Rodrigo Rueda

Ferramenta para Segmentação Interativa de Imagens

Orientadora:
Profa. Dra. Nina S. T. Hirata

Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciências da Computação

São Paulo

2007

Sumário

Lista de Figuras

1 Introdução

O problema de segmentação de imagens consiste, basicamente, em separar uma imagem em regiões cujos elementos possuam características semelhantes. Algoritmos que segmentam imagens são utilizados em aplicações de diversas áreas, tais como análise de imagens médicas e edição de imagens. Embora existam técnicas de segmentação automáticas, elas são geralmente específicas para um tipo de aplicação, de modo que não temos garantia de bons resultados no caso geral (em que, a priori, não temos informações sobre a imagem a ser segmentada).

Tendo em vista essas limitações, surgiram algoritmos semi-automáticos para segmentação de imagens. A idéia é, com o mínimo possível de interação humana, conseguir obter segmentações relativamente arbitrárias. Existem, também, diferentes abordagens para a segmentação interativa e, dentre alguns artigos que descrevem técnicas para a solução do problema, foram escolhidos dois ([5],[4]) para estudo neste trabalho.

Primeiramente serão introduzidos alguns conceitos básicos necessários à compreensão dos algoritmos escolhidos e, em seguida, os algoritmos serão apresentados junto de alguns resultados obtidos a partir da ferramenta construída. Adiante, a estrutura da ferramenta, seu objetivo e sua forma de utilização serão expostos.

2 Conceitos

Aqui serão descritos os principais conceitos necessários à compreensão do processo de segmentação de imagens e dos algoritmos implementados.

2.1 Imagens Digitais

Uma imagem digital pode ser descrita como uma matriz bidimensional $I_d[x, y]$ cujos elementos são chamados pixels e a eles são associados valores que correspondem à intensidade da imagem naquela coordenada.

Em uma imagem digital monocromática, um pixel é representado por um escalar inteiro entre 0 e L. Neste trabalho, usaremos $L = 255$, o que corresponde a dizer que teremos 256 níveis de cinza. Para imagens coloridas, utilizaremos o espaço de cores RGB, em que cada pixel é representado por um vetor com três componentes (as cores primárias: vermelho, verde e azul), cada um no intervalo de 0 a 255.

2.2 Grafos

Seja V um conjunto qualquer. Denota-se por $V^{(2)}$ o conjunto de todos os pares não ordenados dos elementos de V. Cada elemento de $V^{(2)}$ é da forma $\{v, w\}$ com $v \neq w$. Um grafo é um par (V, A) em que A é um subconjunto de $V^{(2)}$. Os elementos de V são chamados vértices e os de A são chamados arestas.

Muitos algoritmos de segmentação fazem uso de grafos e, para isso, um grafo associado à imagem digital é definido da seguinte forma: cada vértice corresponde a um pixel e as arestas conectam pixels vizinhos de acordo com alguma definição de vizinhança. Seja $p \in \mathbb{Z}^2$ o vetor das coordenadas de um pixel da imagem, as funções de vizinhança $(N(p))$mais comuns são:

- Vizinhança de von Neumann
  $N(p) = \{q \in \mathbb{Z}^2 : \|p - q\|_1 := \sum_{i=1}^{n} |p_i - q_i| = 1\}$
- Vizinhança de Moore
  $N(p) = \{q \in \mathbb{Z}^2 : \|p - q\|_\infty := \max_{i=\overline{1,n}} |p_i - q_i| = 1\}$

2.3 Autômatos Celulares

Um autômato celular é um sistema dinâmico, discreto no tempo e no espaço, e que opera em um conjunto de células de uma estrutura uniforme e regular. Cada célula possui um estado que, conforme o autômato evolui no tempo, é modificado de acordo com uma função de transição baseada nos estados das células vizinhas.

Mais formalmente, definiremos um autômato celular A como uma quadrupla $A = (P, S, N, \delta)$ onde S é um conjunto de estados, N uma função de vizinhança, δ uma função de transição e $P \subsetneq \mathbb{Z}^n$ um conjunto que define as células do autômato.

Por fim, diremos que um autômato é associado a uma imagem digital I quando $P \subsetneq \mathbb{Z}^2$ for o conjunto dos pixels de I.

2.4 Segmentação

Aplicações de diversas áreas utilizam-se do processo de segmentação de imagens, que consiste em dividir uma imagem em regiões de interesse de acordo com algum contexto. Em editores gráficos, por exemplo, a segmentação pode ser utilizada para possibilitar que partes de uma imagem sejam recortadas e inseridas em outra imagem (composição de imagens).

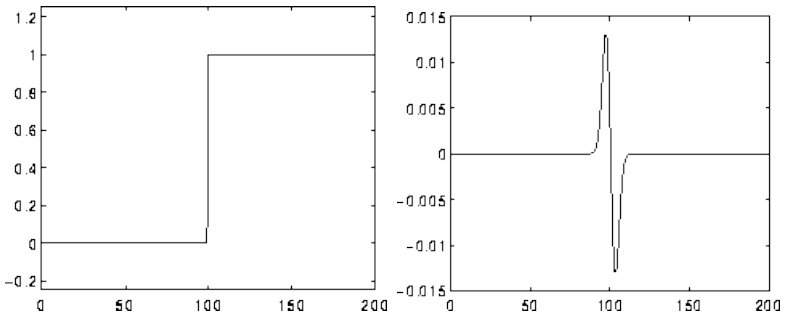
Podemos separar os algoritmos de segmentação em dois grupos: automáticos e interativos. Os primeiros necessitam pouca ou nenhuma interação humana, mas geralmente são específicos no que diz respeito ao tipo de imagem que conseguem segmentar e ao nível de detalhes do resultado. Os interativos superam essas duas limitações fazendo uso de interação humana para controlar e guiar o processo - é esse tipo de algoritmo que é usado na composição de imagens, já que dificilmente algoritmos automáticos conseguiriam captar o contexto da segmentação.

Grande parte dos algoritmos de segmentação baseia-se na localização de regiões onde

há mudanças na intensidade dos pixels da imagem e, para isso, utilizam detectores de borda baseados no gradiente e no laplacian, entre outros.

2.4.1 Zero-Crossing

Zero-crossing é uma técnica de detecção de bordas baseada no Laplacian que, por sua vez, é uma aproximação para a segunda derivada de uma imagem em escalas de cinza. A figura abaixo[1] ilustra o que acontece com o laplacian de uma imagem digital em uma borda.



(a) Borda (b) Laplacian

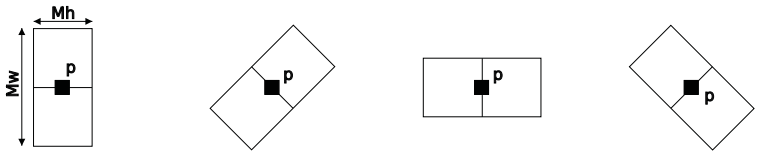
Figura 2.1: Laplacian de uma imagem em escalas de cinza.

O laplacian cruza o zero exatamente na borda. Na prática não é o que acontece e o laplacian dificilmente assume zero como valor. Ainda assim, podemos identificar os zero-crossings através dos picos positivo e negativo do laplacian. Outro problema é a obtenção de zero-crossings não só nas bordas, mas em todas regiões onde há mudança de intensidade dos pixels. Para amenizar esse problema, a imagem orignal é suavizada através de um filtro gaussiano antes do cálculo do laplacian e são selecionados somente os zero-crossings cuja diferença entre os picos positivo e negativo ultrapassam um limiar pré-determinado.

2.4.2 Separabilidade

Métodos de extração de bordas baseados no gradiente ou no laplacian da intensidade dos pixels tendem a ter resultados ruins em imagens com ruídos ou texturas. Para superar esse problema, existem vários métodos baseados na distribuição estatística das características da imagem. Esses métodos são robustos em comparação com os primeiros, mas requerem maior poder de processamento.

[1]Retirada de http://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm



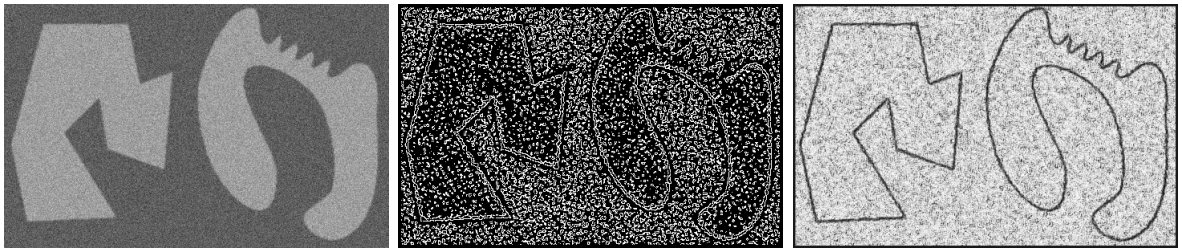
(a) 0º (b) 45º (c) 90º (d) 135º

Figura 2.2: Máscaras usadas no cálculo de separabilidade.

O método da separabilidade, descrito em [1], tenta contornar esses problemas através de um algoritmo eficiente, que estabelece para cada pixel da imagem valores que indicam o grau de diferença entre duas regiões ao redor dele. São utilizadas quatro máscaras que definem essas regiões e os cálculos são feitos da seguinte forma

$$\begin{aligned} \eta &= \frac{{\theta_b}^2}{{\theta_T}^2} \\ {\theta_b}^2 &= n_1(\overline{P_1} - \overline{P})^2 + n_2(\overline{P_2} - \overline{P})^2 \\ {\theta_T}^2 &= \sum_{i=1}^{n_1+n_2} (P_i - \overline{P})^2 \end{aligned}$$

onde n_i é o número de pixels na região i, P_i é a intensidade do pixel i, $\overline{P_i}$ é a média das intensidades dos pixels da região i, $\overline{P}$ é a media da intensidade dos pixels das duas regiões e η mede o grau de separabilidade.



(a) Imagem Original (b) Zero-crossing (c) Separabilidade

Figura 2.3: Zero-Crossing e separabilidade em imagem com ruído

Nota-se a péssima qualidade do zero-crossing, que pode ser melhorado através de uma maior suavização da imagem original e de um valor adequado para o limiar. No entanto, tais mudanças também suavizam as bordas que queremos identificar.

3 Atividades

Aqui são apresentados os dois algoritmos e a ferramenta de segmentação, todos implementados em java, além de alguns resultados obtidos através da ferramenta.

3.1 Growcut

3.1.1 Algoritmo

Seja I uma imagem digital e $A = (P, S, N, \delta)$ o autômato celular associado a essa imagem. O estado inicial do autômato é dado por:

$$\forall p \in P : l_p = 0, \theta_p = 0, \vec{C_p} = RGB_p$$

onde $l_p \in \mathbb{Z}$ é o rótulo, $\theta_p \in [0, 1]$ é a força e $\vec{C_p}$ é a característica da célula. A função de transição δ e a atualização das células são descritas no código a seguir:

para todo $p \in P$ **faça**
  $l_p^{t+1} \leftarrow l_p^t$
  $\theta_p^{t+1} \leftarrow \theta_p^t$
  para todo $q \in N(p)$ **faça**
    se $g(\|\vec{C_p} - \vec{C_q}\|_2) \cdot \theta_q^t > \theta_p^t$ **então**
      $l_p^{t+1} \leftarrow l_q^t$
      $\theta_p^{t+1} \leftarrow g(\|\vec{C_p} - \vec{C_q}\|_2) \cdot \theta_q^t$
    fim se
  fim para
fim para

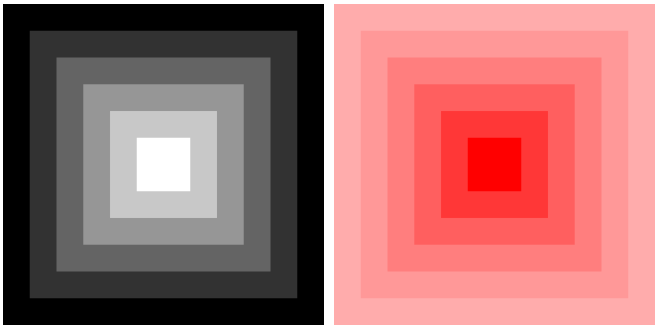
onde a função $g(x)$ é monotônica decrescente, limitada no intervalo $[0, 1]$ e, neste trabalho, assim como no artigo original, é dada por:

$$g(x) = 1 - \frac{x}{max\|\vec{C}\|_2}$$

O usuário inicia o processo de segmentação através da marcação de alguns pixels – denominados sementes – nas áreas correspondentes aos limites dos objetos de interesse. Cada semente tem seu rótulo modificado de acordo com a cor escolhida para rotulação e a força recebe o valor 1. Conforme o autômato evolui, os rótulos propagam-se para outros pixels até que o autômato torne-se estável, ou seja, o estado de cada célula não se altere mais. A garantia de convergência do método decorre do seguinte fato:

$$\forall p \in P : \begin{cases} \theta_p^t \geqslant \theta_p^{t-1} \\ \theta_p \leqslant 1 \end{cases}$$

Na imagem que se segue foi rotulado um único pixel, no centro, com a cor vermelha. A segunda imagem mostra o resultado ao aplicarmos o algoritmo descrito acima – o rótulo foi propagado por toda a imagem, mas perdeu força conforme passava por regiões onde há mudança na intensidade dos pixels.



(a) Imagem Orignal (b) Rótulos

Figura 3.1: GrowCut: propagação dos rótulos e perda de força nas bordas

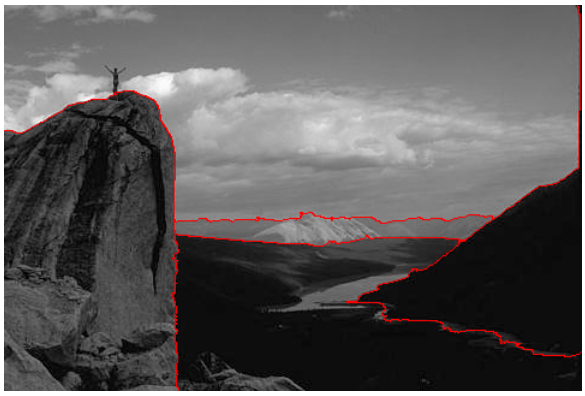
Nos testes feitos, este algoritmo funcionou bem, tanto para imagem em escalas de cinza quanto para imagens coloridas, mas foram notadas algumas deficiências também. A primeira é o tempo necessário para que a segmentação termine – em geral, quanto menos rótulos, maior é o tempo necessário para o autômato convergir – o que leva à necessidade de mais interação. Outro problema é que, para se obter boas segmentações, os rótulos devem ser colocados ao redor de todos objetos de interesse; caso isso não seja feito, serão necessárias novas interações.

Para amenizar tais problemas, optou-se por executar o algoritmo em uma thread separada da interação do usuário. Assim, o usuário pode continuar o processo de rotulação enquanto a segmentação acontece e o algoritmo é capaz de, na maioria dos casos, utilizar os novos rótulos sem perder informações da segmentação obtida anteriormente. Na maioria dos casos, pois quando rótulos são removidos não há como saber para que outras células

aquele rótulo se propagou e, portanto, é necessário iniciar a segmentação desde o início.

3.1.2 Resultados

(a) Imagem Orignal (b) Imagem com Rótulos



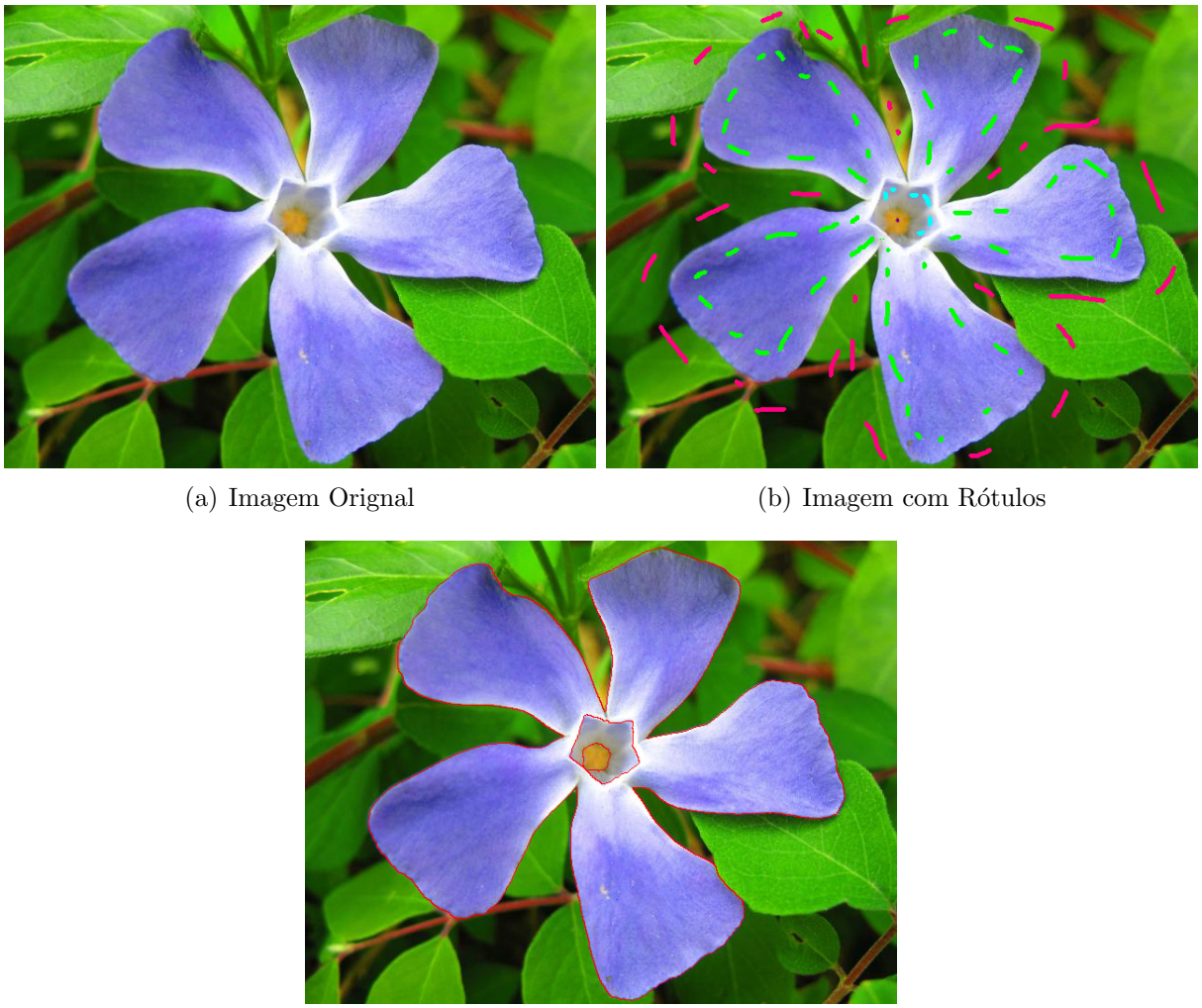
(c) Imagem Segmentada

Figura 3.2: GrowCut: Resultados para a imagem 14037

(a) Imagem Orignal (b) Imagem com Rótulos

(c) Imagem Segmentada

Figura 3.3: GrowCut: Resultados para a imagem 189080



(a) Imagem Orignal

(b) Imagem com Rótulos

(c) Imagem Segmentada

Figura 3.4: GrowCut: Resultados para a imagem “blue flower”

3.2 Sis

3.2.1 Algoritmo

O algoritmo *SIS*, Separability-Based Intelligent Scissors, descrito em [4], é baseado no *IS*, apresentado em [3]. Ambos baseiam-se na busca por caminho de custo mínimo em grafos e têm como forma de interação a escolha de pixels, chamados sementes, que serão os vértices inicial e final da busca.

Sendo p um pixel da imagem e q vizinho de p, de acordo com alguma definição de vizinhança, a função de custo entre os dois pixels será:

$$l(p,q) = w_Z f_Z(q) + w_S f_S(q) + w_{SD} f_{SD}(p,q)$$

onde w_Z, w_S e w_{SD} são pesos especificados pelo usuário e $f_Z(q)$, $f_S(q)$ e $f_{SD}(p,q)$ são o zero-crossing, a separabilidade e a direção da separabilidade, respectivamente.

O zero-crossing $f_Z(q)$ é uma função binária que toma 0 como valor se q faz parte de uma borda e 1 se não faz.

A separabilidade $f_S(q)$ tem custo baixo quando temos um valor alto de separabilidade em q, o que indica que estamos em uma borda e a definimos como

$$f_S(q) = 1 - \eta(q)$$

onde $\eta(q)$ é o maior que se obteve entre as quatro máscaras.

A direção da separabilidade, $f_{SD}(p,q)$, deve ter custo alto quando temos separabilidade alta entre p e q e é definida pela fórmula:

$$f_{SD}(p,q) = \frac{1}{\pi}\{|\theta_p - \phi_{p,q}| + |\phi_{p,q} - \theta_q|\}$$

onde θ_p é a direção de máscara que nos dá maior separabilidade entre as quatro máscaras e $\phi_{p,q}$ é a direção da aresta que liga p a q.

Os pesos usados foram $0,43$, $0,43$ e $0,16$, os mesmo utilizados em [4] e [3], mas seria necessário um estudo melhor da influência desses pesos no resultado da segmentação. Por exemplo: como o zero-crossing é binário e a separabilidade é um valor entre 0 e 1, na prática, o primeiro acaba influenciando muito mais a segmentação, o que pode levar a resultados ruins para imagens com ruídos ou texturas.

As máscaras usadas no cálculo da separabilidade também influenciam muito o resul-

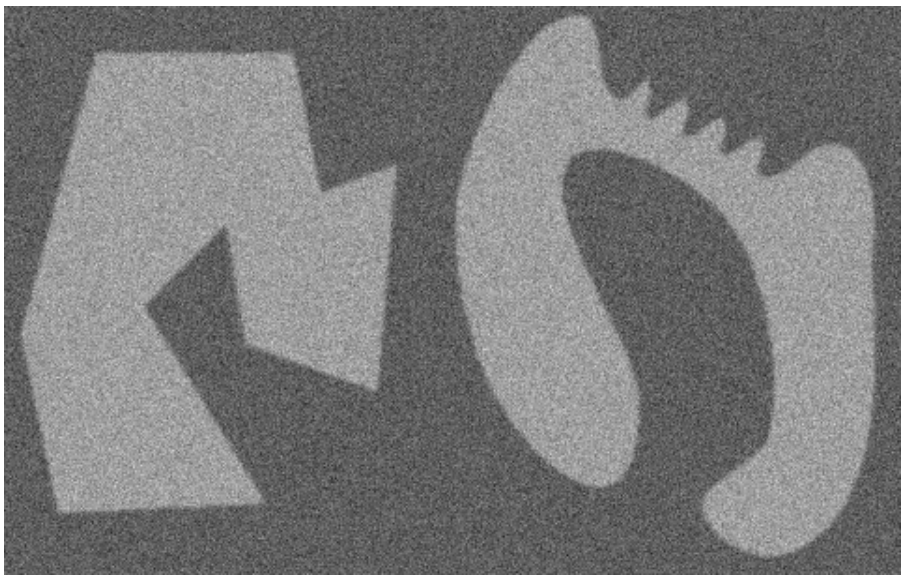
tado final. As suas dimensões é que definem que tipo de texturas ou ruídos conseguirão ser ignorados e, também, que tipo de bordas conseguirão ser identificadas. Com máscaras de dimensões $M_W = 5$, $M_H = 3$ só foi possível captar texturas e ruídos pequenos. Ao aumentá-las, apesar de tratar texturas e ruídos maiores, acabava perdendo a definição das bordas.

Em relação à busca de custo mínimo, o algoritmo implementado baseia-se no conhecido algoritmo de *Dijkstra*. A fila de prioridade é baseada em heap, mas foram realizadas alterações que tornaram o algoritmo mais eficiente, de modo que não foi preciso implementar limitação na região de busca, apresentada em [4]. Essas alterações foram feitas na fila de prioridades: anteriormente, a operação de verificar se um objeto estava presente na lista tinha ordem linear, assim como a de remoção de um objeto. Como todos os objetos que fariam parte da fila eram conhecidos de antemão, bastou adicionarmos a cada um deles a posição deles no heap. Sendo assim, conseguimos verificar a presença do objeto tem tempo constante e removê-lo em tempo logarítmico.

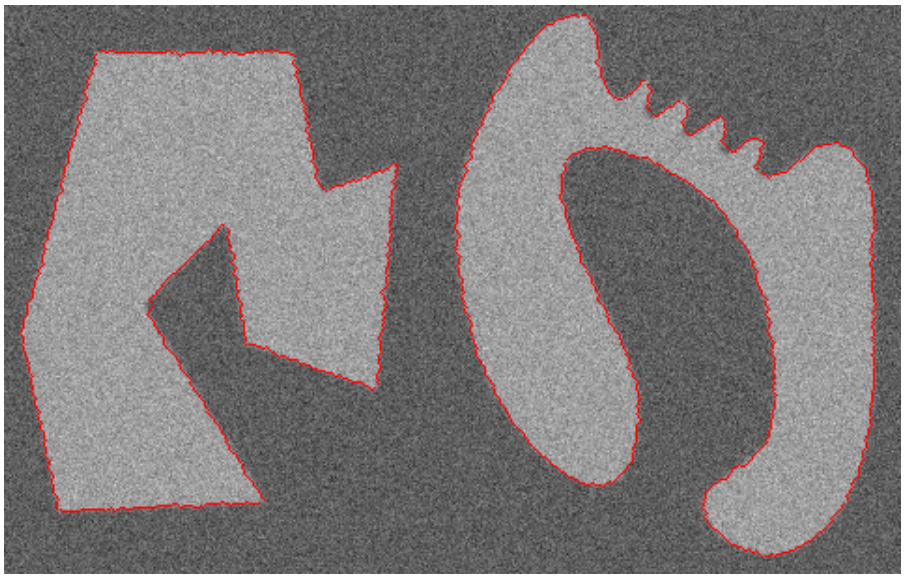
Este algoritmo, por si só, não provê uma grande interatividade e bons resultados. É difícl, a partir da escolha das sementes, ter boa idéia do resultado da segmentação, é difícil saber a que distância devemos colocar as sementes para termos bons resultados e é cansativo termos que recomeçar a segmentação do zero por termos escolhido uma semente errada.

Para superar tais problemas, podemos implementar alguns recursos descritos em [3]. O primeiro é chamado “live-wire”: como, ao selecionarmos uma semente, são calculados os caminhos de custo mínimo de todos os pixels da imagem até aquela semente, podemos, conforme o usuário move o mouse, indicar qual seria a linha de segmentação da semente até o ponto onde o ponteiro do mouse está. O segundo, chamado “path-cooling” trata-se de fazer o próprio algoritmo marcar sementes conforme o mouse se move e uma parte do caminho computado permanece constante. Ambos recursos estão presentes em conhecidas ferramentas de edição de imagens e tornam o processo muito mais interativo.

3.2.2 Resultados



(a) Imagem Orignal



(b) Imagem Segmentada

Figura 3.5: Sis: Resultados para a imagem “noise”

(a) Imagem Orignal (b) Imagem Segmentada

Figura 3.6: Sis: Resultados para a imagem “clown”

3.3 Implementação

3.3.1 Motivação

A ferramenta construída neste trabalho tem como objetivo não só possibilitar a segmentação através dos algoritmos aqui apresentados, mas também permitir que novos algoritmos sejam adicionados a ela de forma fácil. Por esse motivo, ela foi construída utilizando o Framework RCP do Eclipse, que é base para uma das mais conhecidas IDEs de desenvolvimento: Eclipse IDE.

O Framework, baseado na arquitetura de plugins da OSGi[1], além de promover a extensibilidade das aplicações, disponibiliza diversas funcionalidades que ajudam na construção de aplicações complexas – que demandariam muito tempo para serem implementadas do zero. É claro que a utilização do framework traz algumas desvantagens também, entre as quais a mais crítica é o tempo necessário para se familiarizar com ele.

3.3.2 Modelagem

Utilizar a arquitetura de plugins não garante, por si só, a extensibilidade da aplicação. A parte fundamental, que realmente garante tal característica, é a modelagem do sistema.

3.3.3 Algoritmos

Em primeiro lugar, foi definida a interface básica para um algoritmo de segmentação que fosse interativo. A tarefa não é fácil e nem se espera que tal definição englobe todos os possíveis tipos de algoritmo, mas deve ser suficiente para a maioria. Os requisitos básicos levados em conta foram:

- Poder interromper a segmentação e depois continuar
- Poder adicionar novas informações que o algoritmo possa usar, de acordo com a interação do usuário
- Poder obter informações sobre o andamento da segmentação
- Ser independente da ferramenta

[1] Mais informações em www.osgi.org

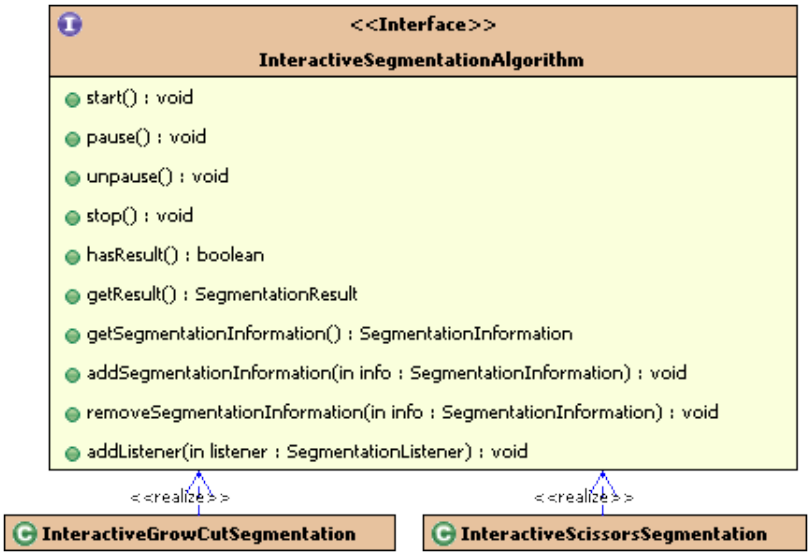


Figura 3.7: Interface para algoritmos de segmentação

Para cumprir com esses requisitos, foram também definidas interfaces que permitem o encapsulamento e a manipulação das informações e obtenção dos resultados de uma segmentação. A idéia é simples: para cada forma de armazenar as informações da segmentação e os resultados dela, seja através de rótulos como no caso do *GrowCut*, seja através de caminhos como no *Sis*, teremos classes que sabem lidar com a adição e remoção de informações e classes que sabem apresentar o resultado de várias modos: linhas de segmentação, imagens extraídas, etc.

Em relação à obtenção de informações sobre o andamento da segmentação, foi utilizado o padrão *Observer*, para que qualquer objeto possa receber os eventos que indicam o andamento do processo. Para mostrar o GrowCut em andamento, por exemplo, nos registramos e ao recebermos o evento de resultado parcial, desenhamos as linhas de segmentação do resultado na tela.

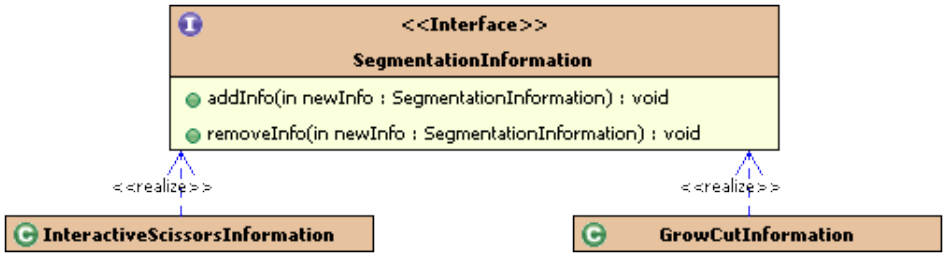


Figura 3.8: Interface para informações de segmentação

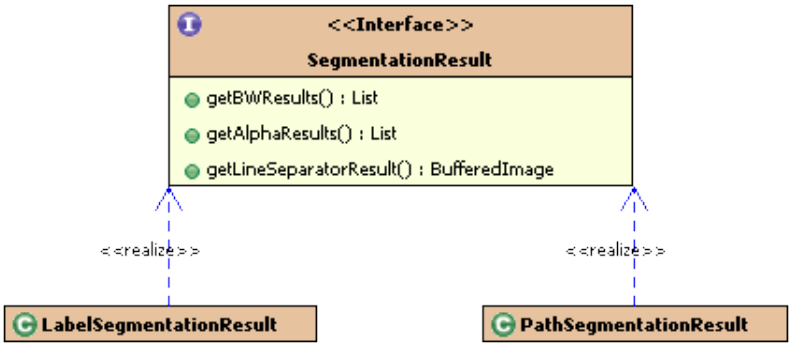


Figura 3.9: Interface para resultados de segmentação

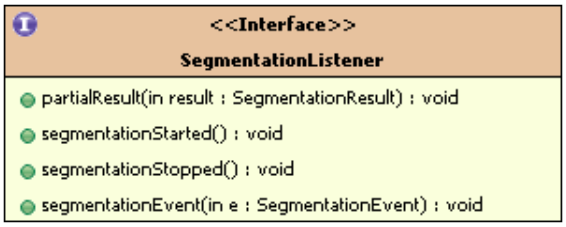


Figura 3.10: Interface para recebimento de eventos de segmentação

3.3.4 Ferramenta

A parte central da ferramenta é a área de interação, chamada de *SegmentationCanvas*, que capta os eventos de mouse, mantém as camadas com as imagens relativas à segmentação, etc. A tarefa principal do *SegmentationCanvas* é repassar os eventos recebidos pra a ferramenta de interação selecionada, que irá realizar as operações necessárias, tais como pintar alguns rótulos na tela e avisar ao algoritmo de segmentação a interação que foi realizada. Desse modo, é possível adicionar novas ferramentas de interação facilmente.

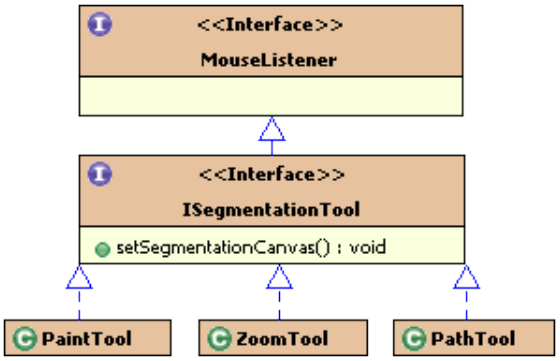


Figura 3.11: Interface para ferramentas de interação

Após implementar o algoritmo de segmentação conforme definido na seção anterior,

para adicioná-lo à ferramenta, é necessário implementar uma interface que indica quais as configurações necessárias para utilizá-lo (que ferramentas de interação utiliza, que camadas são necessárias, etc). Isso deveria ser suficiente para termos um algoritmo adicionado à ferramenta, mas ainda é necessário mexer em partes do código da própria ferramenta para que isso aconteça.

4 Conclusão

Considero que o objetivo principal do trabalho, que era implementar os dois algoritmos de forma eficiente e construir uma ferramenta cujo uso seja viável, foi alcançado. No caso do *GrowCut*, apesar do algoritmo ser um tanto lento e haver a necessidade de bastante interação, o modo como foi implementado permite ao usuário controlar o resultado da segmentação enquanto ela ainda está sendo realizada, contribuindo para segmentação rápidas e arbitrárias. Para o SIS ainda podem ser adicionados recursos que tornem o processo de segmentação mais interativo, mas a parte algorítmica foi implementada de forma bastante eficiente, o que possibilita segmentações muito mais rápidas do que as atestadas no artigo original [4].

A ferramenta é de fácil uso e pode ser utilizada como base para implementação de outros algoritmos, embora ainda possa e deva ser melhorada em vários aspectos, dos quais cito dois:

- Configurabilidade
  Permitir ao usuário modificar configurações básicas para todos os algoritmos, ou para algum algoritmo ou execução em específico.

- Extensibilidade
  Algoritmos novos, e cujo tipo de interação já foi implementado, poderiam ser adicionados à ferramenta sem conhecimento do seu código.

Apesar de, inicialmente, ter proposto uma comparação entre os algoritmos aqui apresentados e o algoritmo implementado em [2], a única comparação possível (que faz sentido) seria quanto ao tempo médio necessário de segmentação para cada imagem. Tal comparação necessitaria de um grande número de imagens, de diversos tipos, e de pessoas para se chegar a um resultado conciso - e não foi feita neste trabalho. O que se pode dizer é que o método apresentado em [2] apresenta resultados excelentes, sendo necessária

pouca interação, para imagens em escalas de cinza, com pouco ruído e texturas e é, nesse caso, melhor do que os algoritmos mostrados neste trabalho.

Por fim, considero uma tarefa importante tentar unir vários algoritmos de segmentação dentro de uma mesma ferramenta, que poderia possibilitar ao usuário não só utilizar esses vários métodos, mas também facilitar comparações entre eles.

5 *Parte Subjetiva*

5.1 Motivação

Fui motivado a realizar este trabalho por ter visto trabalhos de 2006, orientados pela professora Nina, que tinham duas características: estudo de um assunto na área de processamento de imagens e implementação de uma ferramenta: é o tipo de trabalho que eu gostaria de fazer, com um pouco de teoria e aplicação direta.

5.2 Desafios e Frustrações

Inicialmente, planejava ter tempo suficiente para implementar os dois algoritmos escolhidos e construir uma ferramenta para segmentação que fosse flexível e extensível, além de estudar um pouco mais sobre processamento de imagens. Mas, logo que comecei a trabalhar, em meados de Agosto, a falta de tempo se tornou um problema, já que só conseguia fazer algo nos finais de semana e, às vezes, desperdiçava muito tempo com detalhes de implementação.

Excluindo essas frustrações, acredito que o desafio de um trabalho deste tipo, quando realizado individualmente, é fazer todas as partes que o compõe (ferramenta, monografia, apresentação, pôster), mesmo não tendo tanta aptidão para determinado tipo de tarefa.

5.3 Disciplinas importantes

É difícil precisar quais disciplinas do bacharelado foram mais importantes para a realização deste trabalho - até mesmo disciplinas cujo assunto não tem relação direta com o assunto aqui abordado acabam contribuindo de alguma forma. Apesar de não considerar todas disciplinas que são obrigatórias (pelo menos do modo que foram ministradas) essenciais para minha formação, acredito que, no geral, o curso cumpre bem o que eu con-

sidero seu objetivo: dar uma boa base aos alunos para que possam, no futuro, seguir em qualquer área que desejarem, tanto no meio acadêmico, quanto no mercado de trabalho. Mesmo assim, vale citar algumas disciplinas que contribuiram mais para este trabalho:

- MAC0122 – Princípios de Desenvolvimento de Algoritmos
- MAC0328 – Algoritmos em Grafos
- MAC0338 – Análise de Algoritmos
- MAC0438 – Programação Concorrente

As disciplinas de matemática e estatística, principalmente as do início do curso, também são importantes por serem as primeiras matérias em que temos contato com uma linguagem matemática mais formal, necessária para a leitura e escrita de qualquer artigo um pouco mais técnico.

5.4 Interação com o Responsável

Ao meu ver, a interação com a professora Nina foi boa. Apesar de não termos tido tanta interação durante o ano, principalmente porque eu não encontrava tempo para avançar substancialmente no trabalho, ela sempre se colocou à disposição para ajudar no que fosse preciso e, quando eu tive dúvidas, me ajudou a esclarecê-las prontamente.

5.5 Futuro

No futuro, não atuarei na área de processamento de imagens ou área correlata, mas tenho especial interesse na construção de aplicativos facilmente extensíveis. Portanto algumas melhorias serão feitas na ferramenta assim que possível. Em relação aos algoritmos utilizados, também há espaço para melhorias - no caso do SIS, por exemplo, um estudo sobre as constantes utilizadas na função de custo e sobre o σ utilizado na função LoG pode levar a resultados melhores.

Referências Bibliográficas

[1] K. Fukui. Edge extraction method based on separability of image features. *IEICE transactions on information and systems*, 78(12):1533–1538, 19951225.

[2] B. Klava. Ferramenta interativa para segmentação de imagens digitais. 2006.

[3] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 191–198, New York, NY, USA, 1995. ACM.

[4] N. Suetake, E. Uchino, and K. Hirata. Separability-Based Intelligent Scissors for Interactive Image Segmentation. *IEICE TRANS. INF. & SYST.*, E90-D(1), January 2007.

[5] V. Vezhnevets and V. Konouchine. "GrowCut" - Interactive Multi-Label N-D Image Segmentation by Cellular Automata. 2005.