5
Resultados

Nos capítulos anteriores, apresentou-se o problema de mapeamento de textura com relevo bem como três abordagens para resolvê-lo: seqüencial, paralela e *multi-threaded*.

Na abordagem seqüencial, proposta por Oliveira [26], estudou-se quatro variantes de implementação onde o processo de amostragem é realizado através das equações de pré *warping* e o processo de reconstrução responsabiliza-se por interpolar *texels* não-consecutivos resultantes do *warping*.

Na abordagem paralela, proposta neste trabalho, 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*.

Na abordagem *multi-threaded*, também proposta neste trabalho, 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.

Neste capítulo, são apresentadas as estatísticas de tempo consumido dos algoritmos implementados. 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 *Ge Force FX* 5600 com 256Mb de memória de vídeo.

5.1
Medidas de Desempenho

Usualmente são utilizados dois tipos de medida de desempenho: *wall-clock time* e *tempo de CPU*.

Wall-clock time [28] é a latência para completar uma tarefa, incluindo acesso ao disco, acesso à memória, operações de entrada e saída, atividades do sistema operacional, entre outras. *Tempo de CPU* é o tempo em que um processo permaneceu na CPU para realização de suas tarefas.

Dessa forma, estas são as duas medidas adotadas nos experimentos realizados neste trabalho.

5.2
Tempo de CPU

Inicialmente, as quatro variantes de implementação da abordagem seqüencial são comparadas e a mais eficiente destas é utilizada nos experimentos posteriores.

Como este experimento é seletivo, ou seja, deseja-se selecionar o mais rápido algoritmo dentre um conjunto de algoritmos, adotou-se como medida de desempenho o tempo de CPU. Neste caso, tal medida demonstra-se mais justa que o *wall-clock time* uma vez que operações realizadas pelo sistema operacional que não fazem parte do processo verificado, não são contabilizadas.

Para realização do experimento, escolheu-se três conjuntos de amostras (Figura 5.1, no final do capítulo). A Tabela 5.1 apresenta alguns dados para as amostras utilizadas como, por exemplo, resolução (em *pixels*), número de *texels* com valor de profundidade inválido e as dimensões (em unidades da cena) do quadrilátero sobre o qual tal amostra é mapeada.

A Tabela 5.2 apresenta os resultados do processo de medição obtidos para cada variante de implementação. A variante 1 corresponde à abordagem unidimensional realizada em dois passos, enquanto que as variantes 2, 3 e 4 correspondem, respectivamente, às abordagens assimétrica, com compensação de deslocamento e intercalada.

Amostra	**Resolução**	***Texels* Inválidos**	**Dimensões**	
			x	y
1	256x256	3800 (5.80%)	107.33	101.68
2	256x256	3223 (4.92%)	279.63	246.62
3	256x256	30484 (46.51%)	40.52	67.31

Tabela 5.1: Amostras para realização dos experimentos. As Figuras 5.1(a) e 5.1(b), 5.1(c) e 5.1(d), 5.1(e) e 5.1(f); representam, respectivamente, as amostras 1, 2 e 3.

Algoritmo	**Amostra 1**		**Amostra 2**		**Amostra 3**	
	$\overline{CPU}$	*DP*	$\overline{CPU}$	*DP*	$\overline{CPU}$	*DP*
1	48.281	4.790	47.406	3.001	30.141	4.309
2	48.953	5.578	47.859	4.612	30.750	3.241
3	50.406	6.686	49.625	6.075	32.453	4.451
4	**41.953**	7.362	**40.859**	7.734	**26.594**	7.184

Tabela 5.2: Tempo de CPU para as quatro variantes de implementação da abordagem seqüencial, onde $\overline{CPU}$ significa o tempo médio de CPU medido em mili-segundos e *DP* é o desvio padrão em relação ao tempo médio de CPU.

Analisando-se a Tabela 5.2 é possível concluir que a variante de implementação mais eficiente é a variante 4. Este resultado já era esperado uma vez que tal variante realiza um número menor de operações que as demais, devido à intercalação entre os passos horizontal e vertical.

Outro resultado esperado, e comprovado por este experimento, foi um menor tempo médio de CPU para a amostra 3 em todas as variantes, pois, dentre todas as amostras esta é a que apresenta um maior número de *texels* cujo valor de profundidade é inválido e ainda é a amostra que possui o menor quadrilátero sobre o qual a mesma é mapeada (veja a Tabela 5.1).

Além disso, observando-se os valores dos desvios padrões obtidos, pode-se concluir que apesar da variante 4 ser a mais eficiente, esta é a menos estável de todas em relação ao tempo de execução consumido.

5.3

Quadros por Segundo

De acordo com a Seção 5.2, a variante 4 demonstrou-se mais rápida que as demais e, portanto, é utilizada no novo experimento. Tal experimento

trata-se de uma comparação entre as abordagens seqüencial, paralela e *multi-threaded*.

Como visto no Capítulo 4, é o estágio mais lento do *pipeline* que determina a velocidade de renderização, ou seja, a velocidade com que as imagens são atualizadas. Costuma-se expressar tal velocidade em quadros por segundo (fps, do inglês *frames per second*), isto é, o número de imagens renderizadas por segundo. Além disso, segundo Akenine-Möller & Haines [2], a taxa mínima aceitável para uma aplicação ser considerada de tempo real é de aproximadamente 15fps.

No contexto de aplicações gráficas em tempo real, este tipo de medição destina-se a verificar o quão interativa é uma aplicação. Neste caso, não faz sentido desconsiderar os atrasos provenientes de operações realizadas pelo sistema operacional durante o processo de renderização, uma vez que o usuário tem de lidar com tais atrasos quando utiliza uma aplicação na vida real. Por este motivo, adotou-se como medida de desempenho para este novo experimento o número de quadros por segundo baseado no *wall-clock time*.

A Tabela 5.3 apresenta os resultados deste processo de medição obtidos para cada uma das abordagens consideradas, levando-se em consideração um observador estático. As amostras 1, 2 e 3 são as mesmas consideradas como no caso da medição do tempo de CPU.

Abordagem	**Amostra 1**		**Amostra 2**		**Amostra 3**	
	$\overline{fps}$	DP	$\overline{fps}$	DP	$\overline{fps}$	DP
Seqüencial	24.18	0.86	24.93	1.54	38.24	1.86
Paralela	97.97	4.50	96.45	5.01	103.64	4.88
Multi-Threaded	33.14	6.02	31.09	7.67	34.51	8.21

Tabela 5.3: Média de quadros por segundo para as abordagens seqüencial, paralela e *multi-threaded* para um observador estático, onde $\overline{fps}$ significa número médio de quadros por segundo e DP é o desvio padrão em relação ao *fps* médio.

Analisando-se a Tabela 5.3 é possível concluir que, para uma câmera estática, 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. Como descrito no Capítulo 4, tanto a abordagem paralela quanto a abordagem *multi-threaded* foram projetadas para otimizar

o processamento empregado durante a etapa de pré *warping*. No entanto, 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 5.1) e, portanto, o processamento empregado durante a etapa de pré *warping* é inferior em relação às amostras 1 e 2. O segundo ponto se refere à abordagem *multi-threaded* que, como descrito no Capítulo 4, 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 (veja a Seção 4.3.2).

A Tabela 5.4 apresenta os resultados para um observador em movimento. Novamente, as amostras 1, 2 e 3 são as mesmas consideradas como no caso da medição do tempo de CPU.

Abordagem	**Amostra 1**		**Amostra 2**		**Amostra 3**	
	$\overline{fps}$	*DP*	$\overline{fps}$	*DP*	$\overline{fps}$	*DP*
Seqüencial	22.20	2.90	22.13	3.34	37.12	10.39
Paralela	70.14	28.86	66.91	35.04	110.86	55.99
Multi-Threaded	14.39	6.25	11.45	3.59	12.47	5.09

Tabela 5.4: Média de quadros por segundo para as abordagens seqüencial, paralela e *multi-threaded* para um observador em movimento, onde $\overline{fps}$ significa número médio de quadros por segundo e *DP* é o desvio padrão em relação ao *fps* médio.

Nos casos em que o ponto de vista é alterado com muita freqüência

há uma queda na taxa de quadros por segundo, uma vez que se requer um maior número de operações de *warping* para atualizar a imagem resultante de acordo com a nova visão. Analisando-se a Tabela 5.4 é possível notar um maior decaimento dos valores das medidas de *fps* para as duas últimas abordagens.

A abordagem *multi-threaded* apresentou um maior decaimento, pois, como antes, existem três processos para dois processadores, no entanto, aumentou-se o trabalho requerido para o processo principal da CPU.

Apesar do decaimento da taxa de *fps*, a abordagem paralela ainda apresenta os melhores resultados; no entanto, sua instabilidade aumentou. Tal instabilidade se deve à grande dispersão entre as taxas obtidas para tal abordagem, ou seja, ora a taxa de quadros por segundo aumenta muito (uma operação de *warping* está sendo executada e nenhuma atualização da textura na memória de vídeo é realizada), ora tal taxa diminui muito (uma operação de *warping* foi realizada e é necessário atualizar a textura residente na placa de vídeo).

5.4
Utilização de Texturas com Relevo em Jogos Eletrônicos

Jogos eletrônicos são provavelmente o tipo de aplicação em Computação Gráfica que mais demande alto realismo de imagens em tempo real. No entanto, quanto mais detalhada e complexa é uma cena, maior é a quantidade de polígonos requerida para representá-la.

A título de exemplo, a Tabela 5.5 apresenta algumas informações sobre os modelos poligonais utilizados para gerar os mapas de normal associados às amostras dos experimentos computacionais descritos anteriormente.

Amostra	**Vértices**	**Triângulos**	$\overline{fps}$
1	5808	11546	29
2	34377	67826	5
3	17881	35280	10

Tabela 5.5: Informações sobre os modelos poligonais que originaram os mapas de normal das amostras 1, 2 e 3.

As baixas taxas de quadros por segundo, observadas na Tabela 5.5, refletem a grande quantidade de vértices e triângulos requerida para re-

presentar os modelos poligonais das amostras 2 e 3. Uma vez que jogos eletrônicos exigem interação em tempo real, tais modelos, exibidos nesta taxa de quadros por segundo, tornam-se impraticáveis de serem utilizados neste tipo de aplicação. No entanto, a não utilização de modelos detalhados em jogos eletrônicos, pode prejudicar a sua qualidade visual.

Dessa forma, é preciso mesclar a eficiência computacional da renderização baseada em imagens com a qualidade visual garantida pela renderização baseada em polígonos. Observando-se os resultados obtidos através da realização dos experimentos computacionais apresentados neste capítulo, é possível identificar que ambos os benefícios convergem na técnica de mapeamento de textura com relevo.

O Algoritmo 25 apresenta uma maneira de renderizar objetos tridimensionais em jogos eletrônicos através de técnica de mapeamento de textura. Para uma renderização eficiente, o mapa de textura é mapeado como uma textura padrão para objetos extremamente distantes, e para objetos mais próximos a técnica de *bump mapping* é utilizada. Para objetos realmente próximos, os quadriláteros são texturizados através da técnica de mapeamento de textura com relevo.

```
algoritmo renderizaTextura(quad, texRel, cam)
    1 Se cam.posicao está distante de quad então
    2    mapeamentoConvencionalTextura(quad, texRel, cam);
    3 senão se cam.posicao está próximo de quad então
    4    bumpMapping(quad, texRel, cam);
    5 senão se cam.posicao está muito próximo de quad então
    6    reliefTextureMapping(quad, texRel, cam);
    7 fim_se
fim
```

Algoritmo 25: Renderização de objetos 3D em jogos eletrônicos através de técnicas de mapeamento de textura.

5.5 Discussão

Os resultados visuais obtidos pelas quatro variantes de implementação do algoritmo de mapeamento de textura com relevo, para as amostras da Figura 5.1, são indistinguíveis uns dos outros e, portanto, não requerem maiores comentários. Cabe salientar, entretanto, a grande diferença nas

imagens geradas com textura de relevo em relação às geradas com *bump-mapping*. Pode-se verificar nas Figuras 5.2, 5.3 e 5.4 a representação muito mais realista das profundidades, obtida com a técnica de mapeamento de textura com relevo.

Neste trabalho, o processo de amostragem e reconstrução de imagens é realizado a partir de uma única amostra (mapa de normal e mapa de textura) disponível do objeto a ser representado. Desse modo, dependendo do ponto de vista corrente, a amostra utilizada deixaria de ser válida devido à falta de informação necessária para gerar uma visualização correta de acordo com as configurações correntes da câmera. Para resolver este problema, Esteban Clua, em sua tese de doutorado [10], sugere a implementação de um sistema que permita gerar novas amostras de forma dinâmica. Para este fim, é necessário desenvolver uma métrica que seja capaz de inferir em que momento a amostra corrente está acumulando excessivos erros e, portanto, está prestes a ser substituída. Os primeiros trabalhos nesta área foram realizados por Schaufler [32, 33].

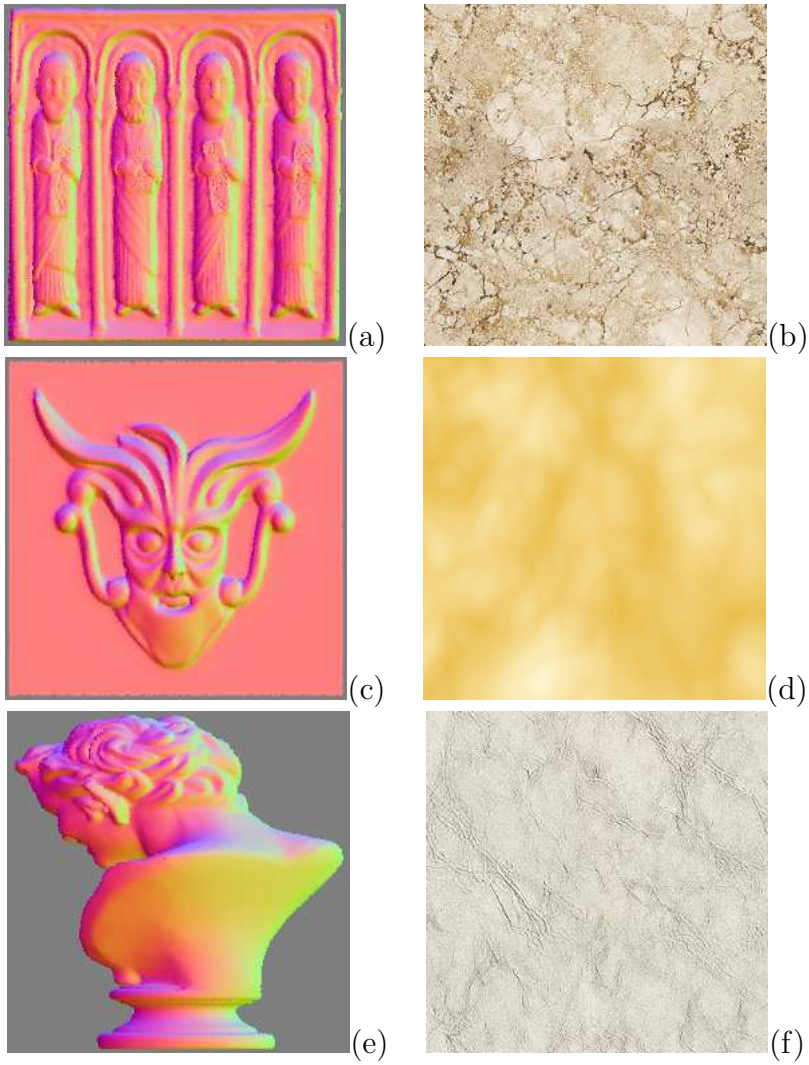

Figura 5.1: Texturas de entrada para os experimentos realizados. Na coluna da esquerda (imagens (a), (c) e (e)) estão os mapas de normais. Na coluna da direita (imagens (b), (d) e (f)) estão os mapas de cor. Todas as imagens possuem uma resolução de 256x256 *pixels*.

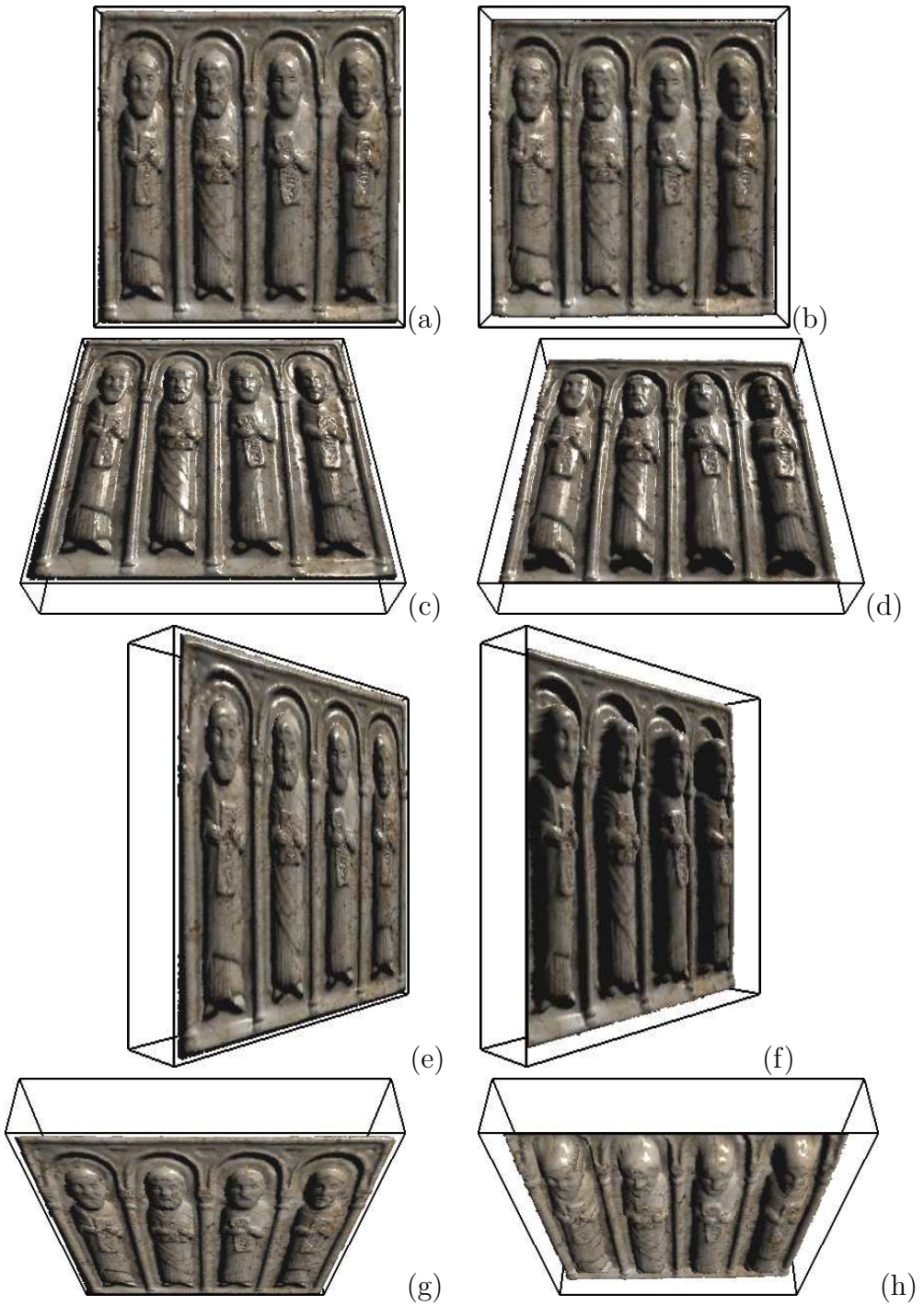

Figura 5.2: Resultados visuais obtidos através da execução da variante 1 a partir das texturas de entrada das Figuras 5.1(a) e 5.1(b). As imagens da esquerda foram geradas através da técnica de *bump mapping*, enquanto que as imagens da direita representam as mesmas vistas que as imagens da esquerda, porém, visualizadas através da técnica de mapeamento de textura com relevo.

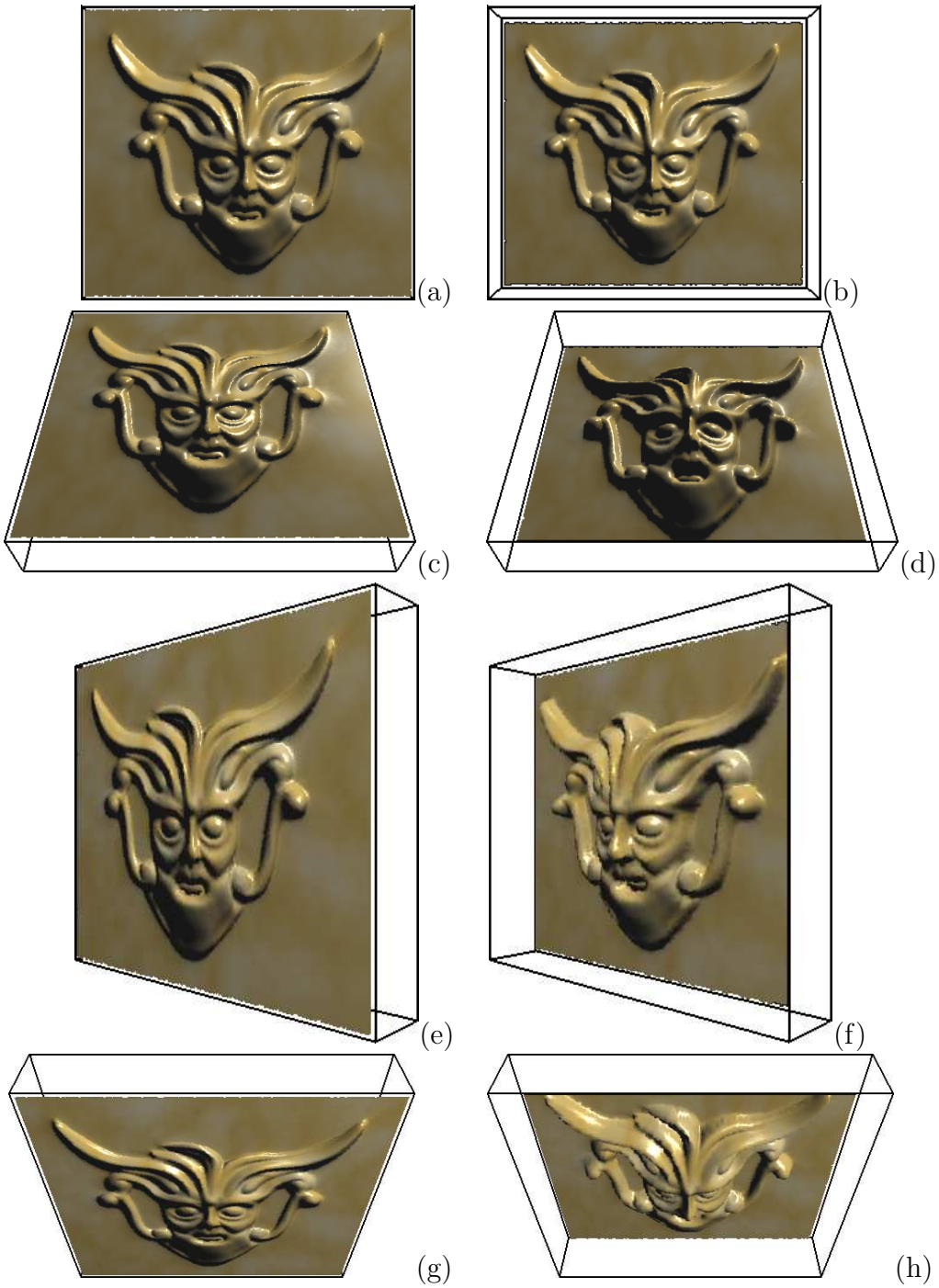

Figura 5.3: Resultados visuais obtidos através da execução da variante 2 a partir das texturas de entrada das Figuras 5.1(c) e 5.1(d). As imagens da esquerda foram geradas através da técnica de *bump mapping*, enquanto que as imagens da direita representam as mesmas vistas que as imagens da esquerda, porém, visualizadas através da técnica de mapeamento de textura com relevo.

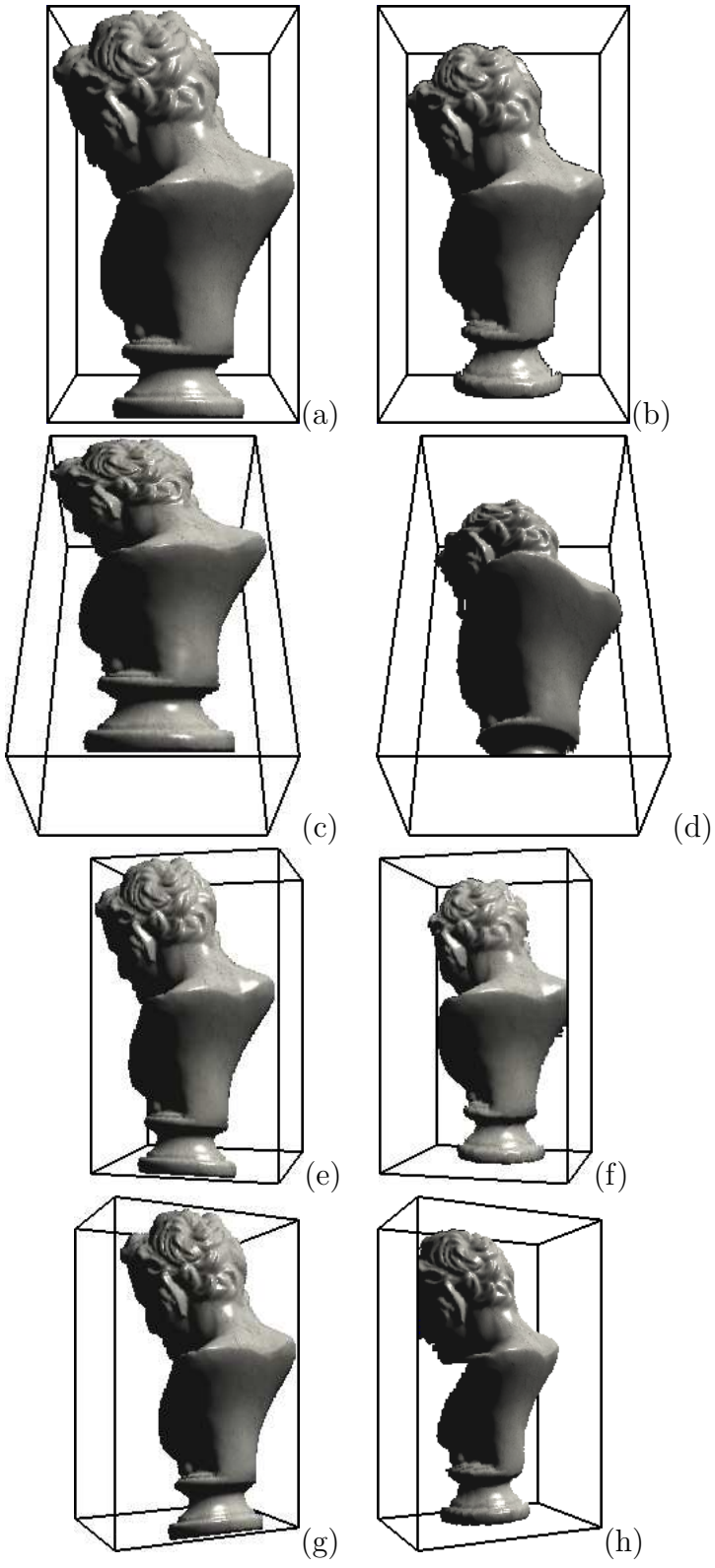

Figura 5.4: Resultados visuais obtidos através da execução da variante 3 a partir das texturas de entrada das Figuras 5.1(e) e 5.1(f). As imagens da esquerda foram geradas através da técnica de *bump mapping*, enquanto que as imagens da direita representam as mesmas vistas que as imagens da esquerda, porém, visualizadas através da técnica de mapeamento de textura com relevo.