Programa MATLAB e resultado final.**

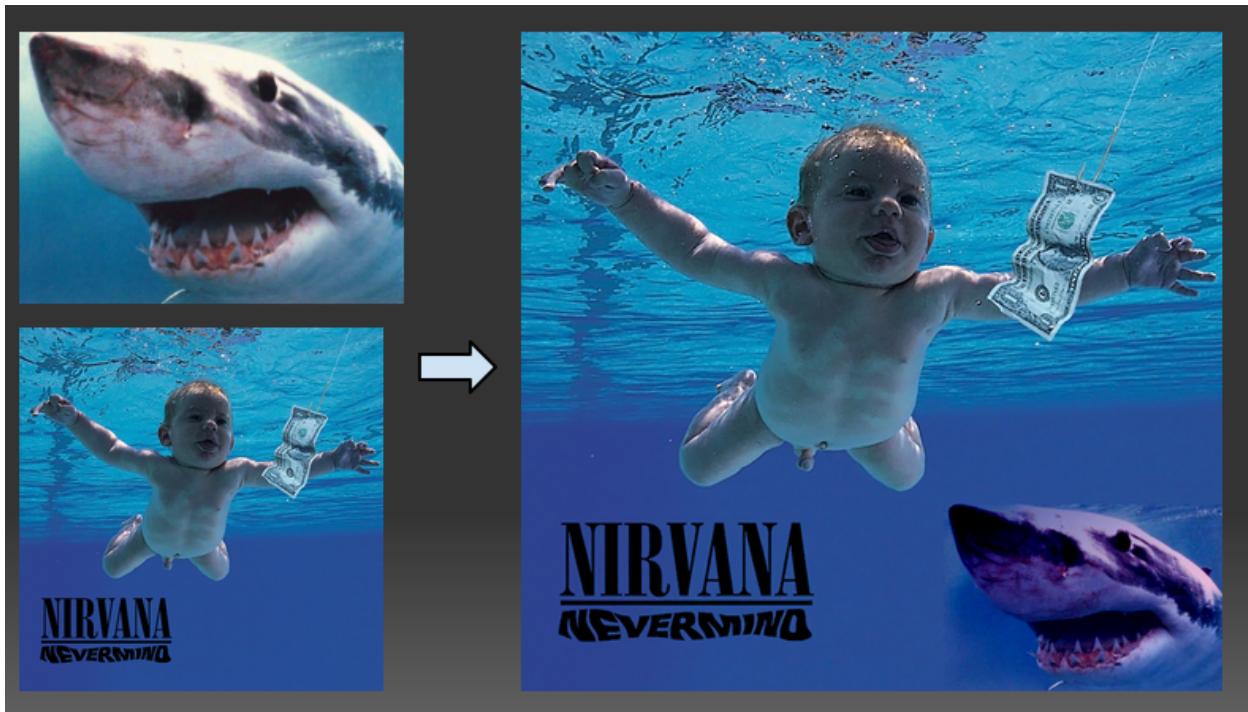
Ao executar o script, a janela PoissonSolver da figura 7.5 será aberta. Deve-se especificar as imagens de entrada e utilizar o botão Mix Images para obter o resultado final em Figure 1, que irá aparecer logo em seguida (o tempo de processamento é curto e o uso de memória, baixo). A imagem final é automaticamente salva no diretório do script sob o nome “finalImage.png”.

## 8. Resultados

Seguem imagens fonte com seus respectivos crop (a própria imagem é usada em caso de omissão) junto com imagens destino e a imagem final.







## 9. Pontos Positivos

- + Melhor compreensão a respeito da representação RGB, gradientes e convolução
- + Tecnologias Java antes não manipuladas
- + Bons resultados
- + Possibilidade de trabalhar novamente com MatLab

## 10. Pontos Negativos

- Muita demora até entender o algoritmo
- Implementação demorou para ser concluída, logo não pôde ser refatorada visando melhorias
- Implementação em Java do core matemático da aplicação não funcionou
- Gostaríamos de ter feito versionamento de código

## 11. Conclusão

Este trabalho teve o objetivo de implementar a teoria contida em um artigo subtido à SIGGRAPH (Poisson Image Editing - Pérez, et al. - 2003). Implementou-se um programa em

MATLAB e outro em Java.

Na fase inicial, fez-se busca por materiais que ajudassem a entender o processo, o qual é bastante complexo. Muitos algoritmos foram estudados nesta fase e alguns artigos lidos.

Já a implementação, esta foi iniciada com o objetivo de ser puramente em Java. Ao longo de seu curso, foram encontrados muitos problemas (como o tamanho gigantesco de um sistema linear que precisava ser resolvido). Portanto, optou-se por usar parte do código pronto em Java para servir de entrada para um novo código implementado em MATLAB. Fez-se isto pois o MATLAB oferece muitas operações já implementadas, as quais não seriam simples em Java.

Os resultados obtidos da forma descrita foram exatamente o que se esperava: a possibilidade de criar imagens compostas que parecem naturais.