UNIVERSIDADE FEDERAL DE PERNAMBUCO

GRADUAÇÃO EM ENGENHARIA DA COMPUTAÇÃO
CENTRO DE INFORMÁTICA

Estudo e implementação de um algoritmo de síntese de texturas utilizando cortes em grafos

TRABALHO DE GRADUAÇÃO

Aluno: Pedro Augusto Bello (pab2@cin.ufpe.br)

Orientador: Silvio Barros Melo (sbm@cin.ufpe.br)

Recife, 14 de dezembro de 2010

Agradecimentos

Agradeço a toda minha família, em especial, a minha esposa pelo amor e companheirismo em todas as horas, a minha mãe, também pelo amor e por ter me sustentado enquanto eu fazia os projetos da graduação sem ganhar dinheiro nenhum, ao meu pai por sempre se preocupar com meu futuro, e ao meu irmão por sempre brigar comigo pela posse do computador (ele deixou de jogar várias vezes para eu poder fazer esse trabalho).

Agradeço também a equipe da maratona de programação do Centro de Informática, por terem feito parte daquilo que considero minha maior fonte de aprendizado e conquistas. Um agradecimento especial para a professora Liliane por conduzir este projeto com tanta dedicação e àqueles que foram meus companheiros de time: Pablo, Eduardo, Hallan, Victor e Maíra (que por sinal também é minha esposa).

Ao professor Silvio Melo por ter me orientado no desenvolvimento deste trabalho.

E também a todos aqueles que direta ou indiretamente contribuíram para minha formação, tanto pessoal quanto profissional.

Resumo

Neste documento, damos uma introdução ao tema da síntese de texturas e descrevemos alguns algoritmos importantes para o entendimento do estado da arte do mesmo. Entre eles, o algoritmo proposto por Kwatra et. Al (2003) foi escolhido como foco principal do trabalho. No método de Kwatra e seus companheiros, pedaços da amostra são copiados para a saída de acordo com o resultado de um algoritmo de cortes em grafos. Uma implementação deste método foi desenvolvida, e mostramos aqui os resultados adquiridos e possíveis melhorias futuras.

Palavras-chave: Síntese de texturas, cortes em grafos, processamento de imagens, realismo, computação gráfica.

Lista de Figuras

Sumário

1. Introdução

Computação Gráfica é o ramo da Ciência da Computação que trata da geração e manipulação de imagens digitais [Shirley et al. 2005]. O amplo desenvolvimento nesta área causou grande impacto em uma série de mídias e indústrias, como filmes, animações e vídeo games.

Uma das principais preocupações na síntese de imagens computadorizadas é a representação do mundo tridimensional na tela bidimensional de um computador. Porém, a modelagem geométrica de objetos 3D não é o único desafio da Computação Gráfica. A obtenção de imagens "realísticas" impressas no monitor é alvo permanente de grande parte dos pesquisadores [Amanatides 1987, Kaufmann 1988]. Apesar do alto grau de complexidade matemática, o traçado bem feito de figuras geométricas complicadas pode não ser suficiente para retirar do observador a sensação de estar de frente com uma cena não real [Figura 1.1-a].

Neste contexto de obtenção do realismo se encontra o mapeamento de texturas. Uma textura pode ser usada para adicionar propriedades às superfícies dos objetos que não são capturadas apenas por sua geometria, como efeitos de cor, transparência, viscosidade e até mesmo representação de características biológicas como a pele de um animal [Walter e Fournier 1998]. A Figura 1.1-b mostra um exemplo de uma cena texturizada.

(a) (b)

Figura 1.1: Importância de texturas na obtenção do realismo. (a) cena antes do mapeamento de texturas, (b) cena já texturizada, apresentando um maior grau de realismo.

Texturas podem ser obtidas através de várias fontes, como fotos escaneadas ou até mesmo desenhadas à mão. Entretanto, a forma mais comum de geração de texturas é a síntese automática a partir de exemplos. Nesta abordagem, o desafio consiste em gerar a partir de uma imagem de entrada, uma imagem de saída que pareça a observadores humanos como vindo "da mesma textura".

Este trabalho tem como objetivo estudar e analisar os algoritmos mais conhecidos de síntese de texturas a partir de amostras. Também foi

implementado uma das técnicas de síntese apresentadas, a técnica escolhida foi a proposta por Kwatra et al. (2003). A decisão de focar no trabalho proposto por Kwatra e seus companheiros veio da idéia de estender o trabalho de Brayner (2009), sobre compressão de texturas. Em seu trabalho, Brayner mostra técnicas para comprimir texturas geradas a partir de uma série de algoritmos de síntese. Porém ele fala das dificuldades existentes na criação de uma técnica de compressão em imagens geradas pelo algoritmo de Kwatra.

Foi no intuito de obter um forte embasamento no estado da arte do problema de síntese de texturas, para poder posteriormente dar uma contribuição de maior valor a área, como por exemplo, complementando o trabalho de Brayner com a criação de uma técnica de compressão de texturas a partir do algoritmo de Kwatra, que se originou a idéia deste trabalho. Apesar de a motivação inicial ter vindo de um trabalho sobre compressão de texturas, o estudo do estado da arte trouxe uma série de outras possibilidades para extensões futuras, que serão descritas no fim deste documento.

No capítulo 2, definiremos alguns conceitos referentes ao problema de síntese de texturas e daremos uma introdução geral à área. No terceiro capítulo explanaremos os principais algoritmos que descrevem o estado da arte da síntese de texturas a partir de amostras. Apesar de existir uma grande variedade de algoritmos com este propósito, mostramos aqueles que foram mais fundamentais e inovadores no desenvolvimento do estado da arte.

Dentre os algoritmos descritos, o proposto por Kwatra et al. (2003), o qual é o foco principal deste trabalho, recebeu um capítulo separado (o quarto). Este inicia com uma breve introdução a cortes em grafos, problema de otimização que é fundamental na técnica de síntese.

No quinto capítulo falamos sobre a implementação do algoritmo e seus resultados, para então concluir o trabalho e citar possíveis trabalhos futuros.

2. Conceitos

Neste capítulo falaremos em mais detalhes sobre o problema de síntese de texturas, e definiremos alguns conceitos ainda não muito claros, como por exemplo, o conceito de texturas.

2.1. Texturas

Apesar de já termos usado o termo “textura” neste trabalho, ainda não o definimos formalmente. Sobretudo porque dificilmente se encontra na literatura uma definição concreta do termo, e também porque a palavra surge com significados diferentes em contextos diferentes. Comumente utilizamos a palavra textura para referenciar algum padrão táctil de uma superfície, porém essa noção não se enquadra em nosso contexto.

Uma noção geral e intuitiva de textura é defini-la como uma propriedade visual, que contém características da superfície de um objeto e permite o reconhecimento de algum material, como por exemplo, grama ou mármore [Walter 1991].

Em computação gráfica, quando falamos em *mapeamento de texturas* [Blinn e Newel 1976, Catmull 1974], estamos nos referindo ao processo de aplicar uma imagem (textura) na superfície de um objeto, com o intuito de dar a ele características que o tornam mais real. Logo nesse caso, textura pode ser compreendida como qualquer imagem que é mapeada a um objeto.

Neste trabalho, usamos uma definição mais restrita que a utilizada em mapeamento de texturas. Em nosso contexto, texturas são imagens com caráter estacionário e que possuem algum padrão que se repete ao longo de sua extensão. Texturas podem apresentar caráter puramente estocástico (como a areia da praia) ou totalmente determinístico (um chão ladrilhado).

2.2. Classificação de texturas

Texturas são geralmente classificadas em regulares e estocásticas, dependendo de seu grau de estruturação e aleatoriedade. Texturas regulares são aquelas que apresentam padrões que se repetem de forma regular e definida por toda a imagem. Texturas estocásticas são mais próximas a sinais de ruído, sem apresentarem uma estrutura aparente. Entre esses dois extremos, existem também as texturas quase-regulares, que são pequenas distorções na estrutura rígida das texturas regulares, como pequenas mudanças de cor ou na forma, e as texturas irregulares que apesar de possuírem padrões que se repetem claramente, os mesmo não aparecem distribuídos regularmente pela imagem [Lin et al. 2004] [Figura 2.1].

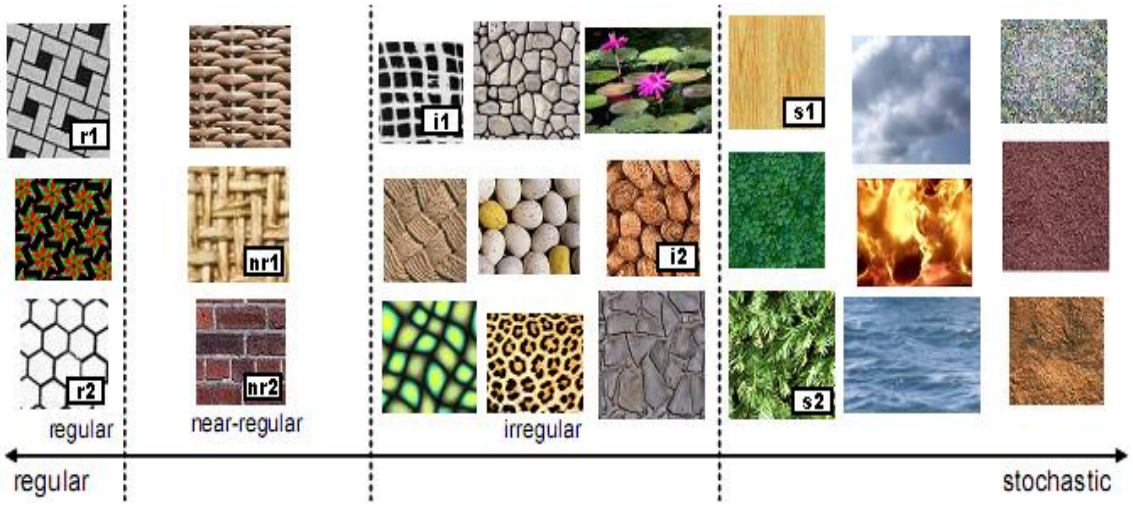


Figura 2.1: Espectro de texturas. Variam linearmente de estocásticas a regulares dependendo do grau de estruturação e aleatoriedade. (Figura retirada de Brayner [2009])

2.3. Síntese de texturas

Síntese de texturas tem sido um permanente alvo de pesquisa em computação gráfica há muitos anos. Embora a criação de texturas possa ser feita através de métodos simples como fotografias escaneadas ou desenhos feito a mão, estes métodos geralmente são deixados de lado, uma vez que pode ser complicado alcançar realismo com desenhos manuais, e que pouco provavelmente se obtém fotos na escala desejada para o mapeamento da textura [Wey e Levoy 2000]. Portanto a síntese digital de texturas é o método mais comum para a geração destas imagens.

Existem duas abordagens principais para a síntese de texturas, a procedural [Fleischer et al. 1995, Turk 1991, Walter e Fournier 1998] e a baseada em amostras [Efros e Leung 1999, Wey e Levoy 2000, Efros e Freeman 2001, Kwatra et al. 2003].

Algoritmos procedurais frenquentemente são rápidos e consomem pouca memória, porém estes métodos são apenas simuladores dos processos reais de geração das texturas, portanto geralmente eles são restritos a produzirem imagens em um domínio específico, como peles de mamíferos [Figura 2.2] [Walter e Fournier 1998] ou fluidos [Kwatra et al. 2007].

(a) (b)

Figura 2.2: Exemplo de síntese de pele de guepardos. (a) mostra a imagem real, enquanto (b) mostra a imagem sintetizada. (Figura extraída do trabalho de Walter e Fournier [1998])

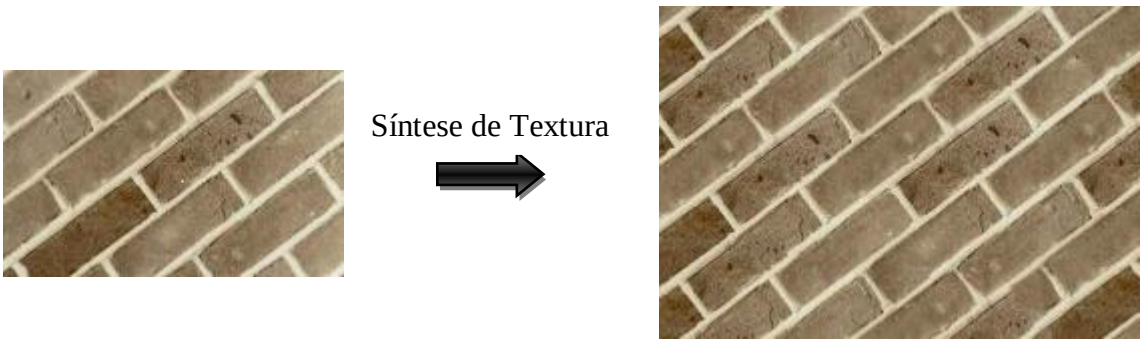
Algoritmos de síntese baseados em amostras podem gerar imagens provenientes de uma grande variedade de texturas de entrada. Uma vez que o algoritmo recebe uma textura como “exemplo”, técnicas desse tipo não precisam se resumir a alguns domínios, portanto possuem uma gama maior de aplicações que os algoritmos procedurais.

Apesar de algoritmos de síntese de texturas restritos a um domínio específico serem ainda assunto de bastante pesquisa no meio acadêmico, o restante deste trabalho se concentrará em síntese de texturas baseada em amostras, e sempre que o termo “síntese de textura” for utilizado, estará referenciando esta abordagem.

2.4. Síntese de texturas a partir de amostras

Síntese de texturas a partir de amostras pode ser definida como o processo de geração de uma textura a partir de um exemplo de textura (geralmente menor que a imagem gerada) dado como entrada. O objetivo da síntese é produzir uma imagem que seja percebida aos olhos humanos como vinda “da mesma textura” da imagem de entrada [Wey e Levoy 2000] [Figura 2.3]. Os maiores desafios dos algoritmos de síntese são:

a) A definição de um modelo formal que capture as propriedades de uma textura. Essas propriedades podem ser estruturais e estocásticas. A síntese se guiará neste modelo para tentar produzir imagens similares à amostra.
b) Baseado no modelo, definir um procedimento de amostragem, que construa a nova textura a partir da textura exemplo. A eficiência do método de amostragem determina a eficiência da síntese.



(a) (b)

Figura 2.3: Síntese de Texturas. Um algoritmo de síntese recebe uma textura de entrada (a), e gera uma textura (b) que pareça similar a amostra e com tamanho especificado pelo usuário.

2.4.1. Campos aleatórios de Markov

Apesar de uma série de modelos terem sido propostos (reação-difusão [Turk 1991], fractais [Worley 1996], histogramas de freqüência [Heeger e Berger 1995]) para a representação de uma textura, sem dúvida o modelo mais conhecido por conseguir representar a maioria das texturas com precisão é o baseado em *Campos aleatórios de Markov* [Efros e Leung 1999, Wey e Levoy 2000, Efros e Freeman 2001, Kwatra et al. 2003].

Modelos baseados em campos aleatórios de Markov modelam texturas como um processo local e estacionário. Dizer que uma imagem é gerada por um processo local significa que cada pixel é caracterizado por um pequeno conjunto de outros pixels em sua vizinhança. Essa caracterização ser a mesma para todos os pixels da imagem determina um processo estacionário.

Na figura 2.4 ilustramos melhor a idéia de representar uma textura por um campo aleatório de Markov. Suponha que lhe fosse permitido olhar apenas através de uma pequena janela da imagem, em uma imagem estacionária, deslizar uma janela de tamanho apropriado pela imagem manterá a porção aparente similar.

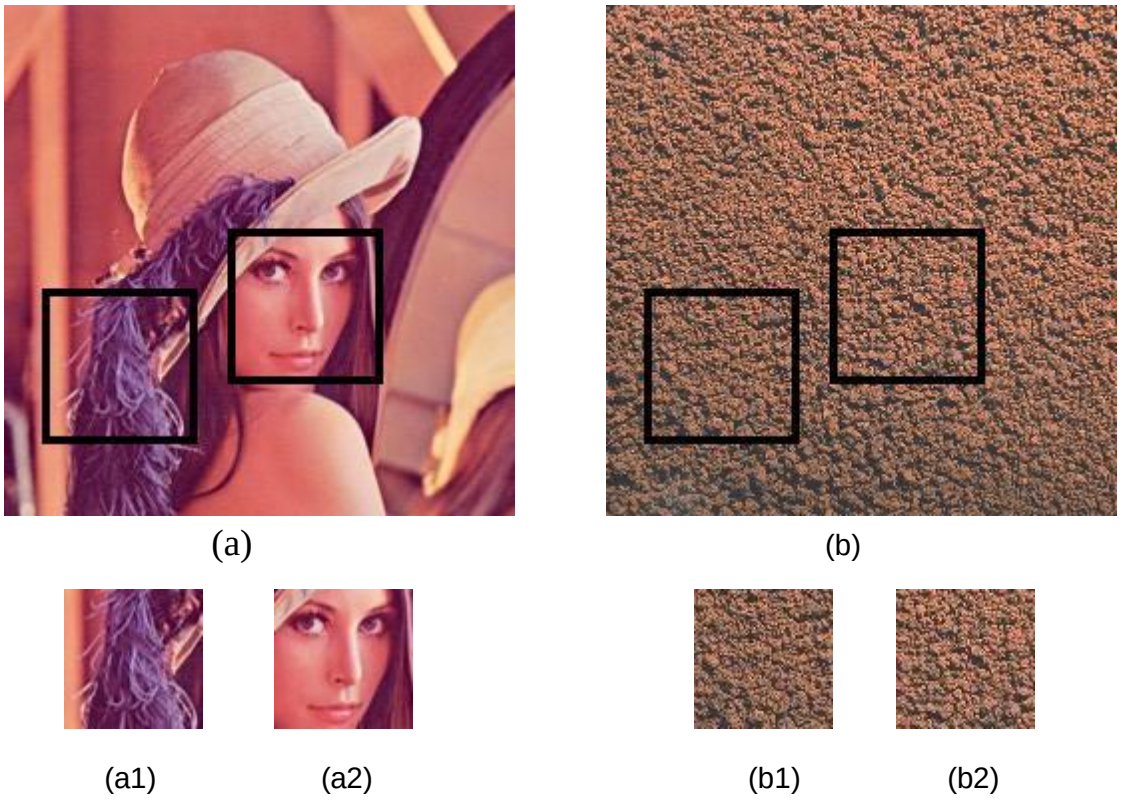


Figura 2.4: Um exemplo de como texturas diferem de imagens normais. (a) mostra uma imagem normal, enquanto (b) mostra um exemplo de textura. Porções (a1) e (a2) da imagem não-textura aparentam ser realmente distintas. Enquanto porções (b1) e (b2) da textura apresentam um alto grau de similaridade.

Muitos algoritmos de síntese de texturas baseados em campos aleatórios de Markov foram desenvolvidos. As formas mais conhecidas de síntese são a síntese baseada em pixels e baseada em *patches* (retalhos). A maioria dos algoritmos de síntese baseados em campos aleatórios de Markov podem ser classificados em um destes dois grupos.

2.4.2. Síntese baseada em pixels

Algoritmos baseados em pixels sintetizam a nova textura um pixel de cada vez, com o valor do novo pixel sendo calculado a partir dos valores dos pixels já sintetizados em sua vizinhança [Efros e Leung 1999, Wey e Levoy 2000]. Na hora de escolher o valor do próximo pixel, se compara a vizinhança do mesmo com a vizinhança dos pixels da imagem de entrada utilizando alguma métrica. O novo valor do pixel será determinado pelos valores dos pixels de entrada que possuem vizinhança mais similar a sua. O tamanho da região de vizinhança é um parâmetro especificado pelo usuário. Para obter resultados satisfatórios é necessário que o tamanho da vizinhança seja suficientemente grande para capturar os padrões da textura de entrada. Entretanto, grandes regiões de vizinhança tendem a diminuir a performance do algoritmo. Estratégias de síntese multi-resolução podem ser usadas para conseguir capturar elementos estruturais de tamanho razoável sem prejudicar a performance [Heeger e Berger 1995].

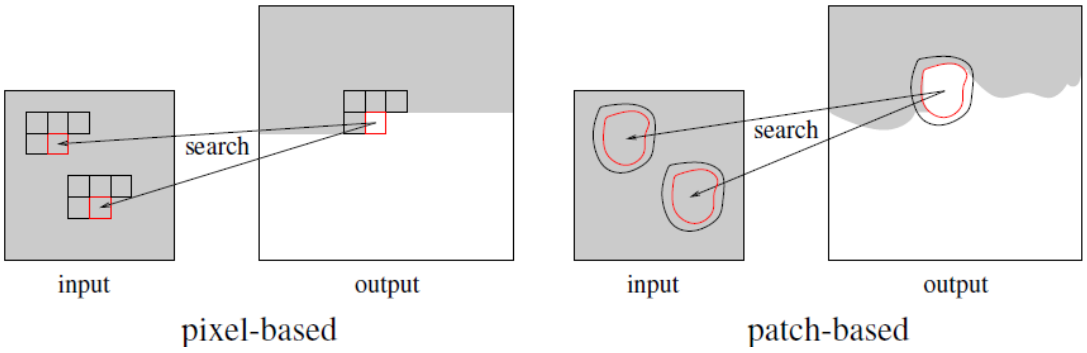


Figura 2.5: Comparação entre abordagens de síntese. (a) mostra um algoritmo baseado em pixels e (b) mostra a síntese por patches. (Figura retirada de Wei et al. [2007])

2.4.3. Síntese baseada em *patches*

Uma característica da síntese baseada em pixels, é que muitas vezes um pixel possui poucas opções de valor a serem associadas a ele. Ou seja, durante o processo de síntese, o valor do próximo pixel já está determinado pelo que já foi construído. Este fator associado ao alto custo computacional da abordagem pixel a pixel traz a idéia de copiarmos pedaços (*patches*) da entrada de uma única vez. A figura 2.5-b ilustra o processo de cópia dos *patches*. A principal diferença entre a abordagem baseada em pixel e a baseada em *patches* é na forma que ocorre a cópia

da unidade de síntese. Na primeira abordagem, o pixel é simplesmente copiado para a textura de saída. Entretanto, na síntese por *patches* é permitido que haja interseção entre o novo *patch* que será inserido e a textura já sintetizada. A estratégia de recorte na região de sobreposição é um ponto-chave na qualidade visual nesse tipo de algoritmo [Figura 2.6].

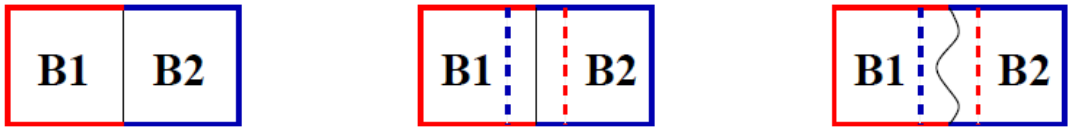


Figura 2.6: Exemplo de como ocorre o recorte na area de sobreposição. (a) mostra dois patches separados. (b) área de sobreposição entre os patches. (c) forma ótima de recorte entre os patches. (Figura extraída de Efros e Freeman [2001])

3. Algoritmos de síntese

Na última decada, uma série de trabalhos foram publicados na área de síntese de texturas baseada em amostras. Nesta seção, descreveremos algumas destas técnicas publicadas. Mostraremos os trabalhos que causaram mais impacto e são essenciais no entendimento do estado da arte do problema de síntese de texturas. Os dois primeiros algoritmos descritos são de síntese pixel a pixel, enquanto o último é baseado em *patches.*

3.1. Síntese com Efros e Leung

A idéia de usar um exemplo de textura real para gerar texturas maiores tem uma longa tradição em Computação Gráfica e Processamento de Imagens. Alguns dos trabalhos mais antigos na área são Fu e Lu (1978), Cross e Kain (1983) e Monne et al. (1981). Diferentemente dos métodos mais recentes, esses primeiros trabalhos abordavam o problema de síntese através de métodos probabilísticos. O trabalho de Efros e Leung (1999) foi pioneiro no uso de amostras de imagem para guiar diretamente o processo de síntese. Enquanto os algoritmos antigos usavam a imagem de entrada como fonte para extração de características que iriam guiar a síntese, Efros e Leung propuseram montar a textura de saída por copiar pixels diretamente da textura de entrada.

Em seu famoso artigo, *A Mathematical Theory of Communication,* Shannon (1948) mostrou uma forma interessante de produzir um texto que se assemelhe a um texto em inglês usando *n-grams.* Um conjunto de n caracteres consecutivos em um texto formam um *n-gram.* A idéia é usar as últimas n letras do texto, para determinar a probabilidade de cada letra do alfabeto ser a próxima, construindo assim uma cadeia de Markov de *n-grams*. Usando uma amostra razoável de textos da língua inglesa, cria-se a tabela de probabilidades para cada *n-gram*.

Inspirados na proposta de Shannon para síntese de textos, Efros e Leung desenvolveram um algoritmo de síntese de texturas, no qual para cada novo pixel a ser gerado, apenas uma quantidade pequena de pixels em sua vizinhança seriam considerados (note que essa abordagem também adota o modelo de campos aleatórios de Markov). Ao contrário do caso de textos, é inviável em uma síntese de imagens criar-se uma tabela para cada possível combinação de pixel de vizinhança, logo outra estratégia foi criada.

O algoritmo basicamente percorre a imagem de saída do centro para as bordas, preenchendo os pixels um a um. Para se escolher o valor do pixel atual, leva-se em consideração a vizinhança já processada do pixel. A vizinhança é definida como um quadrado centrado no pixel atual, e de lado definido pelo usuário, devendo representar o grau de aleatoriedade da textura. O tamanho do quadrado deve também ser suficiente para conseguir agregar os padrões estruturais da entrada.

Uma vez que temos a vizinhança do pixel, procuramos na imagem de entrada por regiões de mesmo tamanho, e que sejam similares a

vizinhança atual de acordo com alguma métrica específica [Figura 3.1]. Seja *p* o pixel na imagem de saída cujo valor queremos calcular, e chame sua vizinhança de *N(p).* Para cada pixel *q* da amostra, nós calculamos a distância entre as vizinhanças de *q* e *p* e a chamamos de $d\big(N(q), N(p)\big)$. Seja d_{best} o valor da menor distância entre alguma região na entrada e *N(p).* O algoritmo seleciona randomicamente um pixel da entrada dentre todos aqueles cujas distâncias calculadas foram menores ou iguais a um limiar, a saber, serão incluídos na escolha todos os pixels *q,* tal que

$$d\big(N(q), N(p)\big) \leq (1 + \epsilon) * d_{best}$$

onde ϵ foi escolhido como 0.1 no trabalho original. *p* receberá o valor do pixel escolhido neste processo. Vale salientar que a distribuição de probabilidade na seleção não precisa ser uniforme, podendo também ser ponderada pelo *d* de cada região.

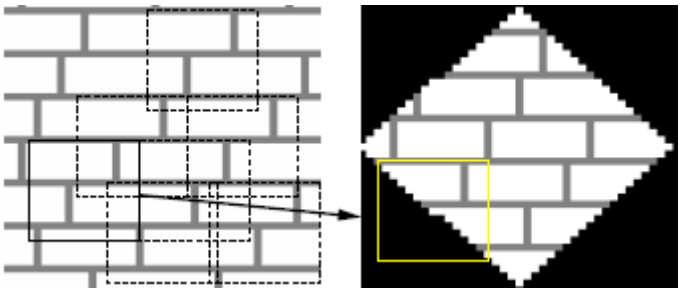


Figura 3.1: Algoritmo de Efrous e Leung. O algoritmo percorre a imagem do centro para as bordas preenchendo os pixels. A vizinhança do novo pixel escolhido é então comparada com regiões de mesmo tamanho na imagem de entrada.

Ainda não foi mencionada uma métrica *d* que seja razoável para nosso propósito. Uma opção é a soma dos quadrados das diferenças entre os pixels correspondentes das regiões. Porém esta métrica não pondera diferentemente pixels que estão mais próximos do centro da região, uma vez que é necessário preservar a *localidade* da textura, deve-se dar maior peso a erros de pixels que estão mais próximos do centro. Isso pode ser obtido usando um núcleo Gaussiano para ponderar os pixels na região.

Alguns detalhes do algoritmo ainda não foram especificados, como por exemplo, determinar quais valores serão atribuídos aos pixels na vizinhança de *p* que ainda não foram processados. Isso é facilmente resolvido se apenas os pixels já visitados forem considerados no cálculo da distância, e posteriormente normaliza-se a distância dividindo o resultado pelo número de pixels envolvidos no cálculo. Outro ponto em aberto é o que acontece no inicio do algoritmo, quando nenhum pixel tem ainda seu valor calculado. Um quadrado 3x3 no centro da imagem de saída pode ser inicializado com uma região da entrada de mesmo tamanho randomicamente escolhida, e o algoritmo inicia a partir dos pixels ao redor deste quadrado central.

Apesar de simples, o trabalho de Efros e Leung gerou bons resultados para uma vasta gama de texturas, além de ter dado uma contribuição importantíssima ao campo, tendo influenciado uma série de trabalhos que o sucederam.

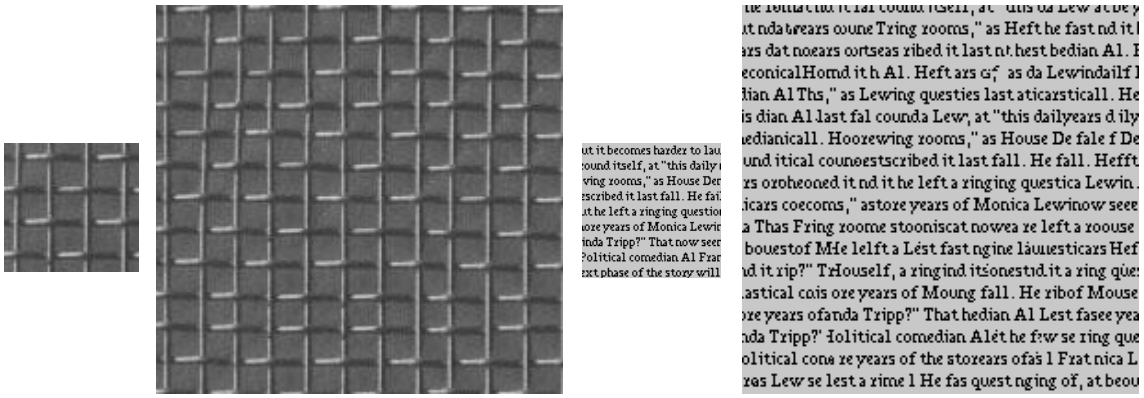


Figura 3.2: Resultados de síntese de texturas com Efros e Leung (1999).

3.2. Síntese multi-resolução com Wey e Levoy

O trabalho de Wey e Levoy (2000) conseguiu um ganho de desempenho considerável em relação aos trabalhos anteriores. Embora o núcleo do algoritmo seja muito similar ao criado por Efrous e Leung (1999), Wey e Levoy propuseram uma modificação que permite uma otimização usando quantização vetorial estruturada em árvores (*Tree-structured Vector Quantization)*. O ganho de performance proporcionou a aplicação de síntese de texturas em áreas ainda não exploradas por algoritmos anteriores como edição de imagens e síntese de texturas temporais (tipo de textura 3D que representa bem fenômenos naturais como fogo, água, fumaça, etc.).

Similarmente ao proposto por Efros e Leung, o algoritmo gera a saída pixel a pixel, porém ao invés de ir do centro para as bordas, ele percorre a saída de cima para baixo e da esquerda para a direita (*scanline order).* A imagem de saída é inicializada como um sinal de ruído, no qual cada pixel recebe um valor aleatório. Os pixels então são sintetizados um a um, e para cada um deles, busca-se na textura de entrada regiões que sejam próximas a vizinhança deste pixel (perceba a semelhança com a abordagem de Efros e Leung). A métrica usada para a comparação entre regiões foi também a soma dos quadrados das diferenças.

Uma diferença importante entre este método e o de Efros e Leung é que neste, o pixel de entrada cuja vizinhança mais se aproxima a vizinhança do pixel atual, será sempre o escolhido (no trabalho de Efros e Leung era proposta uma escolha probabilística). Na síntese proposta por Wey e Levoy, o único elemento randômico no processo é a inicialização da imagem como um ruído. Este ruído inicial é fundamental na síntese dos primeiros pixels processados, pois estes devem considerar o ruído como uma vizinhança válida. Nos pixels posteriores,

apenas vizinhos já processados devem ser considerados, assim como feito no método de Efros e Leung [Figura 3.3].

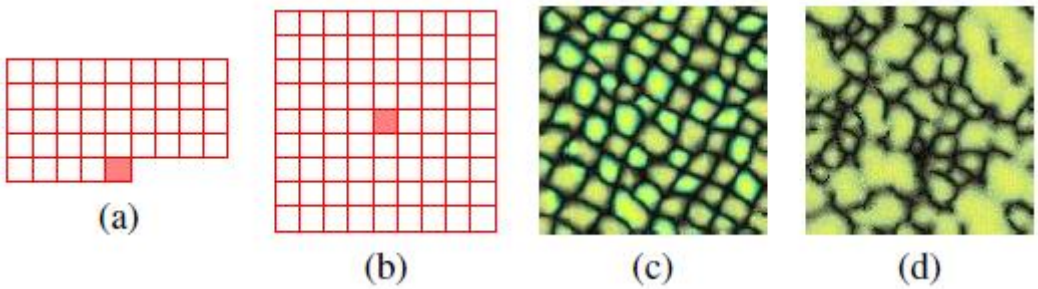


Figura 3.3: Influência da vizinhança na síntese. (a) mostra uma vizinhança que considera apenas pixels já processados. (b) mostra uma vizinhança que não leva em consideração se o pixel já foi sintetizado ou não. (c) mostra a textura gerada com a vizinhança (a). (d) mostra a textura gerada com a vizinhança (b). Ambas as texturas foram inicializadas com o mesmo ruído.

3.2.1. Multi-resolução

O algoritmo de Wey e Levoy usando uma resolução simples captura as características da textura de entrada ajustando o tamanho da região de vizinhança. Entretanto, texturas com componentes estruturais maiores necessitam de regiões de vizinhança maiores o que demanda um alto custo computacional. Esse custo pode ser minimizado com o uso de uma pirâmide de imagem multi-resolução [Burty e Adelson 1983], pois grandes estruturas podem ser capturadas por pequenas regiões de vizinhança em resoluções menores da pirâmide. Em uma pirâmide gaussiana, uma sequência de imagens cada vez menores é criada a partir de uma imagem original. As imagens formadas pelos níveis da pirâmide absorvem as características da imagem original em resoluções menores [Figura 3.4].

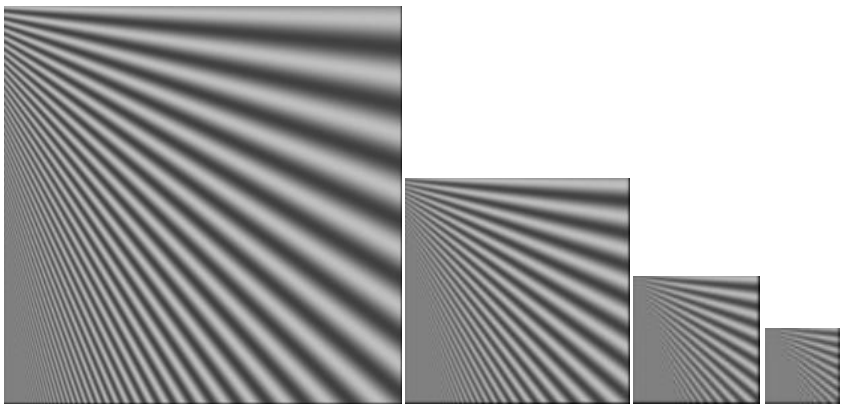


Figura 3.4: Exemplo de uma pirâmide gaussiana de uma imagem.

Utilizando uma estratégia multi-resolução, o algoritmo de Wey e Levoy funciona como descrito a seguir. Inicialmente, duas pirâmides gaussianas são construídas, uma partir da imagem de entrada, e outra a

partir da saída inicializada como um ruído. O algoritmo então altera a pirâmide da imagem de saída a partir dos níveis com menor resolução para os níveis com maior resolução. Quando o algoritmo está em um determinado nível da pirâmide, ele procede como explicado anteriormente, sintetizando pixel a pixel na ordem do *scanline.* A única diferença é na região de vizinhança. No caso multi-resolução, a vizinhança é composta tanto por pixels do nível atual, como por pixels de todos os níveis com resolução menor. A figura 3.5 mostra um exemplo de vizinhança em níveis diferentes da pirâmide.

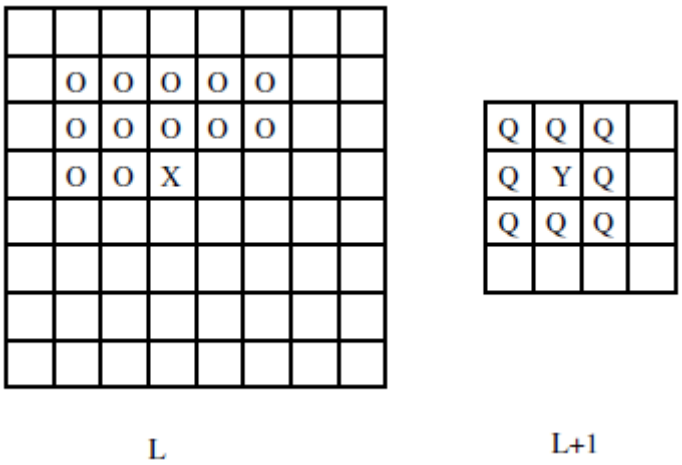


Figura 3.5: Dois níveis de uma pirâmide gaussiana. O nível atual está à esquerda, e o nível com resolução menor à direita. O pixel que está sendo processado está na posição marcada com um X = (L,i,j), onde L é o nível e (i,j) as coordenadas do pixel. O pixel correspondente ao pixel atual no nível L+1 está marcado como Y = (L+1, $\frac{i}{2}$, $\frac{j}{2}$). Desde que o nível com resolução menor já foi completamente sintetizado, a vizinhança do pixel pode conter todos os pixels que estão suficientemente perto dele. No nível atual, a vizinhança deve se limitar a pixels que estão próximos e já foram visitados.

Nós sumarizamos o algoritmo com um pseudo-código do mesmo [Pseudo-código 3.1]. A tabela 3.1 mostra a nomenclatura usada no pseudo-código.

Symbol	Meaning
I_a	Input texture sample
I_s	Output texture image
G_a	Gaussian pyramid built from I_a
G_s	Gaussian pyramid built from I_s
p_i	An input pixel in I_a or G_a
p	An output pixel in I_s or G_s
$N(p)$	Neighborhood around the pixel p
$G(L)$	Lth level of pyramid G
$G(L,x,y)$	Pixel at level L and position (x,y) of G
{R×C,k}	(2D) neighborhood containing k levels, with size R×C at the top level

Tabela 3.1: Tabela de símbolos (Extraída de [Wey e Levoy 2000]).

```
function I_s ← ImageTextureSynthesis(I_a, I_s)
1   InitializeColors(I_s);
2   G_a ← BuildImagePyramid(I_a);
3   G_s ← BuildImagePyramid(I_s);
4   foreach level L from lower to higher resolutions of G_s
5     loop through all pixels p of G_s(L)
6       C ← FindBestMatch(G_a, G_s, L, p);
7       G_s(L, p) ← C;
8   I_s ← ReconImagePyramid(G_s);
9   return I_s;

function C ← FindBestMatch(G_a, G_s, L, p)
10  N_s ← BuildImageNeighborhood(G_s, L, p);
11  N_a^best ← null;    C ← null;
12  loop through all pixels p_i of G_a(L)
13    N_a ← BuildImageNeighborhood(G_a, L, p_i);
14    if Match(N_a, N_s) > Match(N_a^best, N_s)
15      N_a^best ← N_a;    C ← G_a(L, p_i);
16  return C;
```

Pseudo-código 3.1: Algoritmo de Wey e Levoy (Extraído de Wey e Levoy [2000]).

A grande vantagem de usar uma estratégia multi-resolução é poder capturar estruturas grandes da textura de entrada, sem comprometer a performance do algoritmo. Apesar disso, Wey e Levoy sugeriram o uso de uma estrutura de dados para aperfeiçoar ainda mais o desempenho. Posteriormente, em outro trabalho, Wei (2007) propôs outra forma de otimizar o algoritmo. Mostraremos essas melhorias nas próximas seções.

3.2.2. Quantização Vetorial Estruturada em Árvores -TSVQ

Mesmo com a síntese multi-resolução, a busca por regiões similares na textura de entrada ainda é exaustiva. Wei e Levoy (2000) propõem uma otimização através de quantização vetorial estruturada em árvores (Tree Structured Vector Quantization - TSVQ).

Podemos pensar em uma região de vizinhança com *N* pixels, como um ponto N-dimensional. Neste caso, o problema de encontrar a região de entrada mais similar se reduz ao problema de encontrar em um conjunto de pontos em um espaço de dimensão N, o ponto mais próximo a um ponto dado. Em uma TSVQ, constrói-se uma árvore binária que possui cada ponto do conjunto em uma folha. Cada sub-árvore armazena um ponto que tenta representar da melhor forma o conjunto de folhas que a sub-arvore contém.

A busca pela região mais próxima na TSVQ ocorre de forma gulosa, indo sempre ao ramo da árvore cujo valor correspondente àquele ramo mais se aproxima de nosso ponto de consulta. A abordagem é apenas uma aproximação, pois nem sempre a região mais similar é encontrada. Porém os resultados retornados tendem a ser satisfatórios, e devido à grande melhoria na performance do algoritmo, a perda na qualidade da busca pode valer a pena [Figura 3.6]. Uma busca exaustiva pela melhor região levaria tempo $O(NM)$, onde N é o número de pixels na vizinhança e M é o número de pixels na textura de entrada. O uso de TSVQ reduz a complexidade da busca para $O(NlogM)$. Uma desvantagem ao se usar a TSVQ, é a alta quantidade de memória necessária para sua criação. Ao se criar uma árvore com *M* folhas em um espaço com dimensão *N* se consome $O(NM)$ em memória.

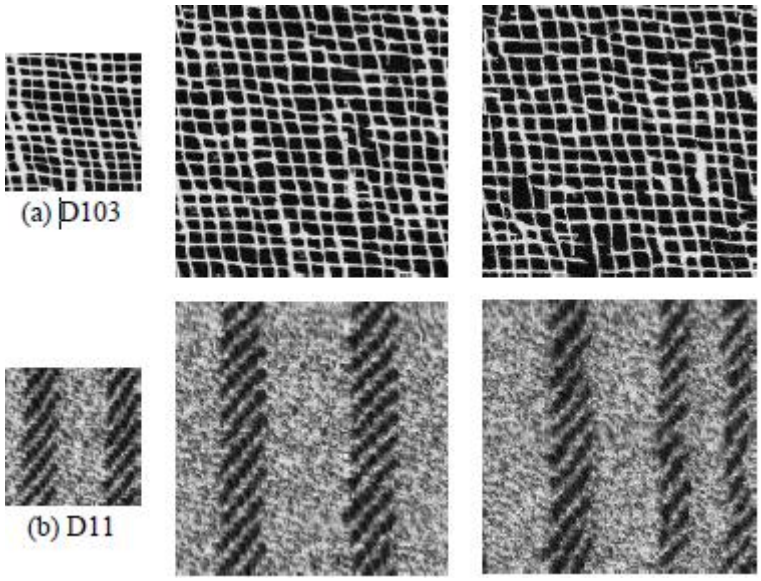


Figura 3.6: Acelerando a síntese usando TSVQ. A textura original é mostrada na coluna da esquerda. Os resultados com a busca exaustiva e com TSVQ são mostrados nas colunas do meio e da direita, respectivamente. O tempo médio para síntese com busca exaustiva é de 360 segundos. A média de tempo para a síntese usando a TSVQ é de 30 segundos [Wei e Levoy 2000].

3.2.3. *K- coerência*

A idéia de usar uma TSVQ para otimizar o processo de busca foi sugerida por Wei e Levoy em seu trabalho original, o mesmo artigo que o algoritmo inteiro foi apresentado [Wei e Levoy 2000]. Mais tarde, Wei et al. (2007) sugerem usar a k-coerência como técnica de otimização, dando a entender que o próprio autor do primeiro trabalho concordou que esta é uma abordagem mais eficiente que a TSVQ. O uso da k-coerência na síntese de texturas foi sugerido por Tong et al. (2002).

A k-coerência consiste de duas etapas básicas, a análise e a síntese. Na análise, o algoritmo constrói para cada pixel da entrada, um conjunto de similaridade. O conjunto de similaridade do pixel *p* contém os

k pixels da entrada que possuem vizinhança mais similar a vizinhança de *p.* K é um parâmetro de entrada fornecido pelo usuário e determina o balanceamento entre eficiência e qualidade. Segundo Wei et al. (2007), um baixo valor de k (entre 2 e 11) funciona bem na prática. A etapa de síntese representa a busca pelo pixel na entrada que será copiado para o pixel atual de saída. Ao invés de testar todos os pixels da entrada, com k-coerência testam-se apenas alguns pixels determinados pelo conjunto de similaridade dos pixels na vizinhança do pixel cujo valor estamos buscando calcular.

Vamos tentar deixar a idéia mais clara com um exemplo [Figura 3.7], supondo que k = 2, nós calculamos o conjunto de similaridade para cada pixel na entrada durante a etapa de análise. Os pixels A e B são os pixels mais similares ao pixel 1. Os conjuntos de similaridades dos outros pixels foram omitidos por simplicidade. Para sintetizar o pixel de saída O, olhamos para os seus vizinhos, e fazemos a união dos conjuntos de similaridades destes, levando em consideração também a posição relativa entre o pixel O e o vizinho correspondente. Os pixels candidatos são os pixels mostrados em azul. A mesma análise de vizinhança que era feita antes com todos os pixels da entrada será feita agora apenas com os pixels mostrados em azul.

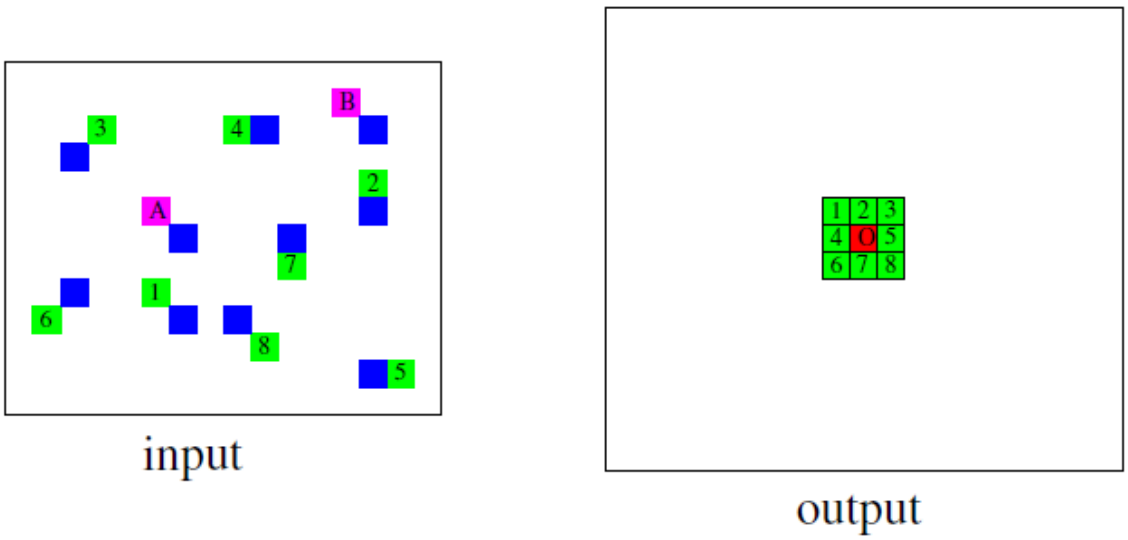


Figura 3.7: Ilustração do algoritmo com K-coerência. O valor final do pixel O será escolhido entre os candidatos mostrados em azul.

3.3. Image Quilting

Os algoritmos apresentados até agora neste trabalho, usavam uma abordagem de gerar a textura de saída pixel a pixel. Efros e Freeman (2001) estavam entre os pioneiros na idéia de copiar pedaços inteiros (*patches*) da amostra de entrada para a saída. Eles nomearam o seu trabalho de *Image Quilting.*

O *Image Quilting* define a unidade de síntese como um bloco quadrado de lado w_b, especificado pelo usuário. Seja S_b o conjunto de todos os blocos quadrados da entrada de lado w_b. Uma idéia inicial para

a síntese seria ir escolhendo novos blocos de S_b randomicamente, e concatenar um ao lado do outro na saída. Como mostrado na figura 3.8-a, o resultado com esta abordagem não é satisfatório, deixando claro que blocos adjacentes não se encaixam. Uma melhoria é mostrada na figura 3.8-b. Se permitimos posicionar os blocos de forma que haja interseção, ao invés de escolhermos o próximo bloco randomicamente, escolhemos aquele que por alguma medida, possui menor erro na região de sobreposição. Apesar da visível melhora, ainda é possível enxergar as descontinuidades nas regiões entre blocos. O truque principal sugerido por Efros e Freeman, foi construir um caminho na região de sobreposição de forma a minimizar a visibilidade no recorte efetuado [Figura 3.8-c].

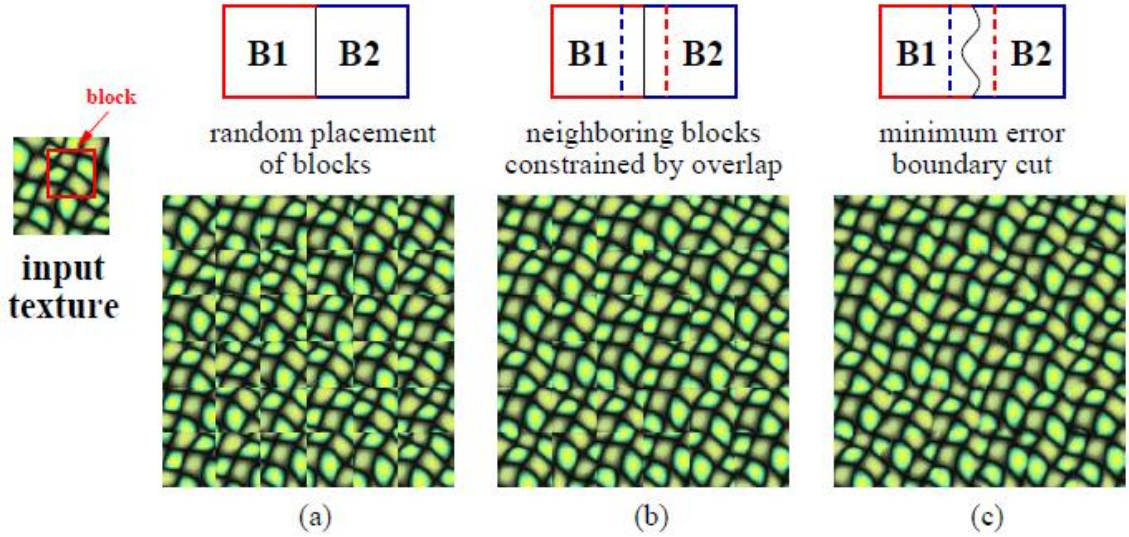


Figura 3.8: Abordagens de colagem dos *patches.* Blocos quadrados da entrada são escolhidos para montar a textura final: (a) blocos são conectados um após o outro randomicamente. (b) os blocos escolhidos são aqueles que de acordo com alguma medida, possuem baixo erro na região de sobreposição. (c) Um caminho de custo mínimo entre os blocos sobrepostos é calculado.

Basicamente, podemos descrever o algoritmo através dos seguintes passos:

- Varrer a imagem de saída na ordem do *scanline* (o incremento deve ser o tamanho do bloco w_b menos um valor w_e que indicará o tamanho da região de sobreposição, que deverá ser controlado pelo usuário).

- A cada passo, escolhe-se aleatoriamente um bloco dentre aqueles que possuem um erro aceitável na região de sobreposição. É considerado aceitável aquele erro que for no máximo 10% maior do que o menor erro de algum bloco. Para o cálculo do erro, podemos novamente usar a soma dos quadrados das diferenças entre os pixels que ocupam o mesmo lugar na região de sobreposição.

- Encontra-se um caminho de menor custo para se fazer o recorte do bloco atual.

Para encontrar o recorte de custo mínimo entre os blocos, o *Image Quilting* utiliza uma abordagem de programação dinâmica. Primeiramente deve-se construir a matriz de erro E. Sejam B_1 e B_2 os blocos que se sobrepõem ao longo da aresta vertical, com regiões de sobreposição B_1^{ov} e B_2^{ov} [Figura 3.8-b]. Se w_b é o tamanho do bloco e w_e é o parâmetro de sobreposição, então a área total da região de sobreposição será $w_e * w_b$. Então definimos a superfície do erro na região de superposição como o quadrado da diferença dos pixels em cada bloco, $E = (B_1^{ov} - B_2^{ov})^2$. Após essa etapa, usa-se programação dinâmica para encontrar um caminho que corte a região de sobreposição verticalmente. A matriz da programação dinâmica é calculada da seguinte forma:

$$PD_{i,j} = E_{i,j} + \min(PD_{i-1,j-1}, PD_{i-1,j}, PD_{i-1,j+1})$$

No fim do procedimento, a posição com menor valor da última linha representará o final do melhor recorte. Para recuperar o caminho, basta voltar na matriz da programação dinâmica até alcançar a primeira linha. Um procedimento similar pode ser aplicado ao caso de sobreposição horizontal. Uma vez que o algoritmo insere os blocos na ordem do *scanline*, os pixels que ficarem a direita do corte vertical e abaixo do corte horizontal receberão o novo bloco, enquanto os outros pixels permanecerão com os valores antigos.

A figura 3.9 mostra algumas texturas geradas com o *Image Quilting.*

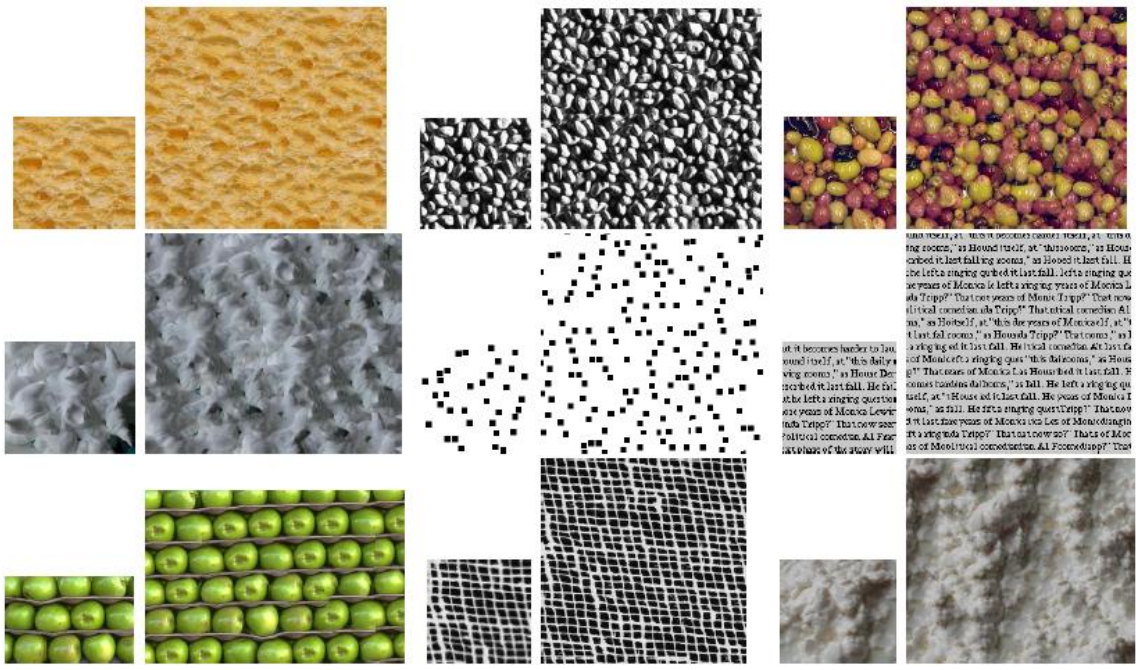


Figura 3.9: Exemplos de texturas sintetizadas com o *Image Quilting* (Figura extraída de Efros e Freeman [2001]).

4. Síntese de Texturas com cortes em grafos

Uma vez que foi implementado o algoritmo proposto por Kwatra et al. (2005), este trabalho receberá um capítulo separado para ser explicado com mais detalhes. Uma vez que o algoritmo se baseia em cortes em grafos, primeiramente introduziremos alguns conceitos e idéias sobre este tema, para posteriormente mostrar o algoritmo de síntese propriamente dito.

4.1. Cortes em grafos e fluxo em redes

Fluxo em redes é um campo da otimização combinatória que foi e ainda é extensivamente estudado, devido a seu vasto número de aplicações em problemas de otimização. Intimamente ligado ao problema de fluxo redes, está o estudo de cortes em grafos. Ford e Fulkerson (1956) provaram que encontrar o fluxo máximo entre dois nós em uma rede é equivalente a determinar o corte mínimo que separa estes dois nós no grafo. Definiremos formalmente estes problemas e descreveremos algumas soluções mais populares.

Seja $G(V, E)$ um grafo no qual cada aresta $(u, v) \in E$ possui um valor não negativo associado $c(u, v) \geq 0$, chamamos este valor de capacidade da aresta. Dois vértices especiais s e t são designados como *source* e *sink* respectivamente. Um fluxo em *G* de s para t é definido como uma função $f: E \rightarrow R$ que satisfaz as seguintes propriedades:

- Restrição de capacidade:

$$\forall\ (u,\ v) \in E,\ 0 \leq f(u, v) \leq c(u, v)$$

- Lei da conservação de fluxo:

$$\forall\ u\ \in V - \{s, t\},\ \sum_{v \in V} f(u, v) = \sum_{v \in V} f(v, u)$$

A primeira restrição nos mostra que o fluxo que passa por uma aresta nunca pode ser maior que sua capacidade, enquanto a lei da conservação de fluxo diz que todo o fluxo que entra em nós intermediários (um nó que não seja *source* ou *sink*) deverá sair do mesmo.

Podemos usar os termos das variáveis do problema para nos ajudar a ter uma interpretação do mesmo, imaginando que precisamos enviar certa quantidade de material de um nó origem (*source)* a um nó destino (*sink).* A quantidade a ser transportada diretamente entre dois nós é limitada por sua capacidade e também nenhum material deve ser desperdiçado em algum nó que não seja o destino.

Nós definimos o valor de um fluxo como $|\mathrm{f}| = \sum_{\mathrm{v} \in \mathrm{V}} \mathrm{f}(s, \mathrm{v})$, ou seja, o valor do fluxo é determinado pela quantidade total de fluxo que deixa o *source*. Devido à lei de conservação do fluxo, a quantidade de fluxo que sai do *source* é igual ao total de fluxo chegando ao sink*.* A Figura 4.1 mostra um exemplo de um fluxo em uma rede.

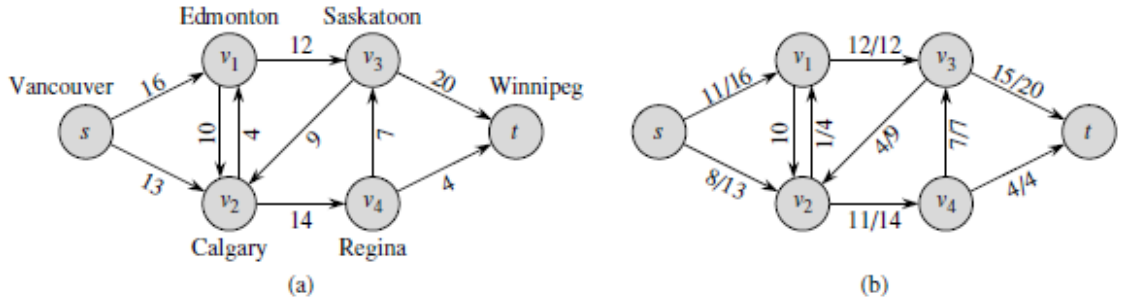


Figura 4.1: Exemplos de fluxo em redes. (a) Um grafo $G(V, E)$ com capacidades nas arestas. (b) Um fluxo f em G. Perceba que o valor do fluxo em cada aresta (primeiro número) é menor que a capacidade da mesma (segundo número). Para todos os nós do grafo, exceto o *source* e o *sink*, o fluxo total de entrada é igual ao fluxo total de saída. O valor do fluxo é $|f| = 19$. (Figura extraída de Cormen et al. [1990])

No mesmo contexto de um grafo com arestas ponderadas, definimos um corte como uma partição dos vértices do grafo em dois conjuntos *S* e *T*. O *peso* de um corte é definido como a soma dos pesos das arestas que têm um vértice em *S* e outro em *T* (em casos de grafos direcionados, contamos apenas as arestas que iniciam em *S* e terminam em *T*) [Figura 4.2]. Associando o peso das arestas às capacidades citadas anteriormente, podemos falar da capacidade de um corte *S*-*T* como $c(S,T) = \sum_{u \in S} \sum_{v \in T} c(u,v)$. Encontrar um corte mínimo em um grafo significa encontrar um corte cuja capacidade tem o menor valor possível dentre todos os cortes neste grafo. O teorema abaixo é o resultado mais importante no estudo de fluxo em redes, e foi demonstrado por Ford e Fulkerson (1956).

Teorema 4.1: **Teorema do fluxo-máximo corte-mínimo**

Em um grafo $G(V, E)$ com source s e sink t, o valor do fluxo máximo entre s e t é igual ao valor do corte mínimo $S - T$, tal que $s \in S, t \in T$.

Uma prova formal deste teorema pode ser encontrada em diversos livros de algoritmos [Cormen et al. 1990, Jungnickel 2004, Kleinberg e Tardos 2005], aqui forneceremos apenas uma idéia intuitiva da prova.

Imagine que quando você define um corte que separa o *source* e o *sink*, todo o fluxo transmitido entre eles deverá passar pelas arestas do corte. Portanto é impossível enviar mais fluxo do que algum corte permite. Isso nos faz perceber que todo fluxo é menor ou igual a qualquer corte. O restante da prova é mostrar que quando se alcança o fluxo máximo, este define um corte onde todas as arestas do mesmo estão saturadas (não permitem mais passagem de fluxo), o que faz perceber que existe um fluxo que possui o mesmo valor de um corte. Como *todo* fluxo é limitado superiormente por *qualquer* corte, então temos um fluxo que é máximo e um corte mínimo.

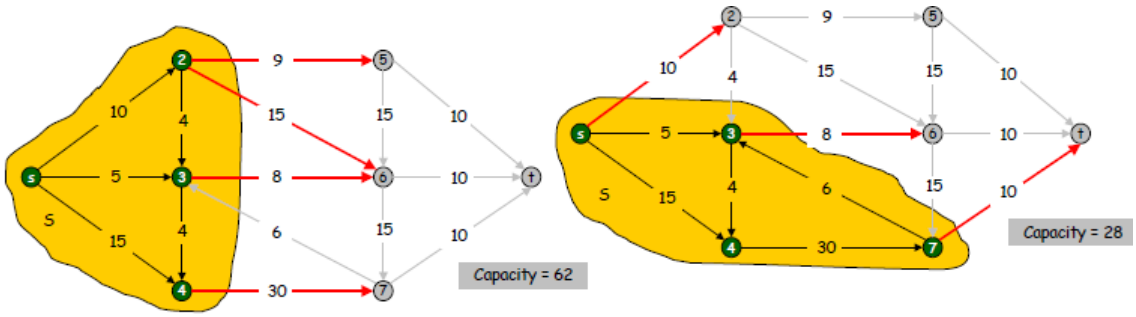


Figura 4.2: Exemplos de cortes em um grafo. (a) mostra um corte de capacidade 62, enquanto (b) mostra um corte melhor, com capacidade 28. Observe que apenas arestas que vão do conjunto do source para o conjunto do sink são contabilizadas (s e t representam o source e o sink, respectivamente).

Além da demonstração do teorema, Ford e Fulkerson criaram um método geral para se encontrar o fluxo máximo / corte mínimo em um grafo. A idéia é incrementar o fluxo do *source* para o *sink* através de *caminhos de aumento.* Estes são caminhos definidos de forma que seja possível passar mais fluxo por eles sem desobedecer às restrições da rede. Também pela definição de caminho de aumento, caso não existam mais caminhos deste tipo, então já encontramos o fluxo máximo.

O método de Ford Fulkerson é genérico, pois ele não explicita qual tipo de busca usar para encontrar os caminhos de aumento. O uso de uma busca inapropriada pode resultar em uma complexidade exponencial no pior caso. Apenas mais de dez anos depois, houve o surgimento de algoritmos polinomiais baseados nessa abordagem.

Edmonds e Karp (1972) provaram que ao se usar uma busca em largura para encontrar os caminhos de aumento, alcança-se uma complexidade assintótica de $O(VE^2)$. Uma abordagem ainda mais eficiente para o problema foi proposta por Dinic (1970). Em seu trabalho, Dinic sugere encontrar um conjunto maximal de caminhos de aumento disjuntos em cada iteração, ao invés de apenas um. Essa otimização resulta em um algoritmo $O(V^2E)$.

Uma nova classe de algoritmos de fluxo em rede surgiu na década de 80 [Tarjan 1984, Goldberg e Tarjan 1986]. Este método, conhecido como *push-relabel,* trabalha de uma forma mais localizada que o de Ford e Fulkerson. Ao invés de olhar para toda a rede a procura de caminhos de aumento, algoritmos de *push-relabel* avaliam um vértice por vez, olhando apenas seus adjacentes na rede. Outra diferença importante, é que a lei da conservação de fluxo não é preservada durante toda execução deste tipo de algoritmo, apenas no final. Um genérico método de *push-relabel* tem complexidade de $O(VE^2)$ no pior caso, entretanto, existem algoritmos desta classe que rodam em tempo $O(V^3)$ [Goldberg e Tarjan 1986].

4.2. Síntese com cortes em grafos

Assim como o *Image Quilting*, o algoritmo proposto por Kwatra et al. (2003) é de síntese de texturas baseada em *patches*. Logo o seu procedimento básico é copiar pedaços da textura de entrada para a textura de saída. A cada iteração, procura-se por um deslocamento coerente do *patch* atual na saída. Ao invés de programação dinâmica, para encontrar o "recorte ótimo" entre o novo *patch* e a textura já sintetizada, se reduz o problema ao de calcular cortes mínimos em grafos.

A abordagem com programação dinâmica assume uma estrutura de grid na região de sobreposição o que restringe os possíveis recortes e impede uma extensão do algoritmo para outras dimensões. A estratégia de recorte com cortes em grafos funciona tanto com imagens bidimensionais como com texturas 3D (vídeos com propriedades locais e estacionárias).

4.2.1. Recorte do novo *patch*

Como mencionado anteriormente, o processo de cópia dos *patches* no algoritmo de Kwatra et al. pode ser dividido em dois estágios. Primeiramente, um posicionamento do *patch* candidato é selecionado por comparar o mesmo com a imagem de saída parcialmente sintetizada. Posteriormente, um algoritmo de cortes em grafos é aplicado para encontrar o melhor recorte deste *patch* a ser copiado para a saída.

O processo de posicionamento do *patch* será descrito na próxima seção. Agora, mostraremos o coração da técnica, que é encontrar o recorte ótimo a partir do *patch* já posicionado. Assim como no *Image Quilting,* a dificuldade é encontrar um caminho que recorte a região de sobreposição entre o novo *patch* e a imagem de saída. Os pixels do novo *patch* que não estiverem já ocupados por antigos *patches* serão automaticamente copiados para a textura final.

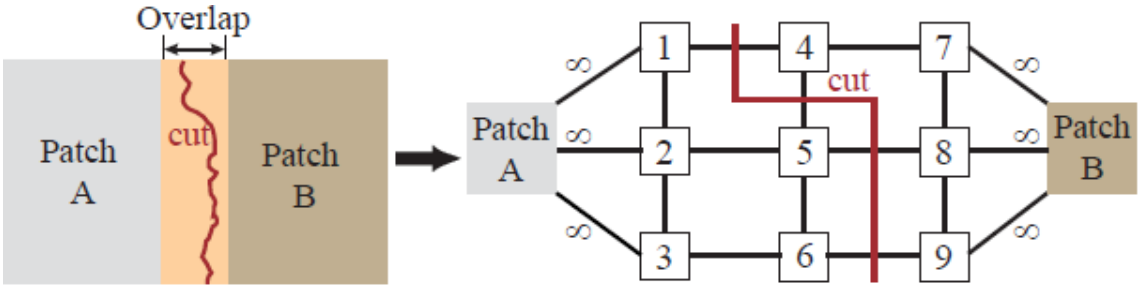
Primeiro, é necessário uma forma de mensurar a qualidade de um recorte. Nesta técnica, o recorte será um caminho *entre* pares de pixels adjacentes (No *Image Quilting* o caminho ocorre sobre os pixels). Considere *s* e *t* dois pixels adjacentes na imagem de entrada. Faça também ***A*** e ***B*** serem o antigo e novo *patch*, respectivmante. Definimos o custo de um recorte entre os pixels nas posições *s* e *t* como:

$$M(s, t, \boldsymbol{A}, \boldsymbol{B}) = |\boldsymbol{A}(s) - \boldsymbol{B}(s)| - |\boldsymbol{A}(t) - \boldsymbol{B}(t)|$$

| X | representa uma norma apropriada (a soma dos quadrados das coordenadas é a mais comum).

A Figura 4.3 ilustra o processo de construção do grafo. Cada pixel na região de sobreposição terá um nó do grafo para representá-lo. Cada nó se ligará aos nós que representam seus pixels adjacentes. O peso de uma aresta indica o custo do recorte passar entre os pixels representados pelos nós terminais da mesma. O corte no grafo irá

particionar os nós em dois conjuntos, os nós que ficarem na mesma partição do *source* continuarão com os pixels inalterados, enquanto a partição do *sink* determinará os pixels na região de sobreposição que receberão o novo *patch.* Os pixels que fazem fronteira com regiões já sintetizadas da imagem que não estão na área de sobreposição devem permanecer com seus antigos valores para que não haja uma descontinuidade. Portanto adiciona-se uma aresta de custo infinito a partir do *source* (ilustrado na figura como “Patch A”), para garantir que aquele nó e o *source* estarão na mesma partição. Analogamente, os pixels que fazem fronteira com regiões ainda não sintetizadas da imagem e que serão agora ocupadas pelo novo *patch* devem agora obrigatoriamente estar do mesmo lado do *sink* (“Patch B”) no corte. Um corte neste grafo representa exatamente a separação entre os pixels que serão alterados ou não. Como o peso da aresta representa o grau de descontinuidade de um corte que passa por ela, devemos calcular um *corte mínimo* neste grafo. Qualquer dos algoritmos que resolvam o problema do fluxo máximo / corte mínimo citado na seção anterior podem ser utilizados nesta etapa.



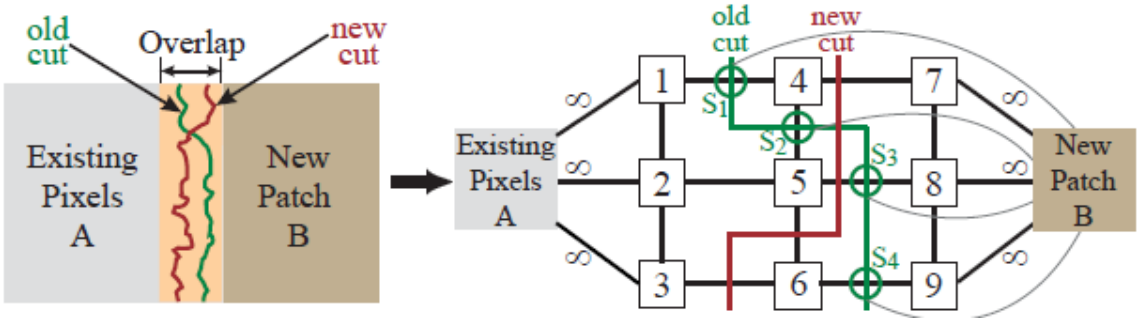
(a) (b)

Figura 4.3: Encontrando o corte. (a) Exemplo de corte entre patches. (b) Formulação do grafo necessário para encontrar o recorte ótimo na região de sobreposição. Pixels 4,7,8 e 9 receberão o *patch* B.

A formulação do grafo explanada até agora é suficiente para encontrar recortes e pode ser utilizada para substituir a técnica de programação dinâmica do Image Quilting. Porém a estrutura construída ainda não é suficiente para explorar o poder de usar cortes em grafos na síntese de texturas. Suponha que uma série de patches já foi copiada para a textura de saída. Existe um potencial para que “costuras” entre patches distintos gerem descontinuidades visíveis. Isto pode ser mensurado através do custo das arestas quando construímos o grafo.

A nova formulação é demonstrada na Figura 4.4. Para cada dois pixels adjacentes que pelos quais foi criada uma costura cria-se um nó adicional, denominado “nó de costura”, e posicionamos este nó entre os nós que correspondem aos pixels envolvidos. Chamemos de *s* e *t* as posições dos pixels envolvidos, chamemos de $\boldsymbol{B}$ o novo *patch* e de $\boldsymbol{A_s}$ e $\boldsymbol{A_t}$ os *patches* envolvidos na costura (A posição *s* recebeu o *patch* $\boldsymbol{A_s}$ e a

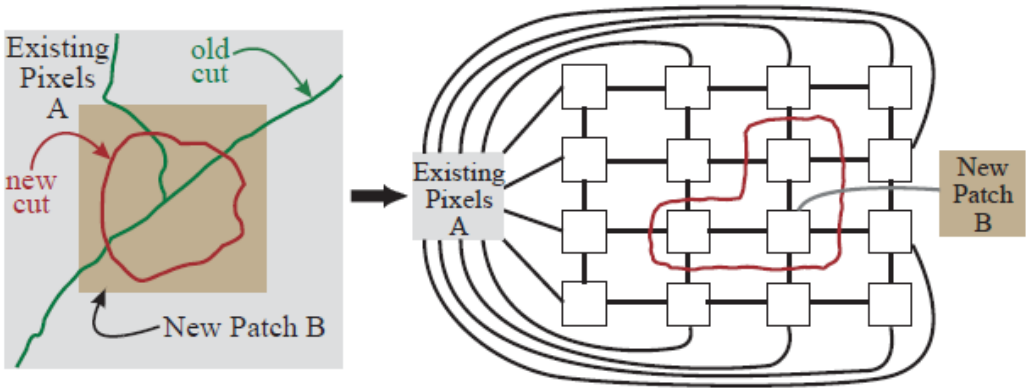
posição *t recebeu* $\boldsymbol{A_t}$). Cria-se uma aresta entre o nó de costura e o *sink,* cujo peso será o custo do recorte antigo $M(s,t,\boldsymbol{A_s},\boldsymbol{A_t})$ **.** Na figura 4.4, o nó s_1 representa a costura antiga entre as posições 1 e 4. Cria-se uma aresta de s_1 para o *sink* com custo $M(1,4,A_1,A_4)$. Criam-se também arestas para representarem a substituição dessa antiga costura por uma nova. A aresta entre s_1 e o nó 1 possui peso $M(1,4,A_1,B)$, e é o custo de associar apenas o pixel na posição 4 ao novo *patch.* Analogamente, a aresta entre s_1 e o nó 4 possui peso $M(1,4,B,A_4)$ e representa o caso em que apenas o nó 1 recebe o novo *patch.* Em suma, a aresta do nó de costura que cruzar o corte determinará a nova configuração dos pixels envolvidos. Se a aresta que liga o nó de costura ao *sink* cruzar o corte, significa que você está pagando pela costura antiga e ela permanecerá lá. Se a aresta que liga ao nó 1 cruzar o corte, significa que apenas o pixel 4 será associado ao novo *patch.* O caso em que a aresta para o nó 4 cruza o corte é análogo. Perceba que para essa formulação funcionar, o nó de costura deve ter no máximo uma aresta cruzando o corte, isso acontece se a métrica utilizada no custo obedecer a desigualdade triangular (o custo de escolher duas arestas é sempre pior que escolher apenas uma) [Boykov et al. 1999].



(a) (b)

Figura 4.4: Nós de costura. (a) Em vermelho está o novo corte, e em verde o corte antigo. (b) Formulação do grafo levando em consideração antigos recortes. Nós de s1 a s4 e seus arcos representam o custo da costura antiga.

Até agora mostramos apenas casos nos quais o novo *patch* é copiado para uma região de borda com os antigos *patches*, pois inicialmente temos o propósito de estender a imagem de saída adicionando o *patch* a posições ainda não ocupadas. Porém pela forma que o problema foi estruturado, nada impede de posteriormente colarmos *patches* em regiões da saída que já estejam totalmente preenchidas, no intuito de melhorar possíveis costuras que ficaram visíveis. Um exemplo dessa estratégia é ilustrado na figura 4.5. A formulação do grafo é a mesma que explicada no parágrafo anterior. Nós da borda são conectados ao *source* para evitarem descontinuidades na fronteira, e os nós de costura já se encarregam de conectarem o grafo ao *sink*. Na figura 4.5-a podemos ver que com essa abordagem, formas totalmente irregulares (como ciclos) podem ser escolhidas como o recorte.



(a) (b)

Figura 4.5: Tratando regiões já preenchidas. (a) posicionamento de um patch em uma região totalmente preenchida. Antigas costuras (verde) são substituídas por uma nova (vermelho). (b) Estrutura do grafo. Pixels da borda são forçados a continuarem com seus antigos valores. Nós de costura e suas arestas foram omitidos por claridade.

4.2.2. Posicionamento dos Patches

Kwatra et al. sugerem três possíveis abordagens para o posicionamento dos *patches.* Estes métodos são: (1) posicionamento aleatório, (2) casamento do *patch* inteiro e (3) casamento de *sub-patch*

Para todos os três casos, na primeira parte do algoritmo o posicionamento é feito de forma que garanta que a textura de saída seja estendida com mais pixels preenchidos, até que a mesma seja totalmente preenchida. Posteriormente o posicionamento tem apenas a função de melhorar a síntese já efetuada. Nesta etapa, nos dois últimos métodos, usa-se o custo das costuras antigas para se procurar por uma região na saída que possui o maior potencial para descontinuidades. Após isso, busca-se apenas por posicionamentos do *patch* que cubram inteiramente a região com erro.

Algo importante de mencionar, é que ao contrário do *Image Quilting* cujo *patch* possui o tamanho de um bloco especificado pelo usuário, Kwatra et al. usam a entrada inteira como o bloco, e o posicionamento consiste apenas de encontrar um deslocamento relativo da entrada na imagem de saída.

Agora descreveremos cada um dos métodos de posicionamento sugeridos:

(1) Posicionamento aleatório: Nesta abordagem, o novo *patch* (a imagem de entrada) é randomicamente posicionado em alguma posição da saída. Este método é sem dúvida o mais rápido, porém só funciona bem em texturas estocásticas, que não apresentam um componente estrutural bem definido.

(2) Casamento do *patch* inteiro: Este método busca por posicionamentos da imagem de entrada que casam bem com a saída parcialmente sintetizada. Para considerar o caso em que apenas uma parcela da imagem de entrada intersecta com a

saída devemos normalizar o custo (soma dos quadrados das diferenças) pela área de sobreposição. A função abaixo deve ser calculada para todas as possíveis translações *t* da entrada:

$$C(t) = \frac{1}{|A_t|} \sum_{p \in A_t} |\boldsymbol{I}(p) - \boldsymbol{O}(p+t)|^2$$

Onde ***I*** e ***O*** são as imagens de entrada e saída, respectivamente, e A_t é a região de sobreposição. O novo *patch* é escolhido randomicamente de acordo com a seguinte função de probabilidade:

$$p(t) \propto e^{-\frac{c(t)}{k\alpha^2}}$$

Onde α é o desvio padrão dos pixels na imagem de entrada, e k é um parâmetro de entrada que indica o quão estocástica será a síntese. Este método é o melhor para texturas estruturadas e semi-estruturadas.

(3) Casamento de *sub-patch:* Aqui, primeiramente um bloco da saída é escolhido (um *sub-patch* ligeiramente menor que a textura de entrada). Então, busca-se por translações da entrada tal que o erro na região de sobreposição com o bloco escolhido seja mínimo.

$$C(t) = \sum_{p \in S_0} |\boldsymbol{I}(p) - \boldsymbol{O}(p+t)|^2$$

A translação é então escolhida estocasticamente, de maneira similar ao método anterior. Este é o melhor método para texturas estocásticas e síntese de vídeos.

4.2.3. Aceleração com FFT

Os métodos de busca por um posicionamento do *patch* descritos podem ser muito custosos se forem implementados de uma maneira trivial. Computar a função de custo *C(t)* para todas as válidas translações da entrada consome tempo $O(n^2)$ onde n é o tamanho da entrada. Entretanto a função de custo pode ser re-escrita como segue:

$$C(t) = \sum_p \boldsymbol{I}(p) + \sum_p \boldsymbol{O}(p+t) - 2 * \sum_p \boldsymbol{I}(p) * \boldsymbol{O}(p+t)$$

Os dois primeiros somatórios são consultas da soma em um sub-retângulo da entrada ou da saída, e podem ser respondidas eficientemente com uma matriz de somas acumuladas. O terceiro somatório é uma *convolução* da entrada com a saída e pode ser computado eficientemente usando a *Transformada rápida de Fourier* (FFT) [Kilthau et al. 2002]. O custo total da computação com estas otimizações resulta em $O(nlogn)$ no total. Para imagens grandes, o tempo de síntese cai de poucos minutos para poucos segundos com esta aceleração.

5. Resultados

Neste trabalho sobre síntese de texturas, além deste documento, uma instância do algoritmo de Kwatra et al. (2003) foi implementado. O código foi escrito em C++, e para a manipulação de imagens com extensões variadas foi utilizada a biblioteca CImg (2010). A biblioteca foi usada no intuito de abstrair a codificação ou decodificação da imagem (em nosso caso, um conjunto de pixels *rgb*) em algum formato conhecido (JPEG, bitmap, GIF, etc).

Como sugerido no trabalho original, três técnicas de posicionamento de *patches* foram implementadas e comparadas. Testamos também dois algoritmos para o problema de corte mínimo, a saber, o algoritmo de Edmonds e Karp [1972] e o Dinic [1970].

A seguir, mostramos uma série de imagens geradas pelo algoritmo, separando os resultados pela técnica de posicionamento utilizada.

5.1. Posicionamento aleatório

Os resultados para esta técnica confirmaram o que já era esperado. Ela é definitivamente a abordagem mais rápida para o posicionamento de *patches* e funciona bem para texturas estocásticas, aquelas que não apresentam uma estrutura regular e estão mais próximas de um sinal de ruído. As figuras 5.1 e .5.2 mostram exemplos de imagens geradas posicionando a textura de entrada aleatoriamente na saída e deixando todo o trabalho para o algoritmo de recorte.

Figura 5.1: Síntese de flocos de arroz usando posicionamento aleatório. A esquerda está a textura dada como amostra na entrada. A direita, a imagem resultante do algoritmo.

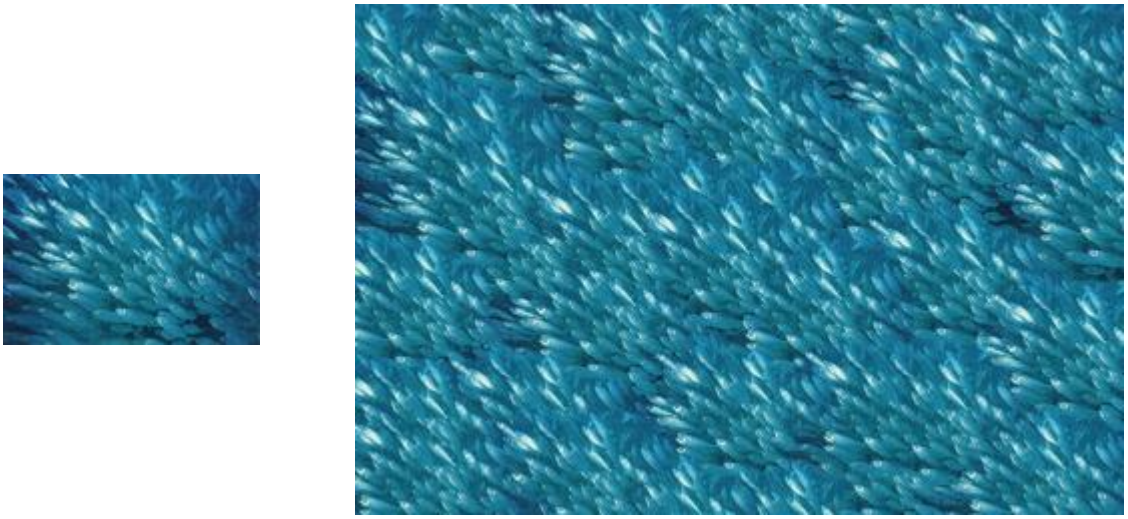


Figura 5.2: Síntese de um cardume de peixes usando posicionamento aleatório.

Entretanto, a simplicidade desta técnica de posicionamento peca quando tenta sintetizar texturas que possuem padrões mais regulares e definidos. A baixa qualidade dos resultados torna proibitivo o uso desta abordagem para geração de texturas que não sejam estocásticas. Na figura 5.3 pode-se ver a fraqueza do posicionamento aleatório na síntese de texturas regulares. Nítidas regiões de descontinuidades permanecem visíveis na textura de saída.

Vale salientar a importância de utilizar este método na síntese de texturas estocásticas devido a sua performance ser bastante superior que os demais.

Figura 5.3: Uso do posicionamento aleatório em uma textura de entrada regular, gerando um resultado não satisfatório.

5.2. Casamento com patch inteiro

Como explicado no capítulo anterior, o *casamento com patch inteiro* é muito mais sofisticado e geral que o posicionamento aleatório. Uma vez que procuramos por translações de entrada que casem bem com o estado atual da saída, este método consegue produzir ótimos resultados para praticamente todo o espectro de texturas.

A figura 5.4 mostra exemplos de síntese de texturas regulares. Ao contrário do que acontecia quando tentávamos transladar a imagem de entrada para uma posição aleatória da saída, quando se tentar casar a entrada em uma região similar, até mesmo texturas em que os padrões se repetem de uma forma bastante rígida podem ser sintetizadas, produzindo bons resultados.

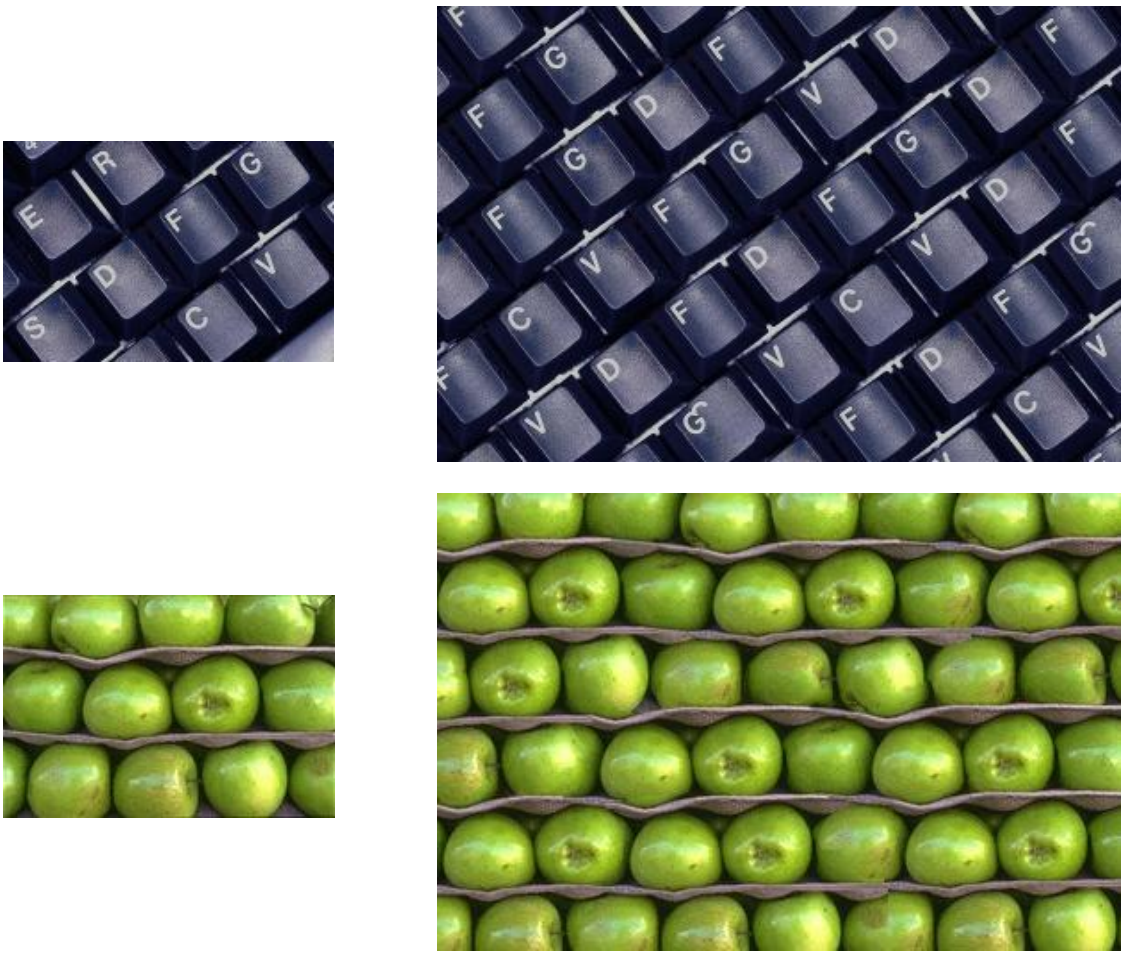
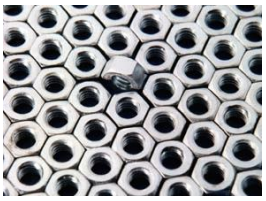


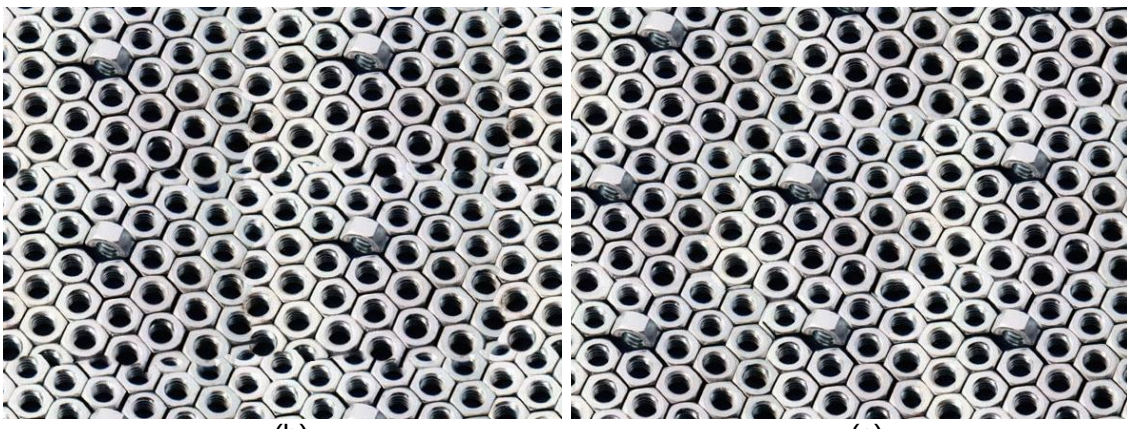
Figura 5.4: Síntese de texturas regulares utilizando *casamento de patch inteiro*.

Em nossa implementação, definimos um parâmetro de entrada chamado *overlap* que não é citado no trabalho original. Este parâmetro indica o tamanho da região de sobreposição máximo permitido durante a etapa de inicialização da textura (etapa na qual a textura ainda não está totalmente preenchida). Em texturas regulares o *overlap* necessita ser maior para conseguir capturar o padrão da imagem. Porém, um alto valor de *overlap* resulta em um tempo de síntese maior.

Para ilustrar como a escolha do parâmetro *overlap* influencia no resultado final, mostramos na figura 5.5 duas texturas resultantes da mesma amostra de entrada, porém com diferentes valores de *overlap*. Em uma delas, regiões de costura entre *patches* são facilmente identificáveis, enquanto na outra não. Porém, melhorar a suavidade entre os recortes não é a único fator a ser considerado na escolha do *overlap*. A figura 5.6 mostra um caso em que apesar de gerar uma imagem sem descontinuidades visíveis, um valor insuficiente de *overlap* não consegue capturar as propriedades estruturais da textura por completo.Este exemplo é excelente para percebermos a importância das duas fases do processo de síntese (posicionamento do *patch* e recorte ótimo na região de sobreposição).Na etapa de calcular o corte mínimo, pela forma que os pesos das arestas no grafo são modelados, o corte tenta apenas minimizar a descontinuidade entre pixels adjacentes que ficarem em *patches* diferentes. Logo, o algoritmo tenta maximizar a suavidade que acontecerá a variação entre o novo *patch* e a imagem anterior, sem se preocupar em preservar a estrutura da textura de entrada. A tentativa de manter a regularidade estrutural da textura é responsabilidade do algoritmo de posicionamento de *patches*, que busca encontrar uma região na saída que seja similar a **toda** a textura de entrada. Por este motivo, valores de *overlap* inapropriados podem levar a perda de características estruturais em texturas regulares.



(a)



(b) (c)

Figura 5.5: Influência do parâmetro *overlap* na qualidade do resultado. (a) mostra a textura de entrada do algoritmo que gera (b) e (c). (b) é uma textura com várias regiões com recortes visíveis, gerada com um *overlap* igual a 30. Em (c), que foi gerada com o *overlap* configurado como 100, a qualidade da síntese é bem maior, tornando muito difícil a percepção de descontinuidades nos recortes.

Overlap = 100

Overlap = 150

Figura 5.6: Influência do parâmetro *overlap* na preservação da estrutura. Ambas imagens não possuem descontinuidades visíveis, porém uma delas não absorveu corretamente a estrutura da amostra (ao menos que o contexto seja um livro de fantasia, cavalos de 6 patas não devem ser bem vindos).

Como mencionado, esta estratégia de posicionamento também gera bons resultados para texturas estocásticas, com o bônus de que neste tipo de textura, um baixo valor de *overlap* é suficiente. Na figura 5.7 vemos um exemplo de uma textura estocástica sintetizada com um pequeno *overlap* mas mesmo assim com uma boa qualidade.

Figura 5.7: Síntese de folhagem com *overlap* igual a 50.

No gráfico 5.1 (abaixo) vemos um pequeno comparativo entre os tempos de execução de uma mesma textura de entrada (a saber, a entrada utilizada foi a textura dos cavalos da figura 5.6) em função da variação do *overlap*. Percebe-se o crescimento não linear do tempo de execução do algoritmo em função do *overlap*. De fato, a complexidade do algoritmo de posicionamento depende quadraticamente do *overlap* máximo.

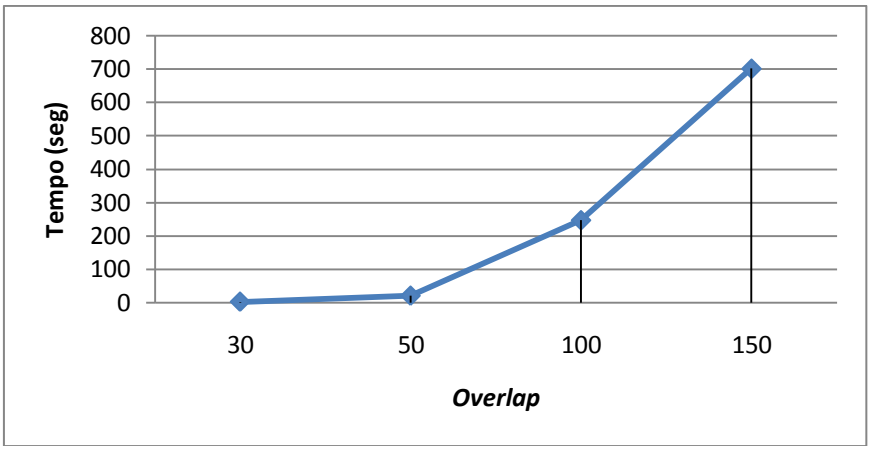


Gráfico 5.1: Relação de dependência quadrática do tempo de execução do algoritmo em função do parâmetro de entrada *overlap*. No gráfico, o tempo é medido em segundos.

5.3. Casamento de sub-patch

Assim como o anterior, este método busca por translações da entrada que sejam similares a saída parcialmente sintetizada, para posteriormente se encontrar o corte mínimo na região de sobreposição. A única diferença entre o *casamento de sub-patch* e o *casamento de patch inteiro* é que no primeiro, um bloco (sub-patch) da saída é escolhido de antemão, e a região de sobreposição considerada no cálculo é a interseção da entrada transladada com este bloco da saída, enquanto no segundo toda a imagem de saída já sintetizada pode entrar no processo.

A vantagem do *casamento de sub-patch* surge quando um tipo diferente de "textura" aparece como entrada do algoritmo. De fato, talvez não possa ser considerado como uma textura de acordo com o modelo de campos aleatórios de Markov, mas podemos chamá-las como texturas *semi-estacionárias*. Estas são imagens que não apresentam similaridades por toda a sua extensão, porém pode ser considerada como estacionária em partes. Um exemplo é ilustrado na figura 5.8.

O processo de síntese para imagens deste tipo ainda não está bem consolidado com o uso do *casamento de sub-patch*. Em nossa implementação alguns parâmetros precisam ser ajustados dependendo da imagem de entrada e as dimensões da imagem de saída geralmente se restringem a alguns valores. Provavelmente, a síntese a partir de amostras deste tipo de imagem ainda precisa ser mais estudada para a

criação de técnicas que dependam menos do usuário. Em nosso caso, nos restringimos a sintetizar duas delas, como mostrado na figura 5.8.

(a)

(b)

Figura 5.8: Dois exemplos de síntese com imagens que podem ser consideradas do que chamamos “semi-estacionárias”. Na exemplo de cima (a), percebe-se a fragilidade do método pela quantidade excessiva de repetição.

Além dessa nova possibilidade com imagens que se assemelham a texturas em algumas regiões, o método de posicionamento por casamento de sub-patch também produziu bons resultados com texturas estocásticas e regulares, mostrando então ser o mais geral dos métodos de posicionamento de *patch* [Figura 5.9].

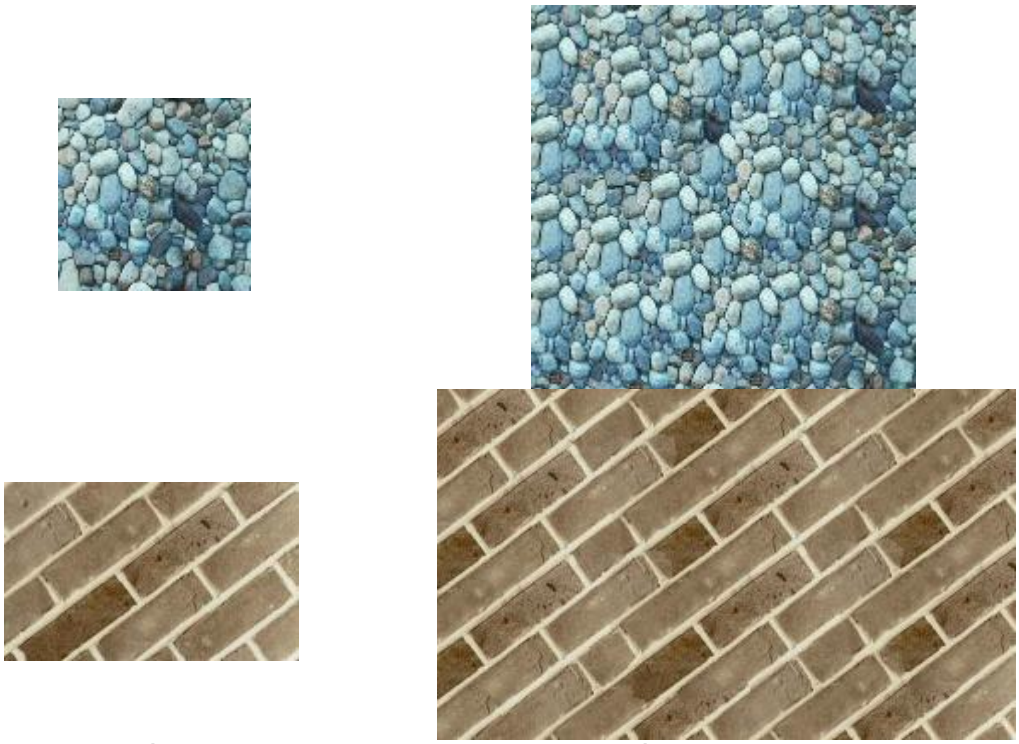


Figura 5.9: Síntese de texturas regulares e estocásticas com o casamento de *sub-patch.*

5.4. Algoritmo de corte mínimo

Dois algoritmos que encontram o corte mínimo em um grafo ponderado foram implementados, o Edmonds-Karp (1972) e o Dinic (1970). Apesar dos dois terem sido codificados, o Edmonds-Karp foi descartado rapidamente devido a apresentar uma performance muito inferior ao Dinic em praticamente todos os casos de entrada. Essa diferença pode ser interpretada pela diferença assintótica entre os algoritmos. Enquanto o Edmonds-Karp possui complexidade $O(VE^2)$, o Dinic roda em tempo $O(V^2E)$.

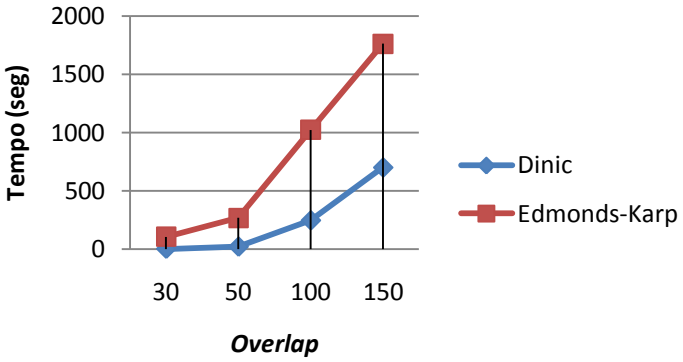


Gráfico 5.2: Tempo de processamento dos algoritmos de Dinic e Edmonds Karp em função do overlap. A textura usada como entrada foi a da figura 5.6.

6. Conclusão e trabalhos futuros

Durante a última década, síntese de texturas se consolidou como um campo forte de pesquisa em computação gráfica e processamento de imagens. Muitos algoritmos de síntese foram criados e publicados. Dentre eles, detalhamos aqui alguns dos mais importantes para o desenvolvimento da área.

O método no qual focamos este trabalho, o proposto por Kwatra et al. (2003) teve fundamentais contribuições no estado da arte do problema de síntese de texturas. Primeiramente, permitiu a criação de recortes dos *patches* com formatos arbitrários. Segundo, ele foi o primeiro algoritmo a levar em conta antigas costuras entre os *patches* no cálculo do custo do novo recorte. E por último, apesar de não ter sido comentado em detalhes neste trabalho, Kwatra et al. foram os primeiros a mostrarem uma técnica para síntese de vídeos a partir de uma técnica de síntese de texturas baseada em *patches.*

Uma versão do trabalho de Kwatra et al. foi também implementada, e seus resultados avaliados. Vimos que os métodos de posicionamento por *casamento de patch inteiro* e *casamento de sub-patch* são genéricos o suficiente para produzirem bons resultados para praticamente todas as texturas do espectro.

Algumas melhorias na implementação ainda podem ser adicionadas, como por exemplo, na etapa de refinamento (após a imagem de saída já ter sido totalmente preenchida), precisamos detectar uma região da textura de saída que tenha uma alta probabilidade de visíveis costuras entre os *patches*. Isto foi implementado através de uma abordagem de medir a descontinuidade entre pixels adjacentes na região. Porém dessa forma, só levamos em conta erros de suavidade na textura resultante, sem detectar possíveis erros estruturais.

A técnica de junção de *patches* pode ainda ser aplicada em outros contextos, como por exemplo fusão de imagens [Kwatra et al. 2003], transferência de texturas [Efros e Freeman 2001] e síntese de vídeos [Kwatra et al. 2003].

Outra possibilidade de trabalho futuro é retornar ao que motivou o desenvolvimento deste trabalho, que foi a ausência de um algoritmo de compressão de texturas para imagens geradas a partir do algoritmo de síntese com cortes em grafos. Desta forma, um novo algoritmo poderia ser criado dando uma contribuição de peso ao estado da arte da compressão de texturas.

Por fim, o desenvolvimento deste trabalho, através da pesquisa e implementação de trabalhos anteriores no campo, ajudou a obter uma forte familiaridade com problemas de processamento de imagens, sobretudo no ramo da síntese de texturas a partir de amostras, possibilitando novos rumos para pesquisas futuras na área.

Referências

Amanatides, J. (1987) Realism in Computer Graphics: A survey.

Blinn, J. F. E Newell, M. E. (1976) Texture and reflection in computer generated images.

Boykov, Y., Veksler, O., and Zabih, R. (1999) Fast approximate energy minimization via graph cuts. In International Conference on Computer Vision.

Brayner, F. L. (2009) Compressão de texturas utilizando síntese de texturas. February. Dissertação (Mestrado em Ciência da Computação), Universidade Federal de Pernambuco, Centro de Informática, Recife.

Burt, P. J., and Adelson, E. H. (1983) A multiresolution spline with application to image mosaics. ACM Transactions on Graphics

Catmull, E. (1974) A Subdivision Algorithm for Computer Display of Curved Surfaces. PhD thesis, Computer Science Department, University of Utah, Salt Lake City, Utah.

CImg. (2010) The CImg Library, disponível em http://cimg.sourceforge.net/, acessado em outubro de 2010.

Cormen, T.H., Leiserson, C.E., and Rivest, R.L. (1990) Introduction to Algorithms.

Cross, G., and Kain, A. K. (1983) Markov random field texture models.

Edmonds, J., and Karp, R. M. 1972. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems.

Efros, A. A., Leung, T. K. (1999) Texture Synthesis by Non-parametric Sampling

Efros, A. A., Freeman, T. W. (2001) Image Quilting for Texture Synthesis and Transfer.

Fleischer, K., Laidlaw, D., Currin, B., and Barr, A. (1995) Cellular texture generation.

Ford, L.R., Fulkerson, D.R. (1956) Maximal flow through a network.

Fu, K. S., and Lu, S.Y. (1978) Computer generation of texture using a syntactic approach.

Goldberg, A. V., and Tarjan, R. E. (1986) A New Approach to the Maximum Flow Problem.

Heeger, D. J., and Bergen, J. R. (1995) Pyramid-based texture analysis/synthesis.

Jungnickel, D. (2004) Graphs, Networks and Algorithms. Third Edition

Kaufmann, A. (1988) TSL – A Texture Synthesis Language.

Kilthau, S.L., Drew, M., and Moller, T. (2002) Full search content independent block matching based on the fast fourier transform.

Kleinberg, J., Tardos E. (2005) Algorithms Design.

Kwatra, V., Schödl, A., Essa, I., Turk, G. e Bobick, A. (2003) Graphcut Textures: Image and Video Synthesis Using Graph Cuts.

Kwatra, V., Adalsteinsson, D., Kim, T., Kwatra, N. Carlson, M. and Lin, M. (2007) Texturing Fluids .

Lin, W. C., Hays, J. H., Wu, C., Kwatra, V. and Liu, Y. (2004) A comparison study of four texture synthesis algorithms on regular and near-regular textures.

Monne, J., Schmitt, F., and Massaloux, D. (1981) Bidimensional texture synthesis by markov chains.

Shannon, C. E. (1948) A mathematical theory of communication. Bell Sys. Tech. Journal, 27.

Tarjan, R. E. (1984) A Simple Version of Karzanov's Blocking Flow Algorithm. Opns. Res. Lett. 2, 265-268.

Tong, X., Zhang,J., Liu, L., Wang, X., Guo, B., and Shum, H-Y. (2002) Synthesis of bidirectional texture functions on arbitrary surfaces. In SIGGRAPH ’02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques.

Turk, G. (1991) Generating textures on arbitrary surfaces using reaction-diffusion.

Walter, M. (1991) A obtenção de texturas na síntese de imagens realísticas num ambiente limitado. Dissertação (Mestrado em Ciência da Computação), Universidade Federal do Rio Grande do Sul

Walter, M., and Fournier, A. (1998). Clonal mosaic model for the synthesis of mammalian coat patterns.

Wei, L. Y., Levoy, M. (2000) Fast Texture Synthesis using Tree-structured Vector Quantization.

Wei, L-Y., Kwatra, V., Lefebvre, S., Turk, G. (2007) Example-Based Texture Synthesis. Course Notes for SIGGRAPH 2007

Worley, S. (1996) A cellular texture basis function.