Texturas com Relevo utilizando Processamento Paralelo

Francisco Fonseca
Bruno Feijó
Marcelo Dreux
Esteban Clua
Pontifíca Universidade Católica do Rio de Janeiro
Departamento de Informática
R. Marquês de São Vicente, 225, Gávea, Rio de Janeiro, RJ, Brasil
{ffonseca,bruno,esteban}@inf.puc-rio.br
dreux@mec.puc-rio.br

Abstract

Com o aumento do poder de processamento do hardware gráfico, também denominado GPU (Graphic Processor Unit), cada vez mais tem crescido a tendência de que este assuma a maior parte do pipeline de visualização, deixando a unidade central de processamento (CPU, Central Processor Unit) com um tempo ocioso cada vez maior. No sentido de aproveitar este tempo, o presente trabalho apresenta duas formas de paralelizar a técnica de mapeamento de textura com relevo na CPU, utilizando-se da GPU somente para cálculos de shading, permitindo que tal técnica seja executada em tempo real juntamente com um algoritmo de iluminação por pixel.

1 Introdução

Manuel Oliveira, em 2000, introduziu mapeamento de textura com relevo como uma técnica para representar detalhes de superfíces tridimensionais [7]. Ao contrário da técnica de mapeamento de textura tradicional [1], em que a ausência de paralaxe[1] revela a aparência não tridimensional da textura, mapeamento de textura com relevo suporta paralaxe do movimento da visão. No entanto, pelo fato de não armazenar informação geométrica suficiente referente aos detalhes a serem simulados, a técnica proposta por Oliveira não permite a representação correta de superfícies não difusas e; além disso, o *overhead* imposto pelo passo de *pre warping* [7] impede que a técnica seja utilizada em aplicações com requerimento de tempo real.

[1]Deslocamento da posição aparente de um corpo devido à mudança de ponto de vista do observador.

Desde então, alguns trabalhos foram propostos com o intuito de aprimorar a técnica de mapeamento de textura com relevo.

Dentre esses, pode-se citar o trabalho de Fujita e Kanai [3] que extende a técnica de mapeamento de textura com relevo para suportar efeitos de *shading* calculados por *pixel* (tais como, mapeamento de reflexão e mapeamento de normal) aproveitando-se da programabilidade oferecida pelos *hardwares* gráficos mais recentes. No entanto, este método ainda não atende aos requisitos de tempo real.

Nesta mesma linha é possível citar o trabalho de Policarpo [8] que implementa todo o processamento exigido pela técnica proposta por Oliveira no *hardware* gráfico. Na abordagem de Policarpo, a direção de visão é transformada para o espaço de textura permitindo que uma busca linear seja realizada para localizar uma interseção com a superfície virtual (representada pelo informação de profundidade). Esta busca linear é posteriormente refinada através de uma busca binária para encontrar um ponto de interseção mais preciso. Este método atende aos requisitos de tempo real bem como suporta efeitos de *shading* calculados por *pixel*, no entanto, pelo fato de concentrar o esforço computacional inteiramente no *hardware* gráfico, a técnica proposta por Policarpo exige a utilização das placas de vídeo de última geração como, por exemplo, a placas *GeForce* da série 6.

Vale notar que com o aumento do poder de processamento do *hardware* gráfico, também denominado GPU (*Graphic Processor Unit*), cada vez mais tem crescido a tendência de que este assuma a maior parte do *pipeline* de visualização, deixando a unidade central de processamento (CPU, *Central Processor Unit*) com um tempo ocioso cada vez maior. No sentido de aproveitar este tempo, o presente trabalho apresenta duas formas de paralelizar a técnica de

mapeamento de textura com relevo na CPU, utilizando-se da GPU somente para cálculos de *shading*, permitindo que a técnica proposta por Oliveira seja executada em tempo real juntamente com um algoritmo de iluminação por *pixel*, mesmo com placas de vídeo menos poderosas.

A contribuição apresentada neste artigo consiste em duas novas abordagens paralelas propostas com o intuito de otimizar o processamento empregado no algoritmo de mapeamento de textura com relevo, sendo que em alguns casos são obtidos ganhos de até 37% em relação ao tempo de processamento da abordagem convencional.

2 Mapeamento de Textura com Relevo

Uma textura com relevo é uma imagem com informação de profundidade obtida através de uma câmera de projeção paralela.

Em termos matemáticos, uma textura com relevo é um par $\{i, K\}$, onde i é uma imagem digital e K é um modelo de câmera de projeção paralela associado a i. Cada elemento do espaço de cor de i é aumentado para incluir um valor escalar representando a distância (profundidade), no espaço Euclideano, entre o ponto amostrado e uma entidade de referência. Uma vez que K é um modelo de câmera de projeção paralela, a entidade de referência é o plano de projeção de K.

Em termos computacionais, o elemento i de uma textura com relevo pode ser representado por uma imagem RGBA onde o canal *alpha* armazena valores de profundidade. As informações referentes ao modelo de câmera K podem ser representadas por uma matriz 4x4.

Em 1997, Leonard Mcmillan introduziu o conceito de *warping* tridimensional de imagens [5] que é a base para a técnica de mapeamento de textura com relevo. A técnica proposta por Mcmillan consiste em uma transformação geométrica que mapeia uma imagem fonte com profundidade $\{i_1, K\}$ sobre uma imagem destino i_2, dessa forma, possibilitando a visualização correta da imagem fonte i_1 de diversos pontos de vista.

Vale notar que o *warping* tridimensional de imagens pode ser interpretado como uma composição de duas transformações bidimensionais: uma transformação perspectiva planar e um deslocamento por *pixel* na direção do epipolo[2]. Dessa forma, o mapeamento de textura com relevo é uma fatoração do *warping* tridimensional de imagens nestas duas componentes. Tal fatoração, proposta por Oliveira [7], permite que a transformação perspectiva planar (essencialmente uma operação de mapeamento de textura) seja aplicada após o deslocamento na direção do epipolo (*pre warping*), dessa forma, beneficiando-se do mapeamento de textura implementado em *hardware* para realizar a transformação final. Veja a Figura 1. Mais detalhes podem ser obtidos na tese de doutorado de Oliveira [7].

[2]A projeção do centro de projeção de uma câmera sobre o plano de projeção de uma outra câmera.

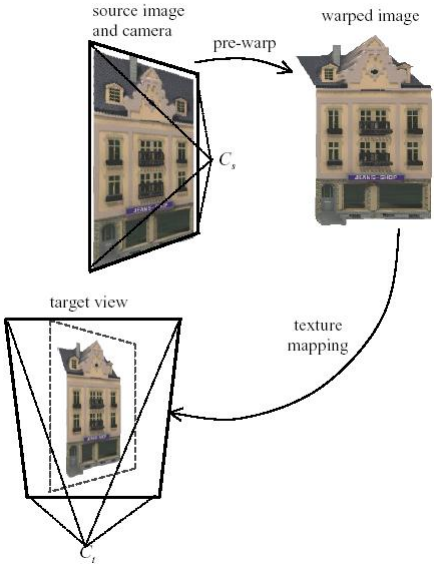

Figure 1. Mapeamento de textura com relevo.

2.1 Implementação

Como definido na Seção 2, uma textura com relevo é composta de cor, profundidade e informação de câmera. Porém, somente com tais informações não é possível capturar efeitos dependentes do ponto de vista e da direção de iluminação. Uma possível solução para representar tais efeitos é o uso de mapas de normais em conjunção com mapeamento de textura com relevo.

Dessa maneira, no caso específico deste trabalho, um mapa de normal é associado a uma textura com relevo. A informação de normal é combinada com profundidade gerando um mapa de normal com profundidade, que é representado como uma imagem RGBA onde o canal *alpha* armazena valores de profundidade e os canais de cor armazenam o vetor normal. A informação de cor é armazenada num mapa de textura convencional que é representado como uma imagem RGB.

A estratégia de implementação adotada para realizar o mapeamento de textura com relevo consiste em um algoritmo de cinco passos, que pode ser representado pelo diagrama da Figura 2.

O algoritmo recebe como dados de entrada a posição corrente p do observador, um quadrilátero q e uma textura com relevo $\{i_s, K_s\}$, onde i_s representa um mapa de normal com profundidade bem como um mapa de textura convencional. Num primeiro momento, é necessário reinicializar os recursos utilizados durante a execução do algoritmo. Tais re-

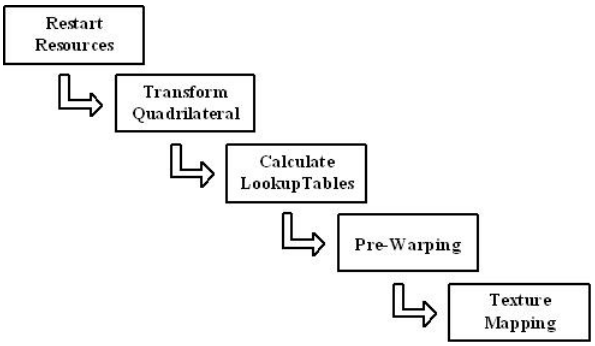

Figure 2. Diagrama do algoritmo de mapeamento de textura com relevo.

cursos são basicamente *buffers* de imagens e *lookup tables*. Devido à aplicação gráfica funcionar como um *loop* em que a cada passo a imagem na tela é atualizada, a reinicialização torna-se necessária para evitar a leitura de informações geradas em execuções anteriores.

Uma vez que a etapa de reinicialização tenha sido concluída, os parâmetros do quadrilátero q são transformados de acordo com a configuração corrente de visualização. Em seguida, algumas *lookup tables* são calculadas com o intuito de evitar cáculos repetitivos, dessa forma, otimizando parte do processo.

A próxima etapa, denominada *pre warping*, é o coração do algoritmo sendo a etapa que mais consome tempo e processamento computacional. Tal etapa é responsável por produzir uma imagem de saída i_t que represente uma visualização parcial do mapeamento de $\{i_s, K_s\}$ sobre q visto de p.

Desse modo, uma vez que a imagem i_t tenha sido produzida seu conteúdo pode ser transferido para a memória de textura da placa de vídeo e, finalmente, mapeada sobre o quadrilátero q para que uma visualização correta seja obtida. Mais detalhes sobre esta implementação podem ser obtidos na dissertação de mestrado de Fonseca [2].

3 Processamento Paralelo

No contexto de aplicações gráficas em tempo real, costuma-se utilizar uma divisão do *pipeline* de renderização em três estágios conceituais: *aplicação*, *geometria* e *rasterização* [6]. Atualmente, os dois últimos estágios são implementados em *hardware* pela placa gráfica, enquanto que o primeiro estágio é implementado na CPU.

Esta mesma divisão pode ser utilizada para representar todo o processamento envolvido durante o mapeamento de textura com relevo. O diagrama da Figura 3 ilustra os estágios do *pipeline* de renderização de texturas com relevo de acordo com a abordagem descrita na Seção 2.1.

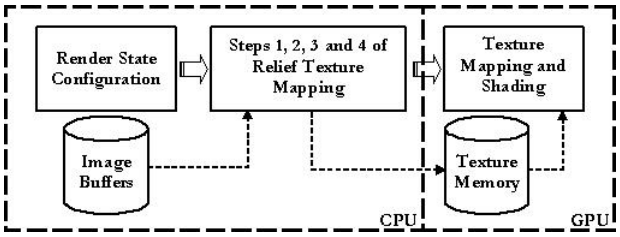

Figure 3. Processo convencional de mapeamento de texturas com relevo.

Neste diagrama, o estágio da aplicação é executado de forma seqüencial e compreende todos os passos realizados na CPU. Os estágios de geometria e rasterização são implicitamente representados pelo passo de **mapeamento de textura e *shading***, realizado na GPU.

Por definição, a velocidade de um *pipeline* é determinada pelo seu estágio mais lento, independente de quão rápido são os outros estágios. Geralmente, este estágio mais lento é denominado de *gargalo*.

Segundo Akenine-Möller & Haines [6], o primeiro passo na otimização de um *pipeline* é localizar o gargalo. Tal localização pode ser efetuada através da realização de um conjunto de testes propostos em [6].

Analisando-se o algoritmo descrito na Seção 2.1, tem-se o sentimento de que o gargalo do processo de mapeamento de texturas com relevo é o estágio da aplicação, uma vez que todos os *texels* do mapa de normal e do mapa de cor são processados uma vez por quadro. Dessa maneira, decidiu-se num primeiro instante realizar os testes referentes a este estágio.

Uma maneira de verificar se o estágio da aplicação limita a velocidade de renderização é enviar dados através do *pipeline* de forma que os outros estágios realizem pouco ou nenhum trabalho. Em *OpenGL* isto pode ser alcançado substituindo-se todas as chamadas de `glVertex3f` e `glNormal3f` por chamadas de `glColor3f`. Dessa forma, o trabalho de envio de dados da CPU permanece inalterado enquanto que o trabalho de envio e recebimento de dados nos estágios da geometria e rasterização é altamente reduzido. Logo, se o desempenho não melhorar, é possível afirmar que o gargalo é o estágio da aplicação. Através da realização deste teste para várias texturas de entrada, constatou-se que o estágio da aplicação é realmente o gargalo.

Diante das observações anteriores, decidiu-se implementar duas abordagens paralelas para o processo envolvido no mapeamento de texturas com relevo. No que se segue, é apresentada para cada abordagem a motivação que permitiu sua implementação bem como a descrição da metodologia

adotada para sua realização.

3.1 Abordagem Paralela

Com o intuito de otimizar o processamento empregado na CPU, criou-se uma *thread* de CPU para executar somente os quatro primeiros passos do algoritmo de mapeamento de textura com relevo. Com o auxílio da tecnologia de *Hyper-Threading* [4] tal *thread* pode ser executada em paralelo com o processo convencional da CPU, permitindo um ganho considerável de tempo. O diagrama da Figura 4 ilustra o novo procedimento.

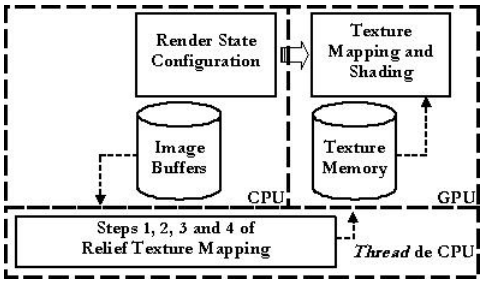

Figure 4. Diagrama da abordagem paralela.

Para garantir que o resultado final seja correto, além de criar a *thread* de CPU é necessário sincronizá-la com o restante dos processos em andamento. Tal sincronização é representada pela máquina de estados ilustrada na Figura 5.

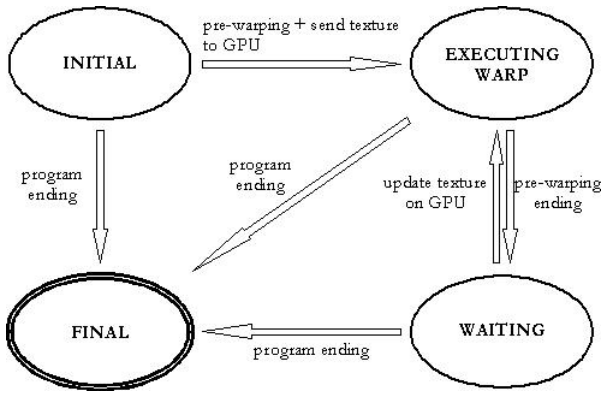

Figure 5. Máquina de estados da thread de CPU.

No momento em que a *thread* é criada seu estado corrente é o estado **inicial**. A transição do estado **inicial** para o estado **executando warp** é feita através da primeira realização da operação de *warp* seguida do envio da imagem resultante desta operação para a memória de textura da placa de vídeo. No restante da execução, o autômato permanece num laço entre os estados **executando warp** e **esperando**, salvo em situações de término de programa em que a *thread* de CPU é levada para o estado **final**. No estado **executando warp** o processo realiza a operação de pré *warping* que, uma vez terminada, causa a transição para o estado **esperando**. Neste estado, a nova imagem resultante é enviada para a memória de textura, desta forma, atualizando o seu conteúdo. Uma vez que a memória de textura é atualizada ocorre a transição para o estado **executando warp** indicando que uma nova operação de *warping* pode ser efetuada.

Assumindo-se que a máquina utilizada para executar o algoritmo objeto deste trabalho possui a tecnologia de *Hyper-Threading*, um processador pode ser designado a permanecer exclusivamente dedicado à *thread* de CPU sem que o desempenho do *pipeline* gráfico seja prejudicado. Portanto, a *thread* pode ser criada com a prioridade normal.

A transição do estado **inicial** para o estado **executando warp** ocorre durante a execução da rotina principal de desenho realizada uma vez por quadro. Abaixo é apresentado um pseudo-código para esta rotina.

algoritmo `draw`()
```
 1 Configura o estado de renderização da API OpenGL;
 2 Ativa programa de vértice;
 3 Ativa programa de fragmento;
 4 Se é a primeira execução de draw então
 5     Reinicializa recursos;
 6     Transforma quadrilátero;
 7     Inicializa lookup tables;
 8     Realiza pre warping;
 9     Envia textura processada para GPU;
10     estadoThread ← EXECUTANDO_WARPING;
11 senão
12     Se estadoThread = ESPERANDO então
13         Atualiza textura na GPU;
14         estadoThread ← EXECUTANDO_WARPING;
15     fim_se
16 fim_se
17 Desenha o quadrilátero com a textura processada;
18 Desativa programa de fragmento;
19 Desativa programa de vértice;
fim
```

Na linha **1** é configurado o estado de renderização da API *OpenGL*. Tal configuração consiste na habilitação de operações como *blending*, *culling*, testes de profundidade, entre outras. Nas linhas **2** e **3** os programas de vértice e fragmento para o cálculo de iluminação por *pixel* são ativados, respectivamente. Na primeira vez que a rotina `draw` é executada, os quatro primeiros passos do algoritmo de mapeamento de textura com relevo (linhas **5** a **8**) são realizados e, em seguida, a textura resultante do *warping* é enviada para a placa de vídeo através da rotina `glTexImage2D` disponibilizada pela API *OpenGL*. Uma vez que a textura

é enviada para a placa de vídeo, a transição do estado **inicial** para o estado **executando warping** pode ser efetuada (linha **10**). Nas circunstâncias em que a rotina `draw` é executada pela segunda vez em diante, necessita-se verificar se o estado corrente da *thread* de CPU é **esperando**. Em caso afirmativo, significa que a *thread* realizou uma operação de *warping* e, conseqüentemente, a textura residente na placa de vídeo pode ser atualizada (linha **13**). Neste caso, esta atualização é efetuada pela função `glTexSubImage2D` também disponibilizada pela API *OpenGL*. Em seguida, a *thread* de CPU sofre uma transição do estado **esperando** para o estado **executando warping** e uma nova operação de *warping* pode ser realizada. Finalmente, na linha **17**, o quadrilátero com a textura mapeada pode ser desenhado e, nas linhas **18** e **19**, os programas de fragmento e vértice são desativados, respectivamente. Note que durante toda a realização da rotina `draw`, a *thread* de CPU é executada em paralelo.

As transições de estado remanescentes são descritas pelo pseudo-código descrito a seguir, onde a *thread* de CPU é implementada.

```
algoritmo threadCPU()
 1  Enquanto verdadeiro faça
 2     Se estadoThread = EXECUTANDO_WARPING então
 3        Reinicializa recursos;
 4        Transforma quadrilátero;
 5        Inicializa lookup tables;
 6        Realiza pre warping;
 7        estadoThread ← ESPERANDO;
 8     senão se estadoThread = FINAL então
 9        retorna 0;
10     fim_se
11  fim_enquanto
fim
```

O laço da linha **1** é responsável por fazer com que a *thread* execute até que seu estado corrente seja modificado para o estado **final**. Durante a realização do laço, se o estado corrente é **executando warping**, os quatro primeiros passos do algoritmo de mapeamento de textura com relevo são executados (linhas **3** a **6**) e a transição para o estado **esperando** é realizada (linha **7**) indicando que uma operação de atualização de textura pode ser realizada. Para o caso em que o estado corrente da *thread* é o estado **final**, a execução da rotina `threadCPU` é finalizada.

3.2 Abordagem Multi-Threaded

Além da abordagem anterior, o uso da tecnologia de *Hyper-Threading* permite que haja um processamento paralelo para diferentes partes dos dados de entrada. Mais especificamente, é possível dividir a textura de entrada $\{i_s, K_s\}$ em duas partes e executar, simultaneamente, os quatro primeiros passos do algoritmo de mapeamento de textura com relevo para cada uma destas partes. Conseqüentemente, um pós processamento é necessário para unir os dois resultados obtidos numa única textura resultante para que a mesma possa ser mapeada sobre o quadrilátero q. O diagrama da Figura 6 ilustra este procedimento.

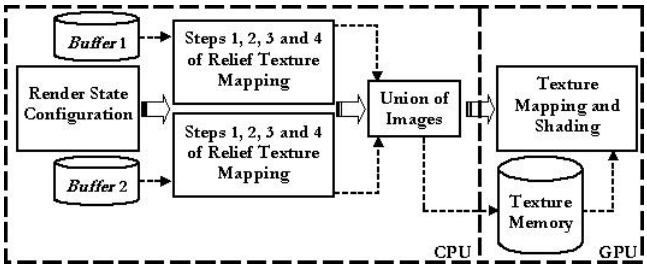

Figure 6. Diagrama do processo multi-threaded.

Inicialmente, é necessário dividir o *buffer* de imagens em duas partes. Vale notar que o termo *buffer de imagens* compreende o mapa de normal com profundidade juntamente com o mapa de cor utilizados como entrada para o algoritmo. Ao contrário do que se poderia imaginar, as duas imagens resultantes do processo de divisão possuem a mesma dimensão da imagem original, por exemplo, a metade esquerda da primeira imagem resultante será preenchida com a metade esquerda da imagem original enquanto que a metade direita da primeira imagem resultante será preenchida com valores inválidos. Veja a Figura 7.

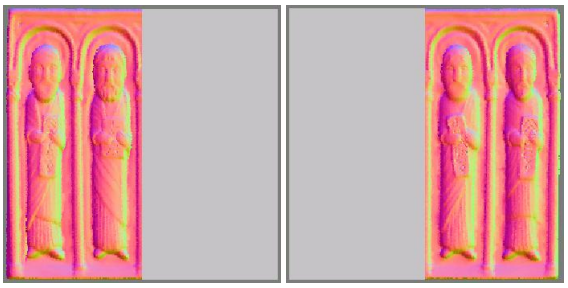

Figure 7. Ilustração do processo de divisão do mapa de normal com profundidade.

Isto é necessário porque durante o processo de pré *warping*, alguns *texels* poderiam deslocar-se além dos limites do plano de suporte da imagem, se esta realmente tivesse a metade do tamanho da imagem original. Conseqüentemente, as imagens resultantes do pré *warping* compreenderiam uma visão incompleta da superfície representada fazendo com que o resultado da união de ambas

também representasse uma visão incompleta. Veja a Figura 8.

Figure 8. Visão incompleta da superfície representada.

No entanto, somente manter o tamanho original das texturas não garante que o resultado final seja correto. Durante o processo de união das duas imagens resultantes poderá haver sobreposição de *texels* nas regiões em que o limite do plano de suporte das imagens foi ultrapassado. Dessa forma, nos casos em que há sobreposição de *texels* descarta-se aquele que possui um valor inválido armazenado.

A rotina `draw`, descrita na Seção 3.1, deve ser modificada para corresponder a esta nova abordagem. Adiante segue um pseudo-código para esta nova rotina.

As modificações em relação à rotina da Seção 3.1 se concentram entre as linhas **5** e **16**, o restante permanece inalterado e, portanto, não requer considerações adicionais. Na primeira vez que a rotina `draw` é executada, os quatro primeiros passos do algoritmo de mapeamento de textura com relevo (linhas **5** e **6**) são realizados para as duas metades da imagem original. Uma vez que o processo de *warping* é concluído, a união das imagens resultantes pode ser efetuada. Em seguida, a textura resultante de tal união é enviada para a placa de vídeo. Uma vez que a textura é enviada para a placa de vídeo, a transição do estado **inicial** para o estado **executando warping** pode ser efetuada (linhas **9** e **10**) para cada *thread*. Nas circunstâncias em que a rotina `draw` é executada pela segunda vez em diante, necessita-se verificar se o estado corrente da *thread* de CPU, para as duas metades, é **esperando**. Em caso afirmativo, significa que cada *thread* realizou uma operação de *warping* e, conseqüentemente, o resultado de cada *warping* pode ser unido para que a textura residente na placa de vídeo seja atualizada (linha **14**). Em seguida, cada *thread* de CPU sofre uma transição do estado **esperando** para o estado **executando warping** e uma nova operação de *warping* pode ser realizada.

algoritmo `draw`()
```
 1  Configura o estado de renderização da API OpenGL;
 2  Ativa programa de vértice;
 3  Ativa programa de fragmento;
 4  Se é a primeira execução de draw então
 5      Realiza 4 primeiros passos para metade 1;
 6      Realiza 4 primeiros passos para metade 2;
 7      Une as imagens resultantes;
 8      Envia textura processada para GPU;
 9      estadoThread1 ← EXECUTANDO_WARPING;
10      estadoThread2 ← EXECUTANDO_WARPING;
11  senão
12      Se estadoThread1 = ESPERANDO e
              estadoThread2 = ESPERANDO então
13          Une as imagens resultantes;
14          Atualiza textura na GPU;
15          estadoThread1 ← EXECUTANDO_WARPING;
16          estadoThread2 ← EXECUTANDO_WARPING;
17      fim_se
18  fim_se
19  Desenha o quadrilátero com a textura processada;
20  Desativa programa de fragmento;
21  Desativa programa de vértice;
```
fim

4 Resultados

Nesta seção, são apresentadas as estatísticas de tempo consumido dos algoritmos implementados bem como os resultados visuais obtidos. Tais medidas foram realizadas sobre uma máquina *Intel Pentium IV PC* executando a 2.66GHz com 512Mb de memória *RAM* e uma placa de vídeo *GeForce FX* 5600 com 256Mb de memória de vídeo.

Para realização do experimento, escolheu-se três conjuntos de amostras (Figura 9, no final da seção). A Tabela 1 apresenta alguns dados para as amostras utilizadas como, por exemplo, resolução (em *pixels*) e o número de *texels* com valor de profundidade inválido (isto é, *texels* que representam *background*).

Amostra	**Resolução**	***Texels* Inválidos**
1	256x256	3800 (5.80%)
2	256x256	3223 (4.92%)
3	256x256	30484 (46.51%)

Table 1. Amostras para realização dos experimentos.

A Tabela 2 apresenta as taxas de quadros por segundo para cada uma das abordagens implementadas. A abordagem Seqüencial (S) representa a técnica tradicional de

mapeamento de textura com relevo enquanto que as abordagens Paralela (P) e *Multi-Threaded* (M) são os algoritmos propostos neste trabalho.

Abordagem	Amostra 1		Amostra 2		Amostra 3	
	$\overline{fps}$	*DP*	$\overline{fps}$	*DP*	$\overline{fps}$	*DP*
S	24.18	0.86	24.93	1.54	38.24	1.86
P	97.97	4.50	96.45	5.01	103.64	4.88
M	33.14	6.02	31.09	7.67	34.51	8.21

Table 2. Média de quadros por segundo para as abordagens seqüencial, paralela e multi-threaded, onde, fps significa frames per second e DP significa desvio padrão.

Analisando-se a Tabela 2 é possível concluir que a abordagem paralela é superior à abordagem seqüencial. No entanto, a abordagem seqüencial é a mais estável de todas em relação ao número médio de quadros por segundo, o que pode ser comprovado pela baixa dispersão dos dados apresentada pelo desvio padrão obtido.

Somente no caso da amostra 3 a abordagem *multi-threaded* foi inferior à abordagem seqüencial. Para explicar este fato existem dois pontos a serem considerados nesta análise. O primeiro ponto se refere à amostra 3 que é um tipo de amostra especial onde o número de *texels* inválidos é muito grande (veja Tabela 1) e, portanto, o processamento empregado durante a etapa de pré *warping* é inferior em relação às amostras 1 e 2, uma vez que somente *texels* válidos necessitam ser processados. O segundo ponto se refere à abordagem *multi-threaded* que possui um *overhead* de três processos executados sobre dois processadores. Logo, pode-se concluir que o *overhead* inerente à abordagem *multi-threaded* foi superior ao ganho obtido pela otimização da etapa de pré *warping* para o caso da amostra 3 comprovando, desta forma, a inferioridade da abordagem *multi-threaded* em relação à abordagem seqüencial neste tipo específico de situação.

É importante observar que as altas taxas de quadros por segundo obtidas pela abordagem paralela não refletem a quantidade de vezes que uma operação de *warping* é realizada; na verdade, tais taxas se referem à quantidade de imagens renderizadas por segundo, independente se uma nova operação de *warping* foi executada ou não. Isto ocorre uma vez que o processo de *warping* é executado em paralelo com o restante do processo e, conseqüentemente, é possível notar uma atualização progressiva na imagem renderizada quando há muito movimento da câmera.

Além disso, a superioridade da abordagem paralela em relação à abordagem *multi-threaded* já era esperada uma vez que a última consome um tempo razoável durante a realização da operação de união de texturas.

Os resultados visuais obtidos podem ser vistos na Figura 10.

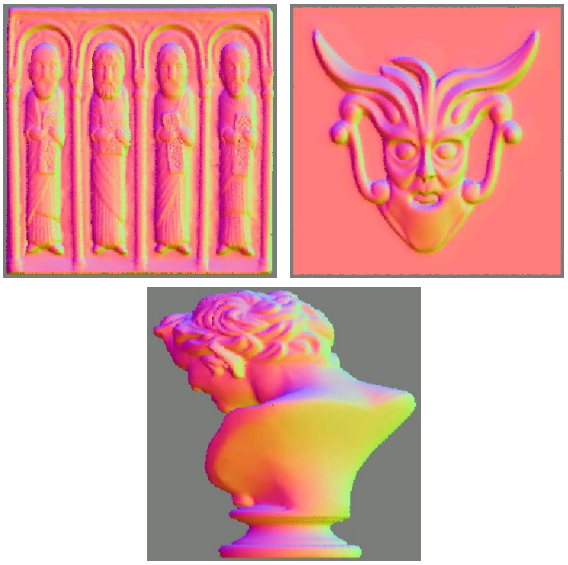

Figure 9. Amostras 1 (esquerda), 2 (direita) e 3 (abaixo).

Figure 10. Resultados visuais.

5 Conclusão e Trabalhos Futuros

Com o intuito de otimizar o processo de mapeamento de texturas com relevo, foram implementadas duas novas abordagens:

Paralela. Na abordagem paralela criou-se uma *thread* de CPU para executar os quatro primeiros passos do algoritmo de mapeamento de textura com relevo. Tal *thread* é executada em paralelo com o processo principal da CPU, utilizando-se para este fim a tecnologia de *Hyper-Threading*.

Multi-Threaded. Nesta abordagem, dividiu-se a textura de entrada em duas partes com o intuito de executar, simultaneamente, os quatro primeiros passos do algoritmo de mapeamento de textura com relevo para cada uma destas partes, sendo necessário um pós processamento para unir os dois resultados obtidos numa única textura resultante.

Diante disso, pode-se concluir que a paralelização do processo de mapeamento de textura com relevo demonstra-se favorável uma vez que são obtidos ganhos de até 37% em relação ao tempo de processamento da abordagem convencional.

Em relação aos trabalhos futuros, existem duas questões de otimização que podem resultar em trabalhos interessantes: a utilização de um *pipeline* de multi-processadores no processo mapeamento de textura com relevo e uma variante da abordagem *multi-thread* que processe as linhas e colunas da textura em paralelo.

Além disso, um estudo sobre a taxa de *warpings* por segundo (wps, ou seja, o número de operações de *warping* realizadas por segundo) em relação à taxa de quadros por segundo poderia permitir uma implementação mais eficiente (com menos operações de *warping*) do processo, caso a taxa de wps se confirme ser maior que a taxa de fps.

Finalmente, uma estratégia de implementação que utilize vários processadores físicos, ao invés do uso da tecnologia de *Hyper-Threading*, seria um trabalho interessante para se comparar com as abordagens aqui implementadas.

References

[1] Catmull, E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Dezembro 1974. Tese (Ph.D em Ciência da Computação). Department of Computer Science, University of Utha, Utha, 1974.

[2] Fonseca, F. M. A. *Texturas com Relevo utilizando Iluminação por Pixel e Processamento Paralelo*. Janeiro 2004. Dissertação de Mestrado. Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2004.

[3] Fujita, M; e Kanai, T. Hardware-Assited Relief Texture Mapping. In: European Association for Computer Graphics (EUROGRAPHICS). 2002. *Proceedings of the annual conference of the European Association for Computer Graphics* Saarbrücken, Germany, 2002.

[4] Marr, D. T.; *et al*. Hyper-Threading Technology Architecture and Microarchitecture. *Intel Technology Journal*. v. 6, n. 1, p. 4-152, fev. 2003.

[5] Mcmillan, L. *An Image-Based Approach to Three-Dimensional Computer Graphics*. Abril 1997. Tese (Ph.D em Ciência da Computação). Department of Computer Science, University of North Carolina, Chapel Hill, 1997.

[6] Akenine-Müller, T.; Haines, E. *Real-Time Rendering*. 2. ed. Massachusetts: A K Peters, 2002. 482 p.

[7] Oliveira, M. M. de. *Relief Texture Mapping*. Março 2000. Tese (Ph.D em Ciência da Computação). Department of Computer Science, University of North Carolina, Chapel Hill, 2000.

[8] Policarpo, F.; Oliveira, M. M.; Comba, J. L. D. *Real-time relief mapping on arbitrary polygonal surfaces*. Proceedings of the Symposium on Interactive 3D Graphics and Games, Washington, District of Columbia, 2005.