Síntese de Texturas em Tempo-Real Baseada em *Patches*

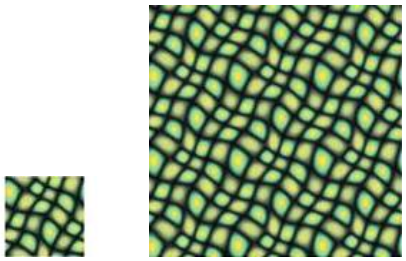
F.L. Brayner
Orientador: M. Walter
Centro de Informática - UFPE

Resumo

A síntese de textura baseada em amostras utiliza uma pequena textura como fonte para gerar uma textura que tem aparência bastante semelhante em relação à amostra inicial. Neste trabalho estendemos a técnica de síntese de textura baseada em patches *apresentada em [4] adaptando-a parcialmente para executar em uma GPU (*Graphics Processing Unit*). As comparações entre as implementação completamente em CPU e a extensão para GPU mostram a superioridade desta última por tomar proveito do paralelismo do processador gráfico. Mostramos que quanto maior a resolução (maior a quantidade de interpolações entre os pixels no processo de síntese) maior é a vantagem do algoritmo implementado em* GPU. *Apesar de ser vantajoso a execução em GPU ainda apresenta algumas limitações que serão explicadas juntamente com as possíveis modificações para reduzí-las.*

1. Introdução

Síntese de textura tem uma variedade de aplicações em Visão Computacional, e Processamento de Imagens. A utilização de texturas reais, ao contrário de texturas geradas proceduralmente, tem se mostrado uma das melhores opções para se atingir um bom grau de realismo nas imagens. Os algoritmos de síntese de textura utilizam uma pequena textura como fonte para gerar uma textura que tem aparência bastante semelhante em relação à amostra inicial (Fig. 1). Vantagens como poder ter a resolução da nova textura definida pelo usuário e a redução das atividades de modelagem ao criar essas texturas automaticamente podem ser obtidas como as técnicas de síntese. Ao longo dos anos, surgiram várias técnicas com esta finalidade, inicialmente baseadas em pixel [1], até as técnicas baseadas em *patches* (parte da amostra selecionada para gerar alguns pixels na imagem de saída). Neste último a textura final é formada pela junção de dois pedaços ou blocos da amostra original, com uma métrica RGB para selecionar os melhores *patches*. Neste trabalho estendemos a técnica de síntese de textura baseada em *patches* apresentada em [4] adaptando-a parcialmente para executar numa *GPU*. As modificações no algoritmo ocorreram no sentido de paralelizar na GPU a fase de interpolação dos pixels conhecida como *Feathering* [5]. Apesar de não ser o maior gargalo do processo síntese apresentado em [4], o tempo de execução do algoritmo é reduzido ao ter parte de sua execução realizada na *GPU* .



(a) Amostra (b) Textura gerada

Figura 1. Exemplo de síntese a partir de amostras

2. Trabalhos Relacionados

Síntese e análise de texturas são estudadas a mais de 50 anos na psicologia, estatística e visão computacional. Os trabalhos de Gibson [2] e Bela Julesz [3] na década de cinqüenta foram pioneiros e importantes para o desenvolvimento deste campo de pesquisa.Seus trabalhos sugeriam o processo de síntese de texturas em dois passos (1) selecionar o conjunto de estatísticas adequado (2) encontrar um algoritmo que encaixe essas estatísticas.

As abordagens para realização de síntese de textura evoluíram bastante. Muita pesquisa ocorreu no fim dos anos 70 e início dos anos 80. Os primeiros modelos tentavam garantir estatísticas globalmente, o que é uma tarefa difícil. Uma solução mais simples que surgiu posteriormente foi desenvolvida utilizando estatísticas localmente, um pixel por vez. Algoritmos de síntese baseados em pixel [1] de-

senvolvem uma imagem de saída pixel por pixel. Essa abordagem é interessante para texturas estocásticas mas geralmente falham na ocorrências de uma entrada mais coerente. Já síntese baseada em patches, copia partes da amostra de uma única vez na imagem final ao invés de pixels únicos. Por conta disso estas técnicas tendem a serem melhores e mais rápidas na captura da coerência visual em relação as técnicas baseadas em pixel.

3. Visão Geral

Nas próximas sub-seções entraremos em detalhes sobre as adaptações realizadas sobre [4] para que o mesmo executasse parcialemte numa *GPU*. A execução de um algoritmo em GPU requer algumas condições como: haver independência entre os pixels gerados pelo processador de fragmentos. Alguns algoritmos de síntese são mais facilmente adaptáveis a uma GPU que outros. Alguns trabalhos como [6] apresentam soluções que tem uma maior propensão para executar numa GPU. Para facilitar o entendimento vamos dividir o processo de síntese em dois passos: execução na *CPU* e a execução na *GPU*.

3.1. Execução na CPU

A parte do algoritmo de síntese textura que executa na CPU é o encontrado em [4] com algumas alterações para possibilitar a execução de parte do processo de síntese na *GPU*. O algoritmo de síntese de textura baseado em amostragem usa *patches* de textura da textura de entrada I_{in} como blocos para construir a textura sintetizada I_{out}. Em cada passo, é "colado"um bloco de textura B_k da textura de entrada I_{in} na textura sintetizada I_{out}. Para evitar falhas sobre as fronteiras dos *patches*, um determinado B_k deve ser escolhido baseado nos *patches* já colados em I_{out}, B_0 , ...,B_{k-1} . Por simplicidade usaremos amostras quadradas de tamanho predefinido W_b x W_b. O processo de construção da textura e a ordem com que os *patches* são colados são demonstrados em (Fig 2).

Sendo I_{R1} e I_{R2} dois patches de textura de mesmo tamanho e forma. Dizemos que I_{R1} e I_{R2} combinam *match* se d (I_{R1}, I_{R2}) $< \alpha$, onde d() representa a distância entre dois *patches* de textura e α é um constante predefinida. Como (Fig. 2) mostra, apenas parte da zona de B_k sobrepõe a zona de fronteira dos *patches* B_0 , ...,B_{k-1} em I_{out}. A seguir é demonstrado o processo de síntese e as modificações realizadas sobre o processo de síntese apresentado em [4], esses são os passos básicos de síntese em [4]:

(a) Randomicamente um patch B_0 de resolução W_bxW_b é escolhido a partir da amostra de entrada I_{in} . Põe B_0 no canto inferior esquerdo de I_{out} . Define $k = 1$.

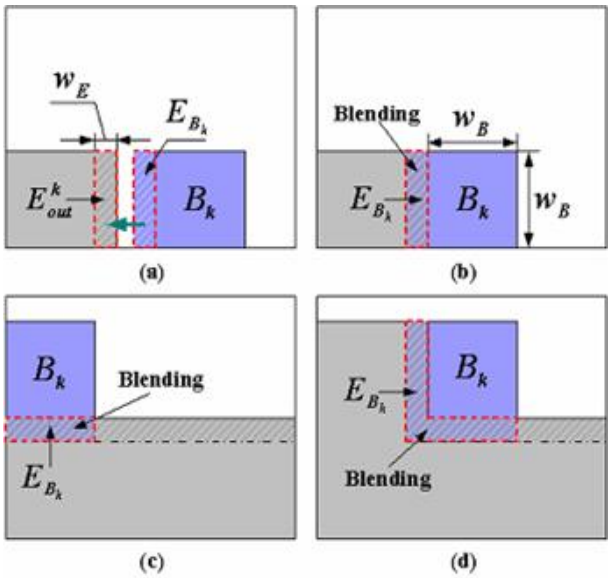


Fig.2: Síntese de textura baseada em patches. A área cinza já foi sintetizada. As áreas pontilhadas são as zonas de fronteiras. (a) Zonas de fronteiras que devem ser combinadas. (b), (c), e (d) são três configurações para o encaixe das zonas de fronteiras. As zonas de fronteira sobrepostas são mescladas.

Figura 2. Sintetizando a textura

(b) Forma-se o conjunto ψ de todas os *patches* da textura I_{in} tal que para cara patch de textura em ψ, sua região de fronteira combina com E^k_{out}.

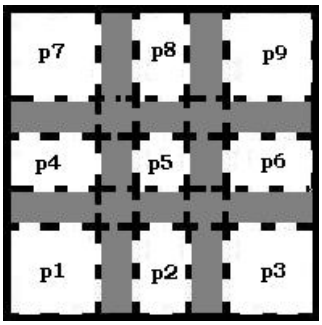
(c) Se ψ é um conjunto vazio, define ψ = B_{min} onde B_{min} é escolhido tal que sua zona de fronteira é a mais próxima de E^k_{out}.

(d)Seleciona-se randomicamente um elemento de Sb como o k-ésimo *patch* de textura B_k. Define $k = k + 1$.

(e) Repete os passos (b), (c), e (d) até que I_{out} esteja totalmente preenchida.

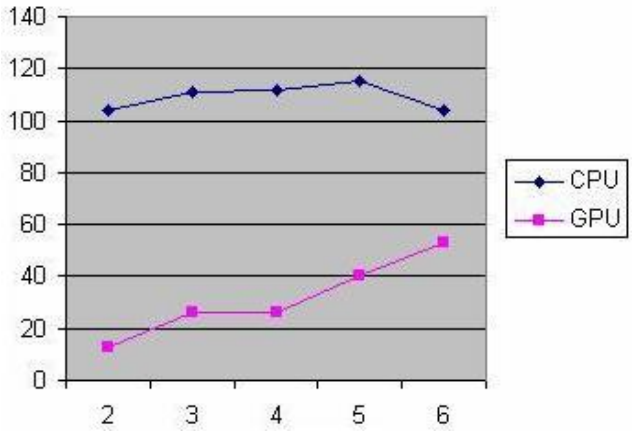
(f) Realiza a mescla nas zonas de fronteira.

O passo (f) utiliza um artifício conhecido como *feathering* [5] para prover uma transição suave entre *patches* adjacentes. Em nosso processo de síntese realizamos algumas adaptações nos passos comentados acima. As mudanças são as seguintes: substituímos os passos (b) e (c) de forma que ao invés de formarmos o conjunto ψ (passo bastante dispendioso) fazemos uma busca randômica na textura até que achemos um patch que combine ou seja satisfaça a condição: $d(E_{B(x,y)}, E^k_{out} < d_{max})$ ou um lim-



(a) Região da textura sintetizada onde é aplicado *feathering*. Esse passo pode tomar proveito do paralelismo de uma GPU pois a cor de um pixel nessa região não depende dos vizinhos.

Figura 3. Região onde ocorre a mescla entre os blocos



(a) $W_b = 60$ / $W_e = 16$

Figura 4. Relação milisegundos / k x k blocos Ex.1

ite I de iterações tenham sido realizadas. Caso o número máximo de iterações I não tenha sido suficiente para encontrar um B_k que combina com E^k_{out} , B_{min} é selecionado tal que sua zona de fronteira é a mais próxima de E^k_{out}. E por último, o ponto crucial para permitir que parte da síntese executasse em uma *GPU*, para cada *patch* selecionado suas coordenadas no espaço da amostra I_{in} são registradas. Veremos que estas coordenadas serão utilizadas na realização do *feathering*.

3.2. Execução na *GPU*

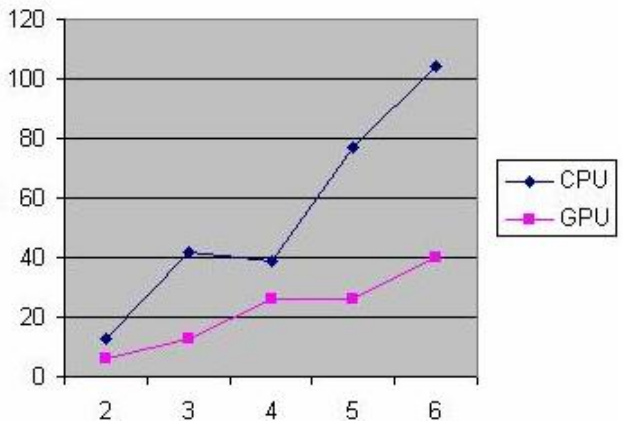
Após selecionados os blocos que formarão a textura final I_{out} estes ainda não foram mesclados. Em Fig. 3 é mostrado um exemplo típico da região que necessita de *feathering* de uma textura sintetizada utilizando 9 *patches*.

Perceba que a interpolação de cada pixel nas regiões de fronteira entre os *patches* é completamente independente dos pixels vizinhos, o que é uma característica ideal para tomar proveito do paralelismo em uma *GPU*. Em nosso caso, não nos interessa algum tipo de manipulação com vértices, logo o algoritmo é todo escrito como *fragment shader*. O *fragment shader* recebe como entrada alguns dados definidos pela parte anterior do processo de síntese executado até então na *CPU*, o tamanho da amostra e a quantidade k de blocos definidos para formar I_{out}. Outra informação dessa vez obtida pelo modelo no qual a textura sintetizada será renderizada são as coordenadas de textura, também conhecidas como coordenadas UV.
Com esse conjunto de informações, o *fragment shader* é capaz de definir quando passar para o *frame-buffer* a cor do pixel definido na amostra diretamente (caso a coordenada UV indique uma região que não precise de interpolação) ou quando o pixel a ser gerado não existe na amostra original e precisa ser gerado através do *feathering*. Observe Fig. 3, a parte cinza são os pixels que precisam de interpolação de pelo menos dois *patches* enquanto a parte branca é composta por pixels que podem ser recuperados diretamente da textura.

4. Resultados

Para medirmos a melhoria na performance do algoritmo, implementamos completamente em CPU a proposta apresentada em [4]. E criamos uma outra versão do programa onde a fase de *Feathering* é executada na *GPU*. Utilizando $W_b = 60$ e $W_e = 16$ medimos o tempo para a execução do *Feathering* nas duas versões e obtemos o seguinte gráfico apresentado em Fig. 4.

Observe que quando a quantidade de interpolações executadas no *Feathering* é relativamente pequena (quantidade de blocos menor) o tempo de execução na CPU é relativamente estável. Apesar de estar com tempo sempre abaixo da execução em *CPU*, o *Feathering* na GPU foi mais afetado pelo aumento da quantidade de interpolações.
Em um outro exemplo, aumentamos consideravelmente a quantidade de pixels que precisam ser gerado através de interpolação utilizando $W_b = 60$ e $W_e = 28$ (esse exemplo é apenas para aumentar a quantidade de interpolações, observe que a relação W_b/W_e não gera uma textura final aceitável, o mesmo efeito poderia ser conseguido aumentando a quantidade de blocos, não foi feito assim devido a limitações explicadas a seguir), a verão em *GPU* nesse caso obteve ainda mais vantagem sobre a versão em *CPU*. Com esses exemplos, fica claro



(a) $W_b = 60$ / $W_e = 28$

Figura 5. Relação milisegundos / k x k blocos Ex.2

a redução do tempo de execução da fase de *Feathering* quando esta é implementada numa *GPU*. A vantagem em relação a *GPU* se torna ainda maior quando temos uma maior quantidade de pixels que necessitam de interpolação.

Apesar de apresentar superioridade em relação a implementação feita puramente na CPU, algumas limitações são impostas na programação em GPU. No nosso caso utilizamos a *GeForce 6200* e shader escrito em GLSL (*OpenGL Shading Language*). Destas limitações, a que mais afetou a forma com que o algoritmo de síntese foi escrito é a seguinte: um array ao ser declarado em um shader deve ter um tamanho conhecido em tempo de compilação e o acesso ao mesmo apresenta restrição semelhante. Com isso, consultas aos dados que seriam diretamente acessados a partir de um índice, são obrigadas a serem feitas dentro de um laço prejudicando a performance do *shader*. Outra limitação é o numero de registradores da placa que limita a quantidade de coordenadas a serem passadas da *CPU* para a *GPU*. Em nosso caso ficamos limitados a utilizar uma quantidade máxima de 36 blocos durante a síntese. Um outro problema que encontramos é em relação a utilização de filtros de textura. Para filtros mais simples como *GL-NEAREST* o comportamento é adequado, para filtros um pouco mais elaborados como *GL-LINEAR* surgem alguns artefatos no modelo renderizado.

5. Trabalhos Futuros

Como trabalho futuro podemos sugerir uma análise mais profunda da parte do algoritmo que executa em CPU para que o mesmo também possa ser escrito em GPU.Na parte que já executa em GPU algumas melhorias podem ser feitas, principalmente em relação as limitações impostas pela GPU. Uma possível forma de diminuir a restrição da quantidade de blocos utilizados na síntese e aumentar a eficiência do *fragment shader* é passar os dados gerados pela parte do algoritmo em CPU como dados de textura. Dessa forma seria evitada a utilização de arrays dentro do *shader* e o gargalo necessário para acessar os elementos no mesmo. Temos também que investigar as modificações necessárias, provavelmente no *fragment shader*, para que não surjam artefatos na utilização de algumas filtragens de textura, como GL-LINEAR. Podemos ter também uma análise mais precisa do impacto da quantidade de interpolações e da resolução da textura final no tempo de execução da *GPU* utilizando *profilers*.

6. Conclusões

A passagem da etapa de ”Feathering”para executar na GPU tornou o algoritmo apresentado em [4] mais eficiente. Essa vantagem decorre pelo fato da etapa de feathering ser passível de execução em paralelo, pois cada pixel pode ser definido de forma independente, caracterítica fundamental para execução em uma GPU. Os gráficos apresentados em Figs. 4 e 5 mostram também que quanto maior a quantidade de interpolações entre os pixels no processo de síntese, maior é a vantagem do algoritmo implementado em *GPU*. Apesar de ser vantajoso a execução em GPU ainda apresenta algumas limitações impostas pela execução em GPU. A declaração e o acesso de arrays só é possível através de variáveis constantes em tempo de compilação o que afeta a performance do shader além de torná-lo pouco extensível.

Referências

[1] J. S. D. Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97*, pages 361–368, New York, NY, USA, 1997.

[2] J. Gibson. *The Perception of the Visual World*. Houghton Mifflin, Boston USA, 1950.

[3] B. Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*, 8(2):84–92.

[4] L. Liang, C. Liu, Y.-Q. Xu, B. Guo, and H.-Y. Shum. Real-time texture synthesis by patch-based sampling. *ACM Trans. Graph.*, 20(3):127–150, 2001.

[5] R. Szeliski and H.-Y. Shum. Creating full view panoramic mosaics and environment maps. Aug.

[6] L.-Y. Wei and M. Levoy. Order-independent texture synthesis.