Implementação do Camouflage Images

Eduardo Menges Mattje

João Paulo Vasquez Dias

Universidade Federal do Rio Grande do Sul

Implementação do Camouflage Images

A camuflagem de imagens é um tipo de arte recriacional, ou seja, que busca recriar uma obra com modificações que mantém a essência da peça original. No seu caso, a camuflagem objetiva esconder uma de primeiro plano num plano de fundo, de tal forma que ela permaneça imperceptível e exija esforço consciente para que sejam encontradas.

A habilidade de se esconder está diretamente relacionada com a forma como a percepção humana funciona. Uma possível explicação é a teoria da integração de características, a qual sugere que a visão e percepção humana trabalham em duas fases principais: busca de características e busca de conjunção. A busca de características é uma fase em grande parte paralela que procura por entidades características como cor, borda e textura para uma caracterização rápida e instantânea de figuras. Em contraste, a busca de conjunção é um processo serial e lento, e é responsável pelo reconhecimento e classificação, integrando pistas (espalhadas) de múltiplas características.

No trabalho de Chu et al. (2010), propôs-se uma abordagem algoritmíca para a montagem das imagens camufladas. Com base nesse trabalho, fizemos uma implementação do algoritmo em uma linguagem de programação.

# Técnica

Primeiramente, ambas imagens são convertidas em escala de cinza, e então quantizada. A quantia de tons nesta etapa é vaga e mal explicada no artigo. Após, ocorre a segmentação de ambas as imagens de acordo com os tons de cinza, usando 4-vizinhança; e a imagem do plano de fundo é cortada pelos segmentos do primeiro plano, ou seja, segmentos do fundo que não tem qualquer sobreposição com qualquer segmento do primeiro plano são descartados.

Então, ocorre a construção de um grafo, cujos nodos são os segmentos do primeiro plano, e do plano de fundo. As arestas são formadas da seguinte forma:

* Dois nodos do plano de fundo são conectados caso seus segmentos estejam conectados, ou seja, caso sejam vizinhos, constituindo as arestas de destaque;
* Um nodo do primeiro plano se conecta com os 6 segmentos do plano de fundo mais próximos, medido através da distância euclidiana, constituindo as arestas de imersão.

A fim de realizar a camuflagem, um novo conjunto de luminâncias é designado ao primeiro plano. Sua construção baseia em duas fontes de energia:

* A energia de imersão, que mensura o quão bem o primeiro plano está camuflado no plano de fundo;
* A energia de destaque, que mensura o quão bem os novos valores de luminância retém as diferenças de luminância do conjunto original.

Desta forma, deseja-se diminuir a soma ponderada dessas energias, tarefa complexa e não trivial, pois há uma dependência mútua entre o conjunto novo de luminâncias e as energias. O artigo aponta que este problema é análogo ao problema de marcação em visão computacional que otimiza uma função de energia. Assim, utilizou-se o algoritmo *alfa-beta swap* em conjunto a um *graph cut*, processo iterativo de minimização de energia.

## Síntese de texturas

Com a luminância corretamente designada aos segmentos, passa-se ao processo de síntese de texturas, isto é, adicionar os detalhes das texturas do plano de fundo, preenchendo, assim, o primeiro plano. O primeiro passo é construir uma banco de dados de amostras de texturas do plano de fundo, onde cada entrada é constituída de uma amostra de textura, e uma amostra de luminância, ambas representando a mesma região, e amostradas nas fronteiras dos segmentos. Já no primeiro plano, o processo de amostragem também é realizado nas fronteiras dos segmentos, de tal forma que, para cada textura, encontra-se uma melhor textura correspondente no banco de dados construído, levando em conta a similaridade entre as texturas, e a distância onde se encontram.

Tendo as fronteiras delimitadas, o interior é preenchido através do crescimento das texturas das fronteiras, copiando-as numa técnica de “*texture-by-numbers*”.

Uma série de outras melhorias podem ser empregadas, como a adição de segmentos de distração, ou a deformação do plano de fundo para se moldar melhor ao primeiro plano.

# Implementação

Inicialmente, começamos a implementar o artigo na linguagem de programação Rust, utilizando as bibliotecas de processamento de imagens disponíveis. As etapas de luminância e quantização foram fáceis, vistos que já foram anteriormente implementadas na disciplina.

## Segmentação

Na etapa de segmentação, houve um dilema quanto ao modo como eles deveriam ser armazenados. Caso fizéssemos na forma de tabela, utilizando labels, a utilização de memória seria inecificente, porém dificultaria significativamente a utilização dos segmentos nos outros processos. Desta forma, utilizou-se a seguinte abordagem:

* Cada segmento é armazenado na forma de um hash map, onde a chave é uma coordenada y, e o valor associado é um vetor.
* Neste vetor, são armazenadas as faixas que o segmento ocupa na linha correspondente à coordenada y.

Ou seja, para uma dada coordenada 7, as faixas [6..10, 14..20] indicam que, na imagem, os pixels da linha de 6 a 10, e de 14 a 20 na linha 7 pertencem ao segmento tal.

No processo de corte dos segmentos sobrepostos, teve de se implementar métodos para testar caso haja sobreposição, tarefa não muito complexa, apesar de computacionalmente cara, pois para cada linha correspondente e para cada faixa, é necessário testes com todas as outras faixas. Felizmente, devido à nossa eficiente implementação dos segmentos, não há grande peso na performance, pois as comparações não passam de números inteiros, que os processadores atuais têm ampla facilidade.

## Grafo

A construção do grafo foi trabalhosa. Primeiro, teve de se implementar métodos que determinam se dois segmentos estão conectados, teste que não é muito diferente do teste de sobreposição previamente feito. Então, surgiu a questão de medir a distância euclidiana entre os segmentos, outro momento de incerteza quanto à implementação, porque não sabíamos como calcular o centroide deles. Após conversar com o *ChatGPT*, escolheu-se determiná-lo através da média das coordenadas x e y, ponderada pela transparência dos pixels.

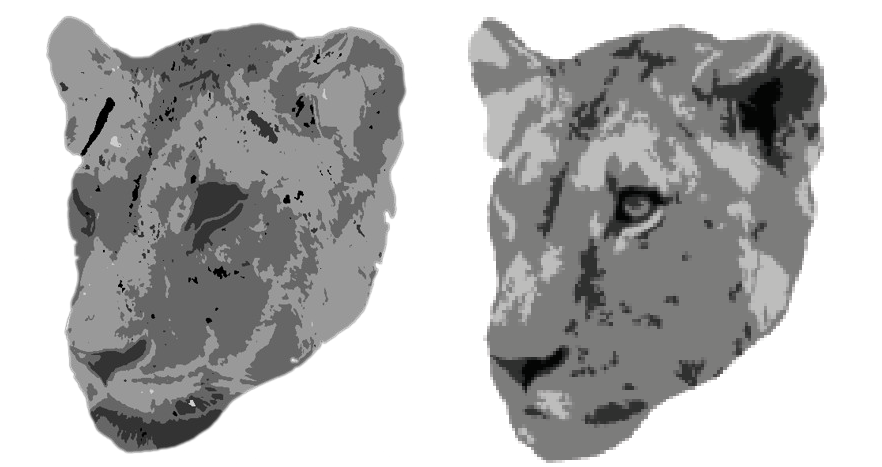
## Otimização

A etapa de otimização foi desafiadora. Trata-se de um problema de otimização de energia NP-hard. Basicamente, deve-se escolher a melhor maneira de distribuir as luminâncias do *background* entre os segmentos do *foreground*, de maneira a minimizar a energia combinada do grafo montado na etapa anterior.

O artigo esclarece que o problema de minimização encontrado é análogo ao problema de *Bayesian labeling of first-order markov random field (MRF),* e portanto a abordagem utilizada para resolução desse problema de *labelling* foi o algoritmo *alpha-beta swap,* proposto por Boykov et al. (2001)*.* A implementação deste algoritmo se mostrou um desafio em Rust, onde tudo teria de ser implementado do zero, sem auxílio de bibliotecas auxiliares. Com isso, desistimos de prosseguir utilizando essa linguagem, e seguimos o trabalho utilizando Python, onde já havia implementação de muitas das operações necessárias para o *alpha-beta swap*, como o algoritmo de *maxflow*/*mincut* proposto por Boykov e Kolmogorov (2004) e disponibilizado na biblioteca PyMaxFlow. Essas operações, pela própria natureza da linguagem, acabaram por serem menos performáticas. Essa decisão de mudar para Python, também impactou na nova implementação das etapas anteriores na nova linguagem.

# Resultados

Os resultados obtidos foram satisfatórios na parte de definição de luminância, atingindo resultados semelhantes ao do artigo:



Respectivamente: nosso resultado, resultado do artigo

Tela de celular com publicação numa rede social

Descrição gerada automaticamente com confiança média

Respectivamente: imagem original, quantizada e com otimização de luminância

Contudo, com a incapacidade de implementar a transferência de textura conforme o artigo, e apenas com a atribuição da nova luminância, os resultados obtidos não foram tão satisfatórios, pelo menos não quanto os obtidos no artigo. Vide imagem abaixo:

Montanha de pedras

Descrição gerada automaticamente

Respectivamente: nosso resultado, resultado do artigo

# Conclusão

O artigo é muito complexo, e abrange diferentes áreas do processamento de imagens, incluindo a minimização de energia de imagens, transferência de textura, e segmentação. Parte dele não foi bem explicada, deixando muita margem à interpretação e adivinhação, que certamente atrasou sua implementação. Acreditamos que o caro Chu-sama deveria ser mais objetivo em sua explicação.

Uma dificuldade encontrada foi com a linguagem Rust, com que começamos o projeto, visto que não há implementação para o algoritmo de max flow, tampouco de transferência de textura. A mudança para Python trouxe consigo todos os prós e contras da linguagem, incluindo a baixa performance e a ocorrência de erros em tempo de execução; aqueles incluem o grande arcabouço das bibliotecas de processamento de imagens já disponíveis na linguagem, e os algoritmos para minimização de energia.

Apesar de não conseguirmos implementar o artigo em sua totalidade, as etapas implementadas serviram de grande aprendizado em termos de:

• estrutura de dados, no que tange o armazenamento dos segmentos na linguagem Rust, e sua implementação;

• minimização de energia, utilizando um avançado algoritmo para grafos;

• ¬técnicas de transferência de texturas, apesar de nossa incapacidade quanto à sua implementação;

• escolha de ferramentas, nem sempre nossa linguagem favorita é a que melhor atende às nossas necessidades, seria melhor começar o projeto direto em Python e atingir a parede texturizada com tijolos mais cedo, às vezes, o que a gente gosta, não é o melhor a se fazer;

• descoberta dos nossos limites como estudantes e programadores.

Referências

Chu, H. K., Hsu, W. H., Mitra, N. J., Cohen-Or, D., Wong, T. T., & Lee, T. Y. (2010). Camouflage images. *ACM Trans. Graph.*, *29*(4), 51-1.