Bruno Pereira Evangelista

Survey de técnicas para simulação de malhas detalhadas: Análise e comparação

Belo Horizonte

Novembro de 2006

Bruno Pereira Evangelista

Survey de técnicas para simulação de malhas detalhadas: Análise e comparação

Monografia apresentada à disciplina Trabalho de Diplomação, do departamento de Ciência da Computação do Instituto de Informática da Pontifícia Universidade Católica de Minas Gerais.

Orientador:
Marcelo Souza Nery

PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Belo Horizonte

Novembro de 2006

Bruno Pereira Evangelista

Trabalho de Diplomação apresentado ao departamento de Ciência da Computação do Instituto de Informática da Pontifícia Universidade Católica de Minas Gerais.

Prof. Marcelo de Souza Nery
Orientador

Sumário

Lista de Figuras

Lista de Tabelas

Abstract

Computer graphics's aplications, in special games and simulators, have tried to create high-quality visual virtual environments wich we can't realize if it's real or rendered. One of the features that determines the level of realism in a virtual environment is the level of details in objects's mesh present in that scene. In real world, objects's meshes must to be construct with a millions of triangles in order to represent precisily those objects. However, as much level of details raise, the gameplay tends to lower. Many techniques were proposed to solve that problem. In this work, we present a survey about the main techniques analysing and comparing features like behavior, visual effects treated, main benefits or not and the best scene environment to use the techniques. The techniques's comparing was made over the stills, taken from the images created with each technique, and over the performance. All the techniques's reports could be used to help the choose for the best technique for each scene environment and application.

Resumo

Aplicativos de computação gráfica, em especial jogos e simuladores, têm buscado criar ambientes virtuais com níveis de realismo tão altos que não possam ser diferenciados do mundo real. Um dos fatores que determina o nível de realismo de um ambiente virtual é o nível de detalhe da malha dos objetos presentes nesse ambiente. Malhas de objetos do mundo real, precisam de milhões de triângulos para serem representadas com precisão. No entanto, conforme o nível de detalhe das malhas aumenta a interatividade com o ambiente tende a diminuir. Para resolver esse problema várias técnicas foram propostas para simular malhas detalhadas, visando criar ambientes com alta qualidade visual e alto desempenho.

Neste trabalho é apresentada uma análise e comparação das principais técnicas existentes para simulação de malhas detalhadas. Na análise das técnicas é apresentado seu funcionamento, efeitos visuais tratados, principais vantagens e desvantagens de sua utilização e ambiente e situação considerada mais adequada para uso da mesma. A comparação das técnicas foi feita com base nas imagens geradas no desempenho obtido no uso de cada uma das técnicas. Todas as informações apresentadas sobre as técnicas poderão ser utilizadas na escolha da melhor técnica a ser utilizada para cada tipo de ambiente ou aplicação.

1 Introdução

1.1 Simulação de malhas detalhadas

Aplicativos de computação gráfica, em especial jogos e simuladores,têm buscado criar ambientes virtuais com níveis de realismo tão altos que não possam ser diferenciados do mundo real. A criação desses ambientes realistas só se tornou possível recentemente com a introdução dos *hardwares* gráficos programáveis. Os *hardwares* gráficos programáveis, além de possuírem um alto poder de processamento, permitem que seja alterada a maneira como os ambientes são renderizados. Alterando a forma como os ambientes virtuais são renderizados permite adicionar-se novos efeitos às cenas, que contribuem para a criação de cenários foto-realistas.

Um dos fatores que determina o nível de realismo de um ambiente virtual é o nível de detalhe da malha dos objetos presentes nesse ambiente. As malhas dos objetos geralmente são formadas por um conjunto de primitivas do tipo triângulo [1]. Malhas de objetos do mundo real, extraídas a partir de *scanners* tridimensionais, precisam de milhões de triângulos para serem representadas com precisão. No entanto, conforme o nível de detalhe das malhas aumenta, a interatividade com o ambiente tende a diminuir.

Uma das técnicas muito utilizada para adicionar detalhes à malha dos objetos é o mapeamento de textura (COOK; TORRANCE, 1982). A partir dessa técnica, pode-se mapear uma imagem para a superfície de um objeto, substituindo a cor padrão do objeto ou misturando-se com a mesma. As imagens utilizadas no mapeamento de texturas geralmente são imagens extraídas do mundo real, visando-se representar com precisão os objetos. No entanto, quando esses objetos são visualizados em ângulos agudos[2], é possível perceber que a superfície real é plana, fazendo com que o mesmo pareça artificial.

Uma maneira mais eficaz para adicionar-se detalhes à malha dos objetos é alterar a forma como a cor é calculada em cada ponto da superfície da malha. Para calcular a cor de cada

[1] Isso ocorre devido à facilidade dos *hardwares* em realizar cálculos com triângulos

[2] Ângulos agudos são ângulos inferiores a 90 graus, entre a câmera e a superfície.

ponto da superfície utiliza-se a equação de renderização (KAJIYA, 1986). No entanto, devido à complexidade dessa equação, a mesma ainda não pode ser calculada com eficiência nos dispositivos gráficos atuais. A equação de iluminação de Phong (BURGER; GILLIES, 1989) é uma especificação da equação de renderização, que pode ser utilizada para calcular a cor de cada ponto da superfície em tempo real. Essa equação é utilizada nas principais APIs gráficas, como DirectX e OpenGL. A partir da equação de Phong pode-se alterar a cor de cada ponto na superfície de uma malha alterando-se a normal daquele ponto. Com isso, pode-se gerar efeitos mais realistas sem alterar a malha dos objetos.

Utilizando a equação de iluminação de Phong como base, várias técnicas foram propostas para simular malhas detalhas, com alto número de triângulos, utilizando malhas menos detalhadas, com um menor número de triângulos (BLINN, 1978; KANEKO et al., 2001; WELSH, 2004; OLIVEIRA; BISHOP; MCALLISTER, 2000; POLICARPO; OLIVEIRA; COMBA, 2005; TATARCHUK, 2006; RISSER; SHAH; PATTANAIK, 2006; DONNELLY, 2005; DUMMER, 2006). O trabalho proposto por Wang (WANG et al., 2003b) utiliza mapas multidimensionais contendo informações pré-processadas sobre iluminação da malha do objeto, essas informações permitem representar com maior precisão a silhueta das malhas renderizadas.

De maneira geral, o objetivo de todas essas técnicas é apresentar uma solução para a renderização de malhas detalhadas que tratem com precisão os efeitos visuais percebidos na renderização das geometrias reais como silhuetas, *motion parallax*, *self-occlusion* e *self-shadow*.

Quando as pessoas se movem, elas percebem os objetos mais próximo se moverem rapidamente, em comparação com os objetos mais distantes; esse efeito é denominado *motion parallax* (HOWARD; ROGERS, 1995). *Self-occlusion* ocorre quando partes mais próximas de uma superfície ocultam partes mais distantes da mesma. Este efeito geralmente é tratado pelas APIs gráficas, utilizando-se o algoritmo de *Z-Buffer*, mas quando a superfície renderizada é simulada é preciso utilizar outras técnicas para tratar este efeito. *Self-shadow* ocorre quando partes de uma superfície geram sombras sobre ela mesma. Como as superfícies renderizadas não são reais, os algoritmos convencionais utilizados para tratar sombras (HASENFRATZ et al., 2003) não podem ser utilizados. Por fim, a silhueta dos objetos determina o contorno externo dos mesmos. Como a simulação de uma malha detalhada é feita com base na superfície da malha, a imagem final gerada apresenta o contorno externo da superfície renderizada. Apenas os pontos gerados na renderização da superfície podem ser alterados, não sendo possível desenhar fora da superfície original renderizada.

Tratando de forma precisa esses efeitos, é possível aumentar o realismo das superfícies de modo que as mesmas não possam ser diferenciadas da malha real do objeto, que necessitaria de

milhões de triângulos para ser representada com precisão.

1.2 Objetivos

O objetivo deste trabalho é apresentar uma análise das principais técnicas existentes para simulação de malhas detalhadas e comparar a qualidade visual das imagens geradas e o desempenho obtido a partir da utilização de cada uma das técnicas.

O objetivo da análise realizada é apresentar o funcionamento da técnica, os efeitos visuais tratados pela mesma, as principais vantagens e desvantagens de sua utilização e ambiente e/ou situação considerada mais adequada para uso da mesma.

O objetivo da comparação é apresentar as principais diferenças entre as imagens geradas e o desempenho obtido para cada uma das técnicas implementadas [3]. Os critérios utilizados para comparação da qualidade visual e desempenho das técnicas serão apresentados na Seção 4.1.

Todas essas informações poderão ser utilizadas na escolha da melhor técnica a ser utilizada para um determinado tipo de aplicativo, ou para um determinado cenário dentro de um aplicativo. Este trabalho também pode ser utilizado como referência para futuras pesquisas na área, por explicitar as principais deficiências de cada técnica e os principais efeitos que ainda não são tratados pelas mesmas.

1.3 Motivação

Atualmente existe uma grande variedade de técnicas para simulação de malhas detalhadas (BLINN, 1978; KANEKO et al., 2001; WELSH, 2004; WANG et al., 2003b; OLIVEIRA; BISHOP; MCALLISTER, 2000; POLICARPO; OLIVEIRA; COMBA, 2005; TATARCHUK, 2006; RISSER; SHAH; PATTANAIK, 2006; DONNELLY, 2005; DUMMER, 2006). Cada uma dessas técnicas trata um conjunto específico de efeitos visuais, que estão diretamente relacionados à qualidade visual das imagens geradas, e apresentam uma diferente relação entre qualidade e desempenho. No entanto, nenhuma dessas técnicas apresenta uma solução genérica, que possa ser utilizada para a renderização de qualquer tipo de malha, a partir de qualquer ponto de vista[4] e apresente uma boa relação de qualidade/desempenho. O motivo pelo qual isto ocorre é que, para tratar cada um dos efeitos visuais listados (silhuetas, *motion parallax*, *self-occlusion* e *self-*

[3] A técnica de View-dependent displacement mapping (WANG et al., 2003b) não foi implementada devido a falta de informações no artigo que inviabilizaram a implementação da mesma.

[4] Ponto de vista é a posição e ângulo pelo qual a cena é observada.

shadow), geralmente demanda-se um maior tempo de processamento na renderização da malha. No entanto, esses efeitos podem ser imperceptíveis para certos pontos de vista. Por exemplo, tratar o efeito de *motion parallax* para uma malha pode ser vantajoso quando a mesma for observada a uma pequena distância; no entanto, tratar esse efeito para todas as malhas de uma cena pode reduzir drasticamente o desempenho do ambiente. Além disso, o observador possivelmente não perceberá este efeito para as malhas observadas a uma grande distância e em que o ângulo de observação entre o olho e a superfície da malha for próximo de 90^o.

Outro problema encontrado é que grande parte dos trabalhos não apresentam comparações entre a técnica proposta e as demais técnicas existentes, ou mesmo apresentam comparações isoladas, relacionadas a qualidade visual ou desempenho. Mesmo técnicas recentes como Cone Step Mapping (DUMMER, 2006) e Sphere Tracing (DONNELLY, 2005), que se propõem a otimizar as técnicas de simulação de malhas detalhadas baseadas em traçado de raios, não comparam o desempenho obtido com as demais técnicas existentes. Além disso, as deficiências existentes em cada uma das técnicas, não são abordadas em todos os trabalhos. Devido a esses problemas, torna-se de grande importância apresentar uma análise e comparação entre as técnicas existentes para simulação de malhas detalhadas.

1.4 Contribuições

Dentre as contribuições desse trabalho as principais são:

- Uma análise das principais técnicas para simulação de malhas detalhadas;
- Apresentação das principais vantagens e desvantages de cada uma das técnicas analisadas;
- Apresentação dos efeitos visuais tradados pelas técnicas;
- Identificação de um ambiente para uso eficaz[5] de cada uma das técnicas;
- Comparação entre as imagens geradas e qualidade visual obtida a partir das técnicas;
- Comparação de desempenho entre as técnicas;
- Código fonte da implementação das técnicas.

[5]Que possibilite uma melhor relação entre qualidade visual e desempenho.

1.5 Divisão do trabalho

A divisão do trabalho é apresentada a seguir.

Na Seção 2 são apresentados os primeiros trabalhos relacionados à renderização de superfícies detalhadas.

Na Seção 3 são apresentadas e analisadas as principais técnicas existentes para a simulação de malhas detalhadas.

Na Seção 4 é apresentada uma comparação entre qualidade visual e desempenho obtidos a partir das técnicas para simulação de malhas detalhadas implementadas.

Na Seção 5 é apresentada uma conclusão e trabalhos futuros.

Por fim, no Apêndice B é apresentado o código fonte da implementação das técnicas para simulação de malhas detalhadas.

2 *Trabalhos relacionados*

A renderização de malhas detalhadas é um dos fatores que determina o nível de realismo de um ambiente virtual. Malhas detalhadas podem ser definidas como malhas observadas do mundo real, compostas por milhares de micro e macro deformações. Essas malhas precisam de milhões de primitivas (usualmente triângulos) para serem representadas de forma precisa em computação gráfica.

A técnica de mapeamento de texturas (COOK; TORRANCE, 1982) permite que uma imagem seja mapeada para a superfície de uma malha. Com isso, pode-se extrair uma imagem do mundo real, como uma foto de uma parede de tijolos, e mapear a mesma para a superfície de uma malha. Quando a malha for renderizada, a imagem da parede de tijolos será desenhada na superfície da mesma, fazendo com que a superfície da malha possa ser identificada como uma parede real de tijolos. Isso aumenta o realismo das superfícies renderizadas, pois as mesmas podem ser identificadas com superfícies do mundo real. O problema do mapeamento de texturas é que o mesmo não modifica a forma como a iluminação da superfície é calculada, com isso, pode-se perceber que a superfície é plana, observando-a de diferentes posições. Apesar do mapeamento de texturas trazer a ilusão de que as superfícies do mundo real estão presentes no ambiente virtual, o observador consegue perceber com facilidade que as superfícies não são reais.

Displacement mapping (COOK, 1984) é uma das técnicas para geração de malhas extremamente detalhadas que podem representar com grande precisão objetos do mundo real. Esta técnica aplica sucessivas subdivisões na malha dos objetos, aumentando drasticamente o número de vértices e triângulos do mesmo. Em seguida, os vértices do objeto são modificados permitindo a representação de micro detalhes na malha do objeto. Algumas ferramentas como Render Man (APODACA; GRITZ, 2000) utilizam um número de subdivisões tão grande que um *pixel* da imagem pode conter vários triângulos. As imagens geradas com a técnica de Displacement mapping possuem uma alta qualidade visual e, por trabalharem com a malha real dos objetos, tratam com precisão os efeitos de *motion parallax*, *self-occlusion* e silhuetas. Além disso, técnicas convencionais para geração de sombras (WOO; POULIN; FOURNIER, 1990)

podem ser utilizadas em conjunto com a técnica de Displacement mapping. A principal desvantagem dessa técnica é o custo de memória necessário para armazenar as malhas, compostas por milhões de triângulos, e o custo de processamento necessário para processar e renderizar essas malhas em tempo real. Isto torna essa técnica inviável para ser utilizada em aplicações de tempo real nos *hardwares* gráficos atuais.

Bump Mapping (BLINN, 1978) foi uma técnica proposta para adicionar detalhes às malhas dos objetos. Diferente da técnica de Displacemente mapping (COOK; TORRANCE, 1982), essa técnica não modifica a malha dos objetos, mas consegue adicionar detalhes na renderização das malhas que não podem ser obtidos utilizando apenas o mapeamento de texturas convencional. Entretanto, comparado com a técnica de Displacement mapping (COOK, 1984), essa técnica apresenta uma qualidade visual muito reduzida, não tratando os efeitos de silhuetas, *motion parallax*, *self-occlusion* e *self-shadows*.

Horizon mapping (SLOAN; COHEN, June 2000) permite a geração de sombras nas malhas renderizadas com a técnica de Bump mapping (BLINN, 1978). As técnicas convencionais de geração de sombras (WOO; POULIN; FOURNIER, 1990) foram descritas para aplicação sobre geometrias reais, não para geometrias simuladas a partir da técnicas como Bump mapping. Uma vantagem dessa técnica é que as sombras que a superfície pode gerar são pré-calculadas em vários mapas, denominados mapas de horizonte. Com isso, é preciso armazenar 8 texturas com informações sobre as sombras da superfície, mas a partir desses mapas, pode-se calcular rapidamente as sombras na superfície. Outra técnica para geração de sombras que pode ser utilizada foi apresentado por Heidrich (HEIDRICH et al., July 2000), que assim como a técnica de Horizon mapping, também utiliza mapas com informações pré-calculadas sobre visibilidade nas malhas para gerar as sombras.

Outros algoritmos foram propostos com o propósito de renderizar malhas detalhas utilizando malhas menos detalhadas e sem alterar sua geometria original (BLINN, 1978; KANEKO et al., 2001; WELSH, 2004; OLIVEIRA; BISHOP; MCALLISTER, 2000; POLICARPO; OLIVEIRA; COMBA, 2005; TATARCHUK, 2006; RISSER; SHAH; PATTANAIK, 2006; DONNELLY, 2005; DUMMER, 2006). No entanto, nenhum desses algoritmos apresenta uma solução genérica para renderização de malhas detalhadas. Cada um desses algoritmos trata um conjunto específico de efeitos visuais e apresenta uma diferente relação entre qualidade visual e desempenho para diferentes pontos de vista. Por esse motivo, pode ser necessário utilizar uma técnica diferente para a renderização de cada malha dentro de um mesmo ambiente. Mesmo para a renderização de uma única malha em separado, pode-se obter uma melhor relação entre qualidade e desempenho aplicando diferentes técnicas na renderização de diferentes partes da malha.

Becker (BECKER; MAX,) apresenta uma técnica para transição entre diferentes técnicas de renderização de malhas detalhadas. Esta técnica visa obter uma melhor relação entre qualidade visual e desempenho na renderização de malhas detalhadas, utilizando diferentes técnicas na renderização de diferentes partes da malha. Para isso utiliza-se variações das técnicas de Displacement mappping (COOK, 1984), Bump mapping (PEERCY; AIREY; CABRAL, 1997) e BRDF (BEN-ARTZI; OVERBECK; RAMAMOORTHI, 2006). A técnica de Displacement mapping é utilizada na renderização de partes da malha observadas a uma pequena distância, a técnica de Bump mapping é utilizada na renderização de partes da malha observadas a uma distância mediana e a equação de iluminação de Phong (BURGER; GILLIES, 1989) é utilizada na renderização das demais partes da malha. Além da distância o ângulo pelo qual a malha é observada também é utilizado na escolha da melhor técnica a ser utilizada. Essa técnica também realiza uma transição suave entre as técnicas utilizadas na renderização da malha, de modo que a transição entre as diferentes técnicas não possa ser percebida.

Tatarchuk (TATARCHUK, 2006) apresenta em seu trabalho um algoritmo de LOD (Nível de detalhe) que pode ser utilizado para a escolha da melhor técnica a ser utilizada na renderização de diferentes partes de um mesmo objeto. O algoritmo proposto funciona na GPU e realiza o cálculo do nível de *mip-mapping* de cada *pixel* renderizado da malha do objeto. O valor de *mip-mapping* calculado para cada parte do objeto é utilizado para escolha da melhor técnica a ser utilizada na renderização daquela parte do objeto.

3 *Técnicas para simulação de detalhes*

3.1 Divisão das técnicas

Neste trabalho foi proposta uma divisão das técnicas para simulação de malhas detalhadas, com base nas características de cada técnica e visando um melhor entendimento das mesmas. As técnicas foram divididas em quatro grupos:

1. Técnica original;
2. Técnicas baseadas em heurísticas;
3. Técnicas baseadas em traçado de raios;
4. Técnicas pré-calculadas.

No primeiro grupo, técnica original, é apresentada a técnica original para simulação de malhas detalhadas proposta por Blinn (BLINN, 1978) e uma extensão dessa técnica que visa otimizar a mesma, além da tirar proveito dos *hardwares* gráficos.

No grupo de técnicas baseadas em heurísticas, são apresentadas duas técnicas para simulação de malhas detalhadas a partir de heurísticas, que permitem aumentar o número de efeitos tratados pelas técnicas. No grupo de técnicas baseadas em traçado de raios, são apresentadas as técnicas mais recentes baseadas no algoritmo de traçado de raios. Essas técnicas permitem renderizar malhas detalhadas com precisão, obtendo uma alta qualidade visual. Por fim, no grupo técnicas pré-calculadas, são apresentadas técnicas que utilizam informações pré-calculadas sobre as superfícies, visando gerar imagens de alta qualidade com alto desempenho.

3.2 Metodologia de análise

Inicialmente será apresentada uma análise detalhada de cada uma das técnicas, observando-se:

- Funcionamento da técnica;
- Efeitos visuais tratados;
- Vantagens e desvantagens das técnicas;
- Ambiente e/ou ponto de vista considerado mais adequado para uso da mesma.

Para as técnicas analisadas que são comparadas na Seção 4, será apresentado um algoritmo em alto nível ilustrando o funcionamento da mesma. As APIs gráficas como DierectX e OpenGL utilizam esse algoritmo para o processamento de *pixels* da superfície das malhas renderizadas. As entradas dos algoritmos apresentados são fornecidas pelas aplicações gráficas que utilizam os mesmos, através de parâmetros definidos pelo usuário ou através de informações extraídas das malhas renderizadas. Alguns parâmetros de entrada do processamento de *pixels* são passados através do processamento de vértices utilizado, esses parâmetros são modificados para cada vértices da malha e interpolados para cada *pixel* da mesma. O Apêndice A é uma referência a todas as funções utilizadas nos algoritmos em alto nível apresentados.

O código fonte de todas as técnicas para simulação de malhas detalhadas implementadas é apresentado no Apêndice B. Para a implementação das técnicas foi utilizada a linguagem de *shaders* HLSL. Todas as técnicas, exceto a técnica de Parallax Occlusion Mapping, foram implementadas utilizando o modelo de shaders (*shader model*) 2.0. A técnica de Parallax Occlusion Mapping foi implementada utilizando o modelo de shaders 3.0. Versões mais recentes do modelo de shaders possuem um maior número de recursos, como desvios dinâmicos, mas geralmente apresentam desempenho inferior.

3.3 Técnica original

3.3.1 Bump Mapping

Em 1978, Blinn (BLINN, 1978) observou que a renderização de superfícies tridimensionais atingiu um alto nível de realismo. No entanto, as imagens geradas ainda possuíam uma aparência artificial, devido à renderização de superfícies lisas, sem a presença de micro ou macro deformações. Baseado nesse problema, ele propôs um algoritmo para perturbar[1] as normais das superfícies, buscando gerar superfícies não lisas, com um maior grau de realismo.

[1] Alteração de um fator Δ feita sobre as normais da superfície.

Em seu trabalho, Blinn utiliza uma aproximação da equação da renderização para renderizar as superfícies[2]. Para isso, considera-se que as superfícies renderizadas são isotrópicas e desconsidera-se a posição de seus vértices. A partir dessas considerações, a luz refletida em cada ponto da superfície é dependente apenas da normal da superfície naquele ponto, além das fontes de luz da cena. Portanto, a cor final de cada ponto da superfície será igual à luz refletida naquele ponto, considerando-se que as superfícies não emitem luz.

Para utilização do algoritmo de Bump Mapping, é necessário calcular dois vetores tangentes à superfície renderizada, denominados tangente e binormal. Esses vetores têm sua direção orientada à superfície da malha renderizada e são calculados através da derivação parcial dos pontos da superfície em relação a U e V, onde U e V são vetores paralelos à superfície, utilizados para o mapeamento de texturas (COOK; TORRANCE, 1982). A partir da tangente e binormal, pode-se calcular a normal da superfície através do produto vetorial entre a tangente e a binormal. O conjunto dos vetores tangente, binormal e normal, para cada ponto da superfície, forma um espaço denominado espaço da tangente. Este espaço é utilizado para o cálculo de iluminação de cada ponto da superfície, o que permite a renderização de qualquer superfície utilizando a técnica de Bump Mapping, não apenas superfícies planas. Para isso, é necessário que todos os objetos da cena (câmeras, luzes e outros) sejam transformados para o espaço da tangente. As equações para cálculo da tangente, binormal e normal são apresentados em 3.1, 3.2 e 3.3. A Figura 3.1 exibe a tangente, binormal e normal para um ponto arbitrário da superfície de uma esfera e um toróide. Os vetores tangente, binormal e normal de cada ponto da superfície definem o espaço da tangente.

$$Tangente = \frac{\partial P}{\partial u} \tag{3.1}$$

$$Binormal = \frac{\partial P}{\partial v} \tag{3.2}$$

$$Normal = Tangent \times Binormal \tag{3.3}$$

Para perturbar a normal da superfície, Blinn utiliza funções matemáticas com padrões de perturbações nas tangentes e binormais, e calcula as normais da superfície a partir das novas tangentes e binormais. Ao aplicar-se perturbações na tangente e binormal de um ponto da superfície, modifica-se a cor final daquele ponto na renederização. Segundo Blinn, o tempo

[2] Apesar disso não estar explicitado em seu trabalho, isso pode ser observado pois apenas a normal da superfície é utilizada no cálculo da cor de cada ponto da superfície.

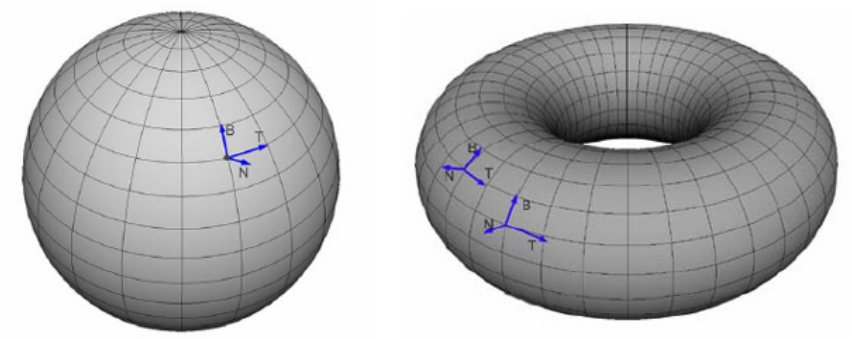


Figura 3.1: Visualização da tangente, binormal e normal de dois pontos arbitrários na superfície de dois sólidos do tipo esfera e toróide (LENGYEL, November 2003).

necessário para renderizar uma superfície utilizando a técnica de Bump Mapping é cerca de duas vezes superior ao tempo necessário para renderizar a mesma utilizando apenas uma textura. A Figura 3.2 exibe a aplicação de uma função de perturbação das normais de uma superfície lisa.

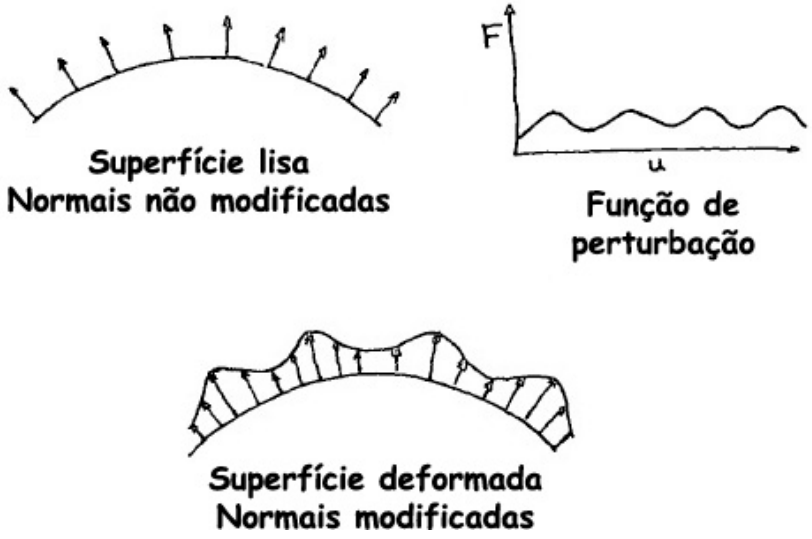


Figura 3.2: Aplicação de uma função de perturbação das normais de uma superfície lisa (BLINN, 1978).

3.3.2 Bump Mapping explorando hardware gráfico

Peercy (PEERCY; AIREY; CABRAL, 1997) propõem uma nova implementação do algoritmo de Bump Mapping para tirar proveito dos *hardwares* gráficos, dividindo o mesmo em operações relacionadas a vértices e *pixels*, e otimizar o processamento realizado pela técnica original. Uma das otimizações proposta por Peercy é calcular as tangentes, binormais e normais para cada vértice da superfície, e interpolar esses vetores para cada *pixel* na renderização da superfície. Antes, esses vetores eram calculados para cada *pixel* na renderização da superfície,

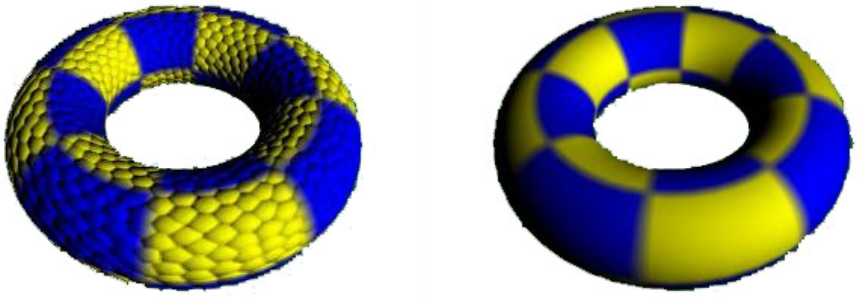


Figura 3.3: Aplicação do Bump Mapping. Toróide utilizando Bump Mapping e textura difusa (esquerda), toróide utilizando apenas textura difusa (direita). Ambos os sólidos possuem o mesmo número de triângulos em sua malha.

o que demandava um maior tempo de processamento, pois o número de *pixels* utilizados na renderização das malhas é geralmente muito superior ao número de vértices da mesma. Outra modificação do algoritmo foi utilizar um mapa de normais do objeto (EVANGELISTA et al., 2005). A Figura 3.4 ilustra um mapa de normais que pode ser utilizado para simular uma parede de pedras.

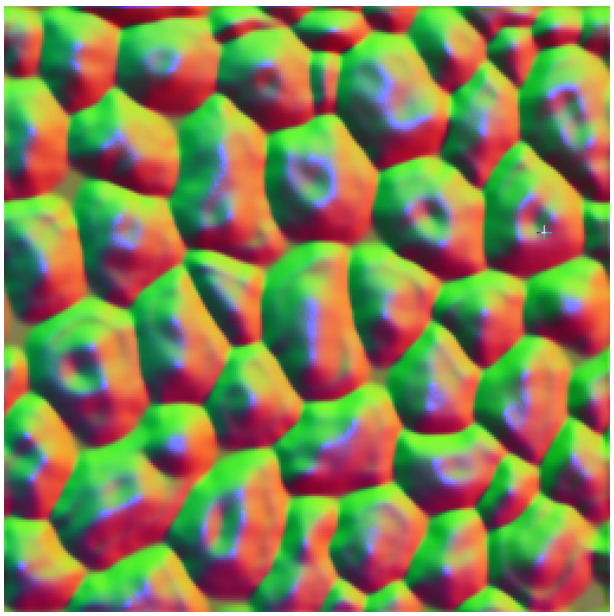


Figura 3.4: Mapa de normais com padrões de pedras. As cores Red Greep e Blue representam respectivamente as normais da superfície nos eixos X, Y e Z.

O mapa de normais pode, por exemplo, ser extraído de uma versão mais detalhada do modelo e utilizado na renderização de uma versão menos detalhada do mesmo. Assim como no algoritmo utilizado por Blinn (BLINN, 1978), todos os cálculos do algoritmo são realizados no espaço da tangente. Para isso, é necessário converter a coordenada de todos os objetos da cena utilizados no cálculo de iluminação, como câmeras e luzes, para esse espaço de coordenadas.

O algoritmo em alto nível 3.3.2 ilustra o funcionamento da técnica de Bump Mapping proposta por Percy (PEERCY; AIREY; CABRAL, 1997). A Figura 3.3 ilustra um toróide renderizado com e sem a aplicação da técnica de Bump Mapping.

As principais vantagens dessa técnica são:

- Fácil de ser implementada e integrada a *pipelines* de renderização;
- Implementação na GPU, utilizando *shader model* 1.0;
- Baixo tempo de processamento;
- Baixo consumo de memória, uma textura contendo as normais da superfície.

As principais desvantagens dessa técnica são:

- Não consegue representar grandes deformações nas superfícies;
- Não trata os efeitos de *motion parallax*, *self-occlusion*, *self-shadow* e silhuetas;
- A superfície aparenta ser plana quando observada de perto ou em ângulos próximos de 90 graus[3].

Algorithm 1 Algoritmo de Bump Mapping (PEERCY; AIREY; CABRAL, 1997) utilizado no processamento de *pixels* da superfície.

$tex1 \leftarrow Textura2D_{(cor\ difusa)}$
$tex2 \leftarrow Textura2D_{(mapa\ normais)}$

for each $Pixel(x,y)$ **do**
  $v \leftarrow \vec{V}_{visao}$
  $l \leftarrow \vec{V}_{iluminacao}$
  $uv \leftarrow coordenada\ de\ textura$

  $color \leftarrow readTexture2D(tex1, uv)$
  $n \leftarrow readTexture2D(tex2, uv)$
  $output(x,y) \leftarrow phongEquation(n, l, v, color)$
end for

Em resumo, a técnica de Bump Mapping é uma técnica fácil de ser implementada e integrada a *pipelines* de renderização, e que requer baixo processamento e memória. A técnica consegue simular malhas detalhadas, mas a qualidade visual obtida com a técnica é baixa, e a superfície parece plana quando observada a ângulos próximos de 90 graus. No entanto, é

[3] O ângulo considerado é formado entre o vetor invertido de visão e a normal da superfície.

possível obter uma maior qualidade visual na renderização de malhas observadas a uma grande distância ou observadas a ângulos próximos de 0 graus (visão quase perpendicular à superfície).

Um ambiente para uso eficaz dessa técnica são aplicações em tempo real, pois uma das principais vantagens da mesma é poder ser utilizada na renderização de milhares de objetos em tempo real, nos *hardwares* gráficos atuais. Essa técnica apresenta uma melhor relação entre qualidade visual e desempenho na renderização de objetos observados a grandes distâncias e e com ângulos de observação próximos de 90 graus.

3.4 Técnicas baseadas em heurísticas

3.4.1 Parallax Mapping

A técnica de Bump Mapping, apesar de proporcionar uma renderização mais realista possui várias limitações, como citado anteriormente. Um dos principais efeitos necessários para proporcionar uma renderização realista é o efeito de *motion parallax*. Em 2001, Kaneko (KANEKO et al., 2001) apresenta uma técnica denominada Parallax Mapping, que pode ser utilizada para simulação de malhas detalhadas, tratando os efeitos de *motion parallax* e silhueta das superfícies. Para tratar o efeito de *motion parallax*, essa técnica utiliza um mapa de curva de nível que representa a profundidade da superfície. Este mapa pode ser quantizado e armazenado em uma textura com 8 bits. A Figura 3.5 ilustra o mapa de profundidade de uma parede com pedras.

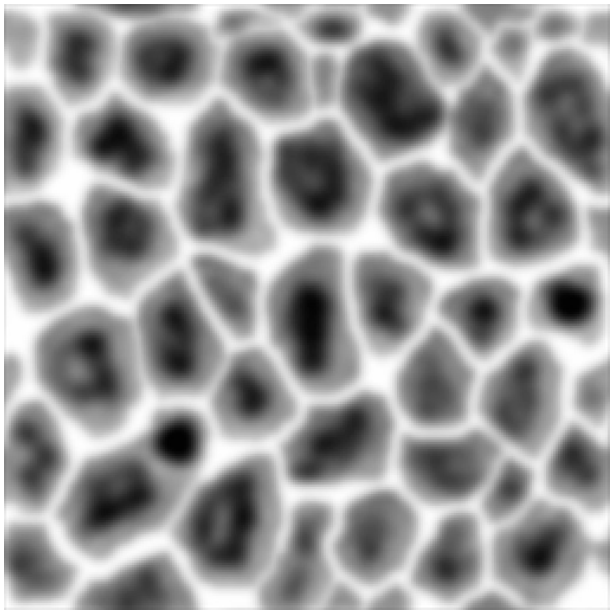


Figura 3.5: Mapa de profundidade com padrões de pedras. Cores mais escuras representam partes mais extrusas da superfície.

Utilizando-se o mapa de profundidade da superfície é possível descobrir o ponto correto que é observado para cada ponto na superfície da malha. A distância entre esses dois pontos, nos eixos U e V, representa o deslocamento necessário na coordenada de textura para aplicar o correto mapeamento de textura ao ponto inicial. Com isso, as texturas contendo as normais e a cor difusa da superfície são mapeadas corretamente. A Figura 3.6 exibe o ponto inicial P_1, que é obtido a partir das coordenadas de textura iniciais do ponto observado, e P_2, o ponto correto que seria observado em superfícies reais. Sem o uso do mapa de profundidade da superfície não seria possível calcular a correta posição que é observada na superfície.

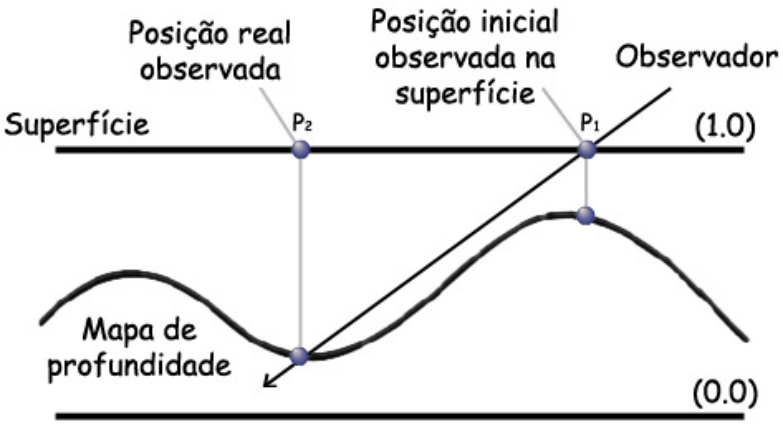


Figura 3.6: Renderização de uma superfície utilizando mapa de profundidade. A posição inicial P_1 é obtida a partir da coordenada de textura inicial do ponto, esta posição é utilizada no algoritmo de Bump Mapping. A posição correta observada é representada pela posição P_2.

A técnica de Parallax Mapping propõe um algoritmo que desloca a coordenada de textura acessada, visando realizar um mapeamento correto da textura. Para isso é utilizada uma heurística para calcular o correto deslocamento de textura necessário para cada ponto da superfície. As novas coordenadas de textura podem ser obtidas através das equações 3.4 e 3.4. Onde u e v são as coordenadas de textura iniciais para um ponto da superfície, θ_u e θ_v são os ângulos entre a normal e o vetor visão nos eixos u e v e $depth(u,v)$ é uma função que retorna a profundidade para cada ponto na superfície. Essas equações, no entanto, são apenas uma heurística para o cálculo da correta coordenada de textura a ser utilizada. Utilizando mapas de profundidades de alta freqüência, ou visualizando a malha a ângulos próximos de 90 graus, faz com que a técnica gere imagens com grandes distorções e uma baixa qualidade visual. A Figura 3.7 ilustra o processo de cálculo da nova coordenada de textura através do mapa de profundidade da superfície. Onde e é o vetor de visão, n a normal da superfície, $depth(\ldots)$ uma função que retorna a profundidade para qualquer ponto da superfície, *A"* é o ponto inicial observado na superfície (pelo vetor de visão) e *A'* o ponto correto observado, de acordo com o mapa de profundidade.

$$u' = u + \tan\theta_u \times depth(u,v) \tag{3.4}$$

$$v' = v + \tan\theta_v \times depth(u,v) \tag{3.5}$$

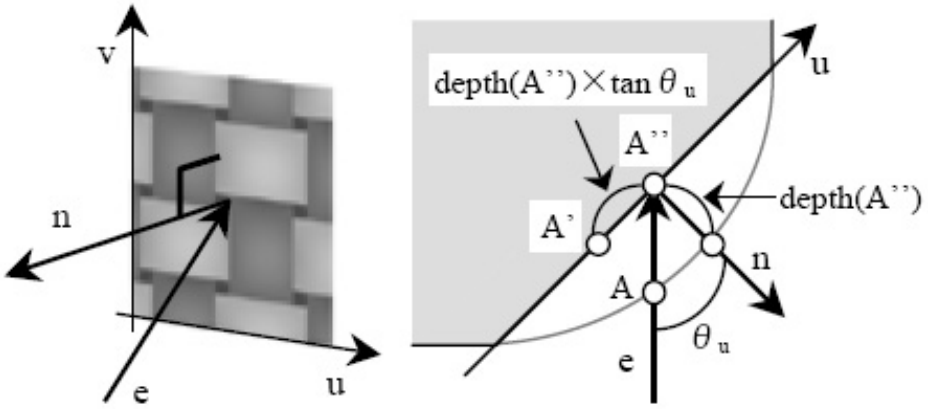


Figura 3.7: Cálculo de mapeamento de textura considerando a profundidade da superfície (KANEKO et al., 2001).

É importante notar que os cálculos realizados por essa técnica não ocorrem no espaço da tangente, como acontece com a técnica de Bump Mapping. Para o cálculo da nova coordenada de textura é realizada uma renderização utilizando projeção ortográfica, para não ocorrerem distorções com a superfície. Devido aos cálculos dessa técnica não serem realizados no espaço da tangente, a mesma apresenta maiores distorções quando aplicada à superfícies curvas, mas consegue renderizar a silhueta dos objetos. A Figura 3.8 apresenta uma imagem gerada a partir da técnica de Parallax Mapping (KANEKO et al., 2001).

3.4.2 Parallax Mapping com offset limitador

A técnica de Parallax Mapping se tornou popular em 2004 a partir do trabalho de Welsh (WELSH, 2004). Assim como Kaneko (KANEKO et al., 2001), Welsh também utiliza uma heurística para o cálculo da correta coordenada de textura para cada ponto da superfície. Na técnica proposta, é inserido um *offset* limitador. Este *offset* limita o deslocamento máximo da nova coordenada de textura em relação à coordenada de textura inicial, reduzindo as distorções ocorridas na renderização das superfícies. Esta técnica é implementada em conjunto com a técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997), nos *hardwares* gráficos programáveis. Isso possibilita que o mapa de profundidade da superfície possa ser quantizado e armazenado junto ao mapa de normais da superfície, onde as normais ficam armazenadas nos canais *RGB* e a profundidade no canal *alpha* da textura. Nesta técnica, todos os cálculos são realizados no espaço

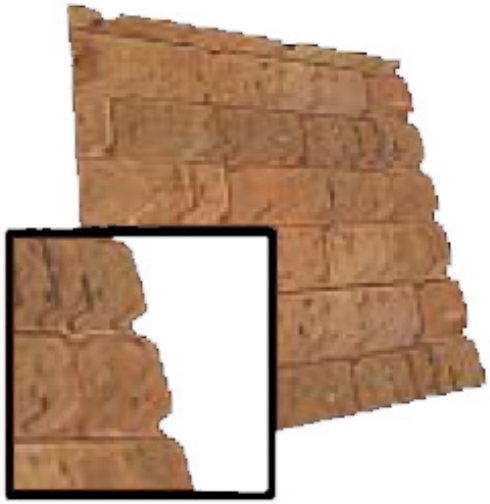


Figura 3.8: Superfície renderizada a partir da técnica de Parallax Mapping proposta por Kaneko (KANEKO et al., 2001).

da tangente, o que permite que a técnica possa ser aplicada a qualquer tipo de superfície.

Para calcular a nova coordenada de textura, o algoritmo utiliza a coordenada de textura inicial, o vetor de visão pelo qual a cena é observada e dois fatores *scale* e *bias* que devem ser configurados para cada tipo de superfície. Estes fatores devem ser ajustados para cada superfície, e são utilizados para modificar o deslocamento de acordo com a altura da superfície em cada ponto. As equações 3.6, 3.7 e 3.8 são utilizadas para o cálculo da nova coordenada de textura.

$$height' = height \times scale + bias \tag{3.6}$$

$$(u,v)' = (u,v) + height' \times \frac{V_{eye}(x,y)}{V_{eye}(z)} \tag{3.7}$$

$$(u,v)'' = (u,v) + height' \times V_{eye}(x,y) \tag{3.8}$$

Na equação 3.7, $(u,v)'$ representa a nova coordenada de textura obtida sem a imposição do *offset* limitador. Como $V_{eye}(z)$ está definido no intervalo]0.0, 1.0] , a divisão pelo mesmo aumenta a distância entre a coordenada de textura inicial e a nova coordenada de textura. Na equação 3.8 $(u,v)''$ representa a nova coordenada de textura, obtida com o *offset* limitador, onde a divisão pelo valor $V_{eye}(z)$ é eliminada. A Figura 3.9 ilustra o uso da técnica de Parallax Mapping no cálculo da correta coordenada de acesso à textura.

O algoritmo em alto nível 3.4.2 ilustra o funcionamento da técnica de Parallax Mapping proposta por Welsh (WELSH, 2004).

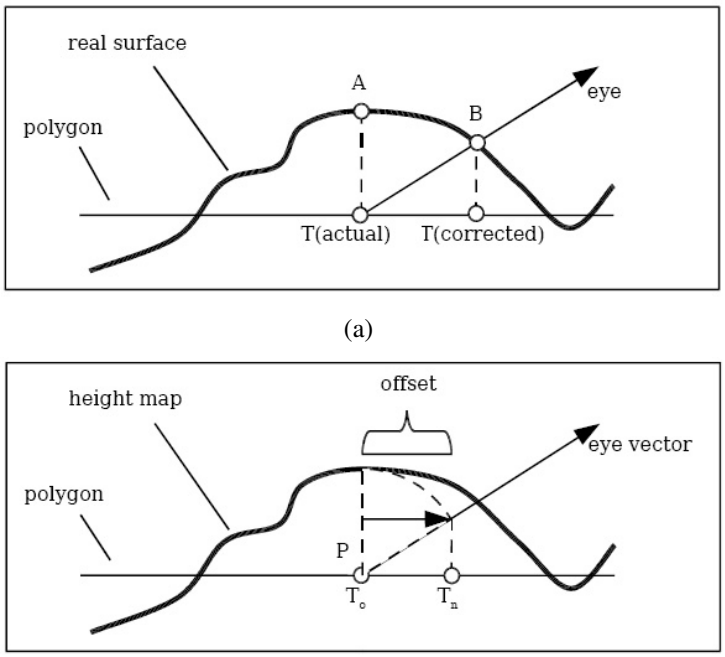


Figura 3.9: Aplicação do Parallax Mapping. (a) O ponto "A" representa o valor obtido utilizando o Bump Mapping padrão, "B" é o valor correto a ser obtido. (b) "T(n)" representa o valor aproximado obtido a partir do offset do Parallax Mapping (WELSH, 2004).

Algorithm 2 Algoritmo de Parallax Mapping (WELSH, 2004) utilizado no processamento de *pixels* da superfície.

$tex1 \leftarrow Textura2D_{(cor\ difusa)}$
$tex2 \leftarrow Textura2D_{(mapa\ normais)}$
$tex3 \leftarrow Textura2D_{(mapa\ profundidade)}$
$scale \leftarrow fator\ de\ escala$
$bias \leftarrow fator\ de\ bias$

for each $Pixel(x,y)$ **do**
  $v \leftarrow \vec{V}_{visao}$
  $l \leftarrow \vec{V}_{iluminacao}$
  $uv \leftarrow coordenada\ de\ textura$

  $offset \leftarrow readTexture2D(tex3, uv)$
  $newUV \leftarrow uv + (offset * scale + bias) * v$

  $color \leftarrow readTexture2D(tex1, newUV)$
  $n \leftarrow readTexture2D(tex2, newUV)$
  $output(x,y) \leftarrow phongEquation(n,l,v,color)$
end for

A principal vantagem dessa técnica é poder ser utilizada em conjunto com a técnica de Bump Mapping nos *hardwares* gráficos programáveis, requisitando apenas três instruções adicionais para ser implementada na GPU. Uma desvantagem da heurística utilizada é que a mesma

apresenta bons resultados apenas quando aplicada a mapas com padrões de repetição, como paredes de tijolos e similares. Quando essa técnica é aplicada a modelos com mapas de profundidade genéricos, como personagens em um jogo, o efeito visual gerado é de baixa qualidade. Além disso, é necessário especificar um fator de escala (*scale*) e um *bias* para cada superfície utilizada. Atualmente a técnica de Parallax Mapping utilizada em aplicações de tempo real é a proposta por Welsh. A Figura 3.10 apresenta uma imagem gerada a partir da técnica de Parallax Mapping (WELSH, 2004).

Figura 3.10: Superfície renderizada utilizando a técnica de Parallax Mapping proposta por Welsh (WELSH, 2004).

As principais vantagens dessa técnica são:

- Permite representar maiores deformações na superfície, utilizando uma heurística para tratamento do efeito de *motion parallax*;
- Fácil de ser implementada e integrada a *pipelines* de renderização;
- Implementação na GPU, utilizando *shader model* 1.0;
- Baixo tempo de processamento;
- Baixo consumo de memória, utiliza o canal *alpha* da textura de normais da superfície.

As principais desvantagens dessa técnica são:

- Não apresenta bons resultados quando aplicados a mapas profundidade genéricos;

- Não trata os efeitos de *self-occlusion*, *self-shadow* e silhuetas;
- Exige a configuração dos fatores de *scale* e *bias* para cada superfície;
- Pode gerar grandes distorções na renderização da superfície.

Um ambiente para o uso eficaz dessa técnica são aplicações em tempo real, na renderização de superfícies com padrões, como paredes de tijolos. Como essa técnica consegue representar maiores detalhes e deformações nas superfícies, ela pode ser utilizada na renderização de objetos que são visualizados a uma pequena distância. Entretanto, quando os objetos são visualizados a ângulos próximos de 90 graus, a qualidade visual observada ainda é baixa.

3.5 Técnicas baseadas em traçado de raios (*ray-tracing*)

3.5.1 Relief Mapping

Relief Mapping foi uma técnica originalmente proposta por Oliveira em 2000 (OLIVEIRA; BISHOP; MCALLISTER, 2000). Originalmente, essa técnica aplicava modificações no mapeamento de texturas nas superfícies, possuindo algumas similaridades com o trabalho proposto por Kaneko (KANEKO et al., 2001). Recentemente, a técnica de Relief Mapping foi modificada por Policarpo (POLICARPO; OLIVEIRA; COMBA, 2005). Em sua nova versão, a técnica de Relief Mapping utiliza como base de funcionamento um algoritmo de traçado de raios (*ray-tracing*) (SILVA; MARTINS, 2005), pode ser aplicada a qualquer tipo de superfície e tira proveito dos *hardwares* gráficos programáveis. O objetivo dessa técnica é simular malhas detalhadas com precisão, tratando os efeitos de *motion parallax*, *self-occlusion* e *self-shadow*.

A técnica de Relief Mapping utiliza um algoritmo de traçado de raios para calcular o ponto exato de colisão entre o vetor de visão, pelo qual a superfície é observada, e o mapa de profundidade da superfície e, com isso, calcular o correto mapeamento de textura para cada ponto da superfície. Outras técnicas como Parallax Mapping (WELSH, 2004), que utilizam heurísticas para calcular o correto mapeamento de textura, apresentam uma qualidade visual inferior e não conseguem representar detalhes com grande precisão. A técnica de Relief Mapping utiliza um mapa de profundidade das superfícies; porem o mapa utilizado é invertido, onde 0 representa a parte com maior altitude e 1 a parte com menor altitude.

No algoritmo de traçado de raios é utilizado o mapa de profundidade da superfície, a posição inicial do raio traçado (formada pela coordenada de textura inicial e altura 0) e o vetor de visão pelo qual cada ponto da superfície é observada. Este algoritmo é dividido em duas partes: busca

linear e busca binária, e é aplicado na redenrização de cada ponto da superfície da malha. Na busca linear é criado um segmento de reta entre o ponto inicial do raio na superfície, definido pela coordenada de textura naquele ponto e altura 0, e o ponto final da superfície, definido pela altura 1 e as coordenadas de textura para essa altura (calculadas a partir do vetor de visão). Esse segmento de reta é dividido em *n* segmentos, onde *n* é o número de iterações da busca linear. O algoritmo então faz *n* iterações viajando com o raio para cada uma das subdivisões da reta. Nesta parte do algoritmo são armazenados os dois pontos mais próximos encontrados, sendo que um deve estar acima da superfície e outro abaixo da mesma. Os dois novos pontos encontrados pelo algoritmo de busca linear formam um novo segmento de reta, onde é aplicado o algoritmo de busca binária. O motivo do uso das técnicas de busca linear e busca binária em conjunto justifica-se, pois nenhuma das duas técnicas sozinha conseguiria encontrar eficientemente o ponto de colisão. Utilizando apenas a técnica de busca binária não haveria como saber se o ponto de colisão encontrado seria realmente o primeiro ponto de colisão entre o raio e a o mapa de profundidade, tendo em vista que podem existir vários pontos de colisão. A Figura 3.11 ilustra o processo do uso da busca binária em separado.

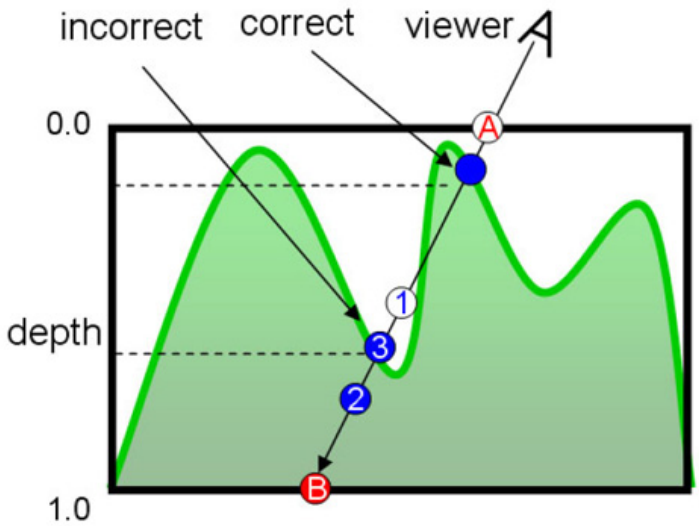


Figura 3.11: Ilustração da técnica de busca binária para detecção do ponto de colisão no mapa de profundidade de uma superfície (POLICARPO; OLIVEIRA; COMBA, 2005).

Utilizando apenas a técnica de busca linear seria necessário um número muito maior de iterações para encontrar o ponto de colisão com a superfície, o que reduziria o desempenho do algoritmo. Após o ponto de colisão ser encontrado é calculada a correta coordenada de textura para aquele ponto da superfície, e em seguida aplicada a técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997). Combinando as técnicas de busca linear e binária, é possível conseguir uma melhor relação entre qualidade visual e desempenho, gerando imagens com alta qualidade visual e desempenho que permite o uso da técnica em aplicações de tempo real. O

algoritmo em alto nível 3.5.1 ilustra o funcionamento da técnica de Parallax Mapping proposta por Policarpo (POLICARPO; OLIVEIRA; COMBA, 2005).

Algorithm 3 Algoritmo de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005) utilizado no processamento de *pixels* da superfície.

$tex1 \leftarrow Textura2D_{(cor\ difusa)}$
$tex2 \leftarrow Textura2D_{(mapa\ normais)}$
$tex3 \leftarrow Textura2D_{(mapa\ profundidade)}$
$linearSteps \leftarrow passos\ de\ busca\ linear$
$binarySteps \leftarrow passos\ de\ busca\ binaria$
$depth \leftarrow fator\ de\ profundidade$

for each $Pixel(x,y)$ **do**
  $v \leftarrow \vec{V}_{visao}$
  $l \leftarrow \vec{V}_{iluminacao}$
  $uv \leftarrow coordenada\ de\ textura$

  $rayPos \leftarrow (uv[0], uv[1], 0.0)$
  $rayStepVec \leftarrow (v.x/(v.z * linearSteps), v.y/(v.z * linearSteps), 1.0/linearSteps)$
  $rayStepVec \leftarrow rayStepVec * depth$

  for $i = 1$ to $linearSteps$ **do**
    $rayPos \leftarrow linearSearch(rayPos, rayStepVec, tex3)$
  end for

  for $i = 1$ to $binarySteps$ **do**
    $rayPos \leftarrow binarySearch(rayPos, rayStepVec)$
  end for

  $newUV \leftarrow (rayPos[0], rayPos[1])$
  $color \leftarrow readTexture2D(tex1, newUV)$
  $n \leftarrow readTexture2D(tex2, newUV)$
  $output(x,y) \leftarrow phongEquation(n, l, v, color)$
end for

A técnica de Relief Mapping também pode ser utilizada para gerar *self-shadows*. Após o descobrimento do ponto de colisão entre o vetor de visão e o mapa de profundidade da superfície é possível traçar novos segmentos de reta entre o ponto de colisão e a posição das luzes da cena. Para cada um dos novos segmentos de reta traçados, deve-se aplicar novamente o algoritmo de Relief Mapping e gerar sombras quando for detectada colisão entre o segmento de reta e o mapa de profundidade da superfície. A Figura 3.12 ilustra a aplicação da técnica de Relief Mapping, com visões bem próximas ao modelo.

As principais vantagens dessa técnica são:

Figura 3.12: Aplicação do Relief Mapping em superfícies arbitrárias. Diferentes mapas aplicados a uma chaleira (POLICARPO; OLIVEIRA; COMBA, 2005).

- Permite simular malhas extremamente detalhadas com precisão;
- Trata corretamente os efeitos de *motion parallax*, *self-occlusion* e *self-shadow*;
- Pode ser facilmente integrada a *pipelines* de renderização;
- Implementação na GPU, utilizando *shader model* 2.0;
- Baixo consumo de memória.

As principais desvantagens dessa técnica são:

- Pode apresentar artefatos na rederização de superfícies;
- Pode ser necessário um grande número de iterações na renderização das superfícies;
- Alto tempo de processamento.

Um ambiente para o uso eficaz dessa técnica são aplicações em tempo real, na renderização de superfícies muito detalhadas que são observadas a uma pequena distância. O uso dessa técnica demanda um alto tempo de processamento, sendo necessário a aquisição de um *hardware* gráfico moderno. No entanto, a mesma consegue gerar ambientes com alto nível de realismo.

3.5.2 Parallax Occlusion Mapping

Parallax Occlusion Mapping (TATARCHUK, 2006) é outra técnica recente para simulação de malhas detalhadas a partir da utilização de um algoritmo de traçado de raios. O objetivo

dessa técnica é simular malhas detalhadas com precisão e sem a presença de artefatos. Os efeitos tratados por essa técnica são *motion parallax*, *self-occlusion* e *self-shadow*.

O algoritmo de traçado de raios utilizado pela técnica de POM é o mesmo utilizado pela técnica de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005); porém, apenas a busca linear é utilizada. A Figura 3.13 ilustra o uso da técnica de busca linear no mapa de profundidade de uma superfície. Por utilizar apenas a busca linear, essa técnica consegue reduzir drasticamente, ou mesmo eliminar, a presença de artefatos na cena. No entanto, para a busca linear apresentar um bom resultado, podem ser necessárias centenas de iterações, tornando difícil a aplicação da técnica tempo real. Para amenizar este problema, a técnica de POM calcula o número de iterações de cada raio em tempo real. Raios perpendiculares à superfície necessitam de um pequeno número de iterações, pois não modificam o mapeamento de textura naquele ponto, enquanto raios quase paralelos à superfície necessitam de um grande número de iterações. Com isso o número de iterações de cada raio é calculado a partir de uma interpolação linear entre o número mínimo e máximo de amostras, definidos para cada superfície, utilizando como fator de interpolação a inclinação do vetor de visão. Devido ao número de iterações da técnica ser cálculado dinâmicamente, a mesma requer o uso de *hardwares* gráficos modernos, com suporte ao modelo de shader 3.0.

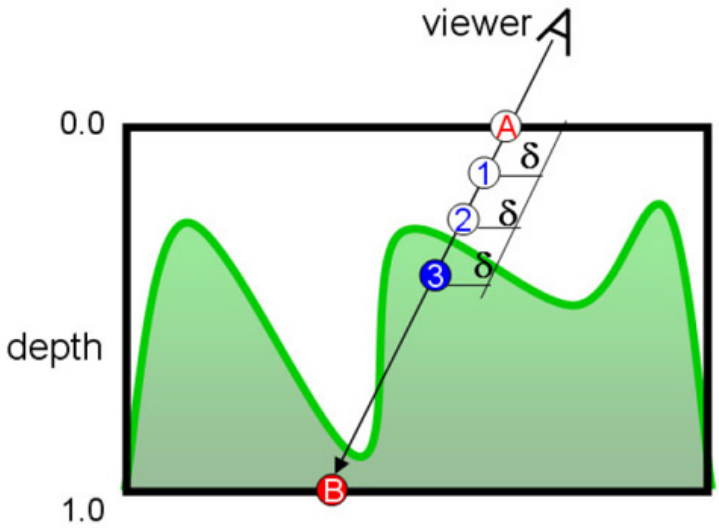


Figura 3.13: Ilustração da técnica de busca linear para detecção do ponto de colisão no mapa de profundidade de uma superfície (POLICARPO; OLIVEIRA; COMBA, 2005).

A técnica de POM também calcula em tempo real o nível de *mip-mapping* (FOLEY et al., 1996) para cada ponto da superfície renderizada. Esse valor pode ser utilizado para verificar quais partes da superfície estão mais próximas e mais distantes do observador e aplicar diferentes técnicas a cada uma dessas partes. Por exemplo, partes mais distantes da superfície renderizada podem utilizar apenas a técnica de Bump Mapping (PEERCY; AIREY; CABRAL,

1997), enquanto partes intermediárias podem utilizar a técnica de POM com um menor número de iterações. O algoritmo em alto nível 3.5.2 ilustra o funcionamento da técnica de Parallax Occlusion Mapping proposta por Tatarchuk (TATARCHUK, 2006).

Algorithm 4 Algoritmo de Parallax Occlusion Mapping (TATARCHUK, 2006) utilizado no processamento de *pixels* da superfície.

$tex1 \leftarrow Textura2D_{(cor\ difusa)}$
$tex2 \leftarrow Textura2D_{(mapa\ normais)}$
$tex3 \leftarrow Textura2D_{(mapa\ profundidade)}$
$pomThreshold \leftarrow fator\ filtro\ limiar$
$minSamples \leftarrow numero\ minimo\ de\ amostras$
$maxSamples \leftarrow numero\ maximo\ de\ amostras$
$depth \leftarrow fator\ de\ profundidade$

for each $Pixel(x,y)$ **do**
  $v \leftarrow \vec{V}_{visao}$
  $l \leftarrow \vec{V}_{iluminacao}$
  $uv \leftarrow coordenada\ de\ textura$
  $sn \leftarrow normal\ original\ da\ superficie$

  $mipmapLevel \leftarrow calculateMipmap(uv)$
  if $mipmapLevel <= pomThreshold$ **then**
    $rayPos \leftarrow (uv[0], uv[1], 0.0)$
    $rayStepVec \leftarrow (v.x/(v.z * linearSteps), v.y/(v.z * linearSteps), 1.0/linearSteps)$
    $rayStepVec \leftarrow rayStepVec * depth$

    $numSteps \leftarrow lerp(minSamples, maxSamples, dotProduct(sn, v))$
    for $i = 1$ to $numSteps$ **do**
      $rayPos \leftarrow linearSearch(rayPos, rayStepVec, tex3)$
    end for

    $uv \leftarrow (rayPos[0], rayPos[1])$
  end if

  $color \leftarrow readTexture2D(tex1, uv)$
  $n \leftarrow readTexture2D(tex2, uv)$
  $output(x,y) \leftarrow phongEquation(n, l, v, color)$
end for

Outro recurso tratado pela técnica é a geração de sombras suaves. A técnica de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), como foi proposta, trata apenas sombras rígidas (sem apresença de umbra e penumbra), o que pode contribuir para a geração de imagens cerrilhadas. Isso ocorre pois, na técnica de Relief Mapping são traçados segmentos de reta entre o ponto de colisão e as luzes da cena, e quando é encontrada uma colisão entre esse segmento e o mapa de profundidade é gerado um ponto de sombra na superfície. Na técnica

de POM os segmento de reta entre o ponto de colisão e as luzes da cena são subdivididos em *n* partes e para cada uma dessas subdivisões é realizado o cálculo de iluminação, considerando-se uma luz de área não pontual. O maior valor de intensidade de luz obtido em todos os pontos é utilizado, gerando com isso sombras suaves. Esta abordagem produz bons resultados visuais, mas é apenas uma heurística para o cálculo de sombras suaves. A Figura 3.14 ilustra a mesma superfície sendo renderizada com sombras rígidas e sombras suaves. A Figura 3.15 ilustra a renderização de uma malha real e uma malha com detalhes simulados através da técnica de Parallax Occlusion Mapping (TATARCHUK, 2006).

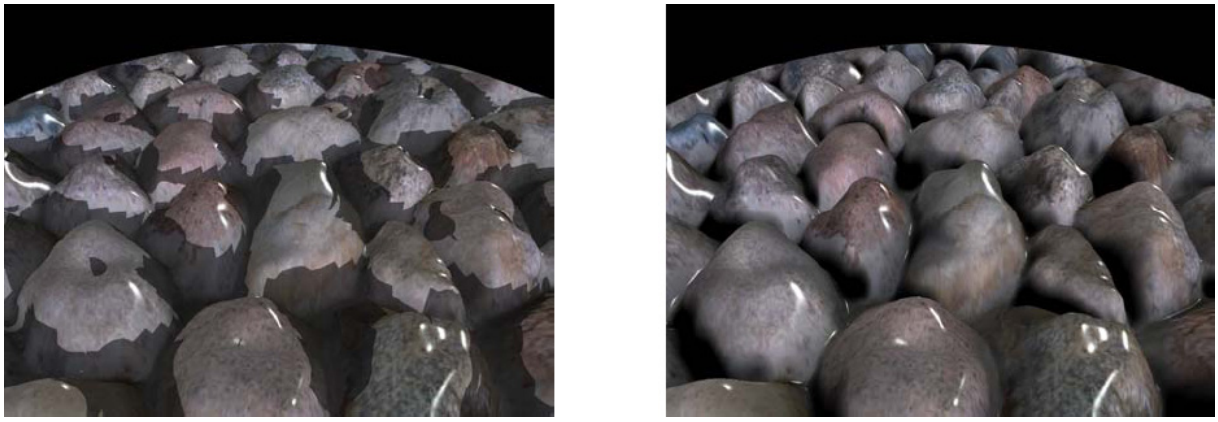


Figura 3.14: Geração de sombras rígidas e suaves para uma mesma superfície. Sombras rígidas e presença de cerrilhamento (esquerda), sombras suaves sem presença de cerrilhamento (direita) (TATARCHUK, 2006).

As principais vantagens dessa técnica são:

- Permite simular malhas extremamente detalhadas com precisão;
- Trata corretamente os efeitos de *motion parallax*, *self-occlusion* e *self-shadow*, gerando sombras suaves;
- Pode ser facilmente integrada a *pipelines* de renderização;
- Baixo consumo de memória.

As principais desvantagens dessa técnica são:

- Implementação na GPU, requer *hardwares* gráficos modernos com suporte ao modelo de shaders 3.0;
- Necessário um grande número de iterações na renderização das superfícies;
- Exige a configuração de fatores de amostra mínima e máxima para cada superfície;

(a) (b)

Figura 3.15: Renderização de uma malha real e uma malha com detalhes simulados através da técnica de POM. (a) Malha real, composta por 1 milhão e 500 triângulos, (b) Malha simulada, composta por 1.100 triângulos (TATARCHUK, 2006).

- Alto tempo de processamento.

A técnica de Parallax Occlusion Mapping pode ser utilizada em aplicações de tempo real, mas requer o uso de *hardwares* gráficos modernos. Devido à técnica demandar um alto tempo de processamento, pode ser difícil aplicar a mesma a vários objetos em uma mesma cena. Portanto, um ambiente para uso eficaz dessa técnica são aplicações em tempo real, com uso de *hardwares* gráficos modernos e aplicações de renderização *offline* (que não são executadas em tempo real). Essa técnica pode ser utilizada em ferramentas para visualização e renderização de malhas devido a alta precisão apresentada na renderização das malhas. Essa técnica apresenta uma melhor relação entre qualidade visual e desempenho na renderização de malhas observadas a distâncias extremamente próximas.

3.6 Técnicas pré-calculadas

3.6.1 View-Dependent Displacement Mapping

View-dependent displacement mapping (WANG et al., 2003a) é uma técnica para simulação de malhas detalhadas que trata corretamente os efeitos de *motion parallax*, *self-occlusion*, *self-*

shadow e silhuetas. Uma grande vantagem dessa técnica é tratar o efeito de silhueta que, dentre os trabalhos apresentados, só havia sido tratado pelas técnicas de Parallax Mapping Kaneko (KANEKO et al., 2001) e pela técnica de Relief Mapping proposta por Oliveira (OLIVEIRA; BISHOP; MCALLISTER, 2000). Para renderizar com precisão a silhueta das malhas, a técnica de VDM não utiliza o espaço da tangente, o que restringe a aplicação da mesma a qualquer tipo de malha.

As técnicas de simulação de superfícies detalhadas que utilizam o algoritmo de traçado de raios conseguem obter uma alta qualidade visual e tratar com precisão os efeitos de *motion parallax*, *self-occlusion*, *self-shadow*. No entanto, essas técnicas podem necessitar de um grande número de iterações para renderização das superfícies, demandando um alto tempo de processamento. A técnica de VDM utiliza um mapa pré-calculado para cada superfície que permite renderizar cada ponto da mesma com apenas uma iteração. O mapa utilizado é endereçado através do quinteto $(x, y, \theta, \sigma, c)$, onde x, y é a coordenada de textura inicial para um determinado ponto na superfície, θ, σ é o ângulo do vetor de visão em coordenadas esféricas e c é a curvatura da superfície. Esses quatro primeiros elementos utilizados no endereçamento do mapa são os utilizados nas técnicas de traçado de raios para o cálculo da nova posição de mapeamento de textura; portanto, esses valores podem ser pré-calculados utilizando um grande número de iterações e armazenados nesse mapa. O quinto elemento desse mapa é a curvatura da superfície. Esse elemento é necessário para a renderização da malha e detecção de silhueta da mesma, pois a técnica de VDM não utiliza o espaço da tangente. A Figura 3.16 ilustra uma malha renderizada a partir da técnica de VDM.

Figura 3.16: Superfície renderizada utilizando a técnica de View-dependent displacement mapping (WANG et al., 2003a). É possível perceber que a silhueta da malha é renderizada em ambas as imagens.

Os mapas pré-calculados para as superfícies possuem as dimensões 128x128x32x8x16,

onde 128x128 refere-se ao mapa de profundidade, 32x8 refere-se ao ângulo de visão e 16 refere-se à curvatura da malha. Com isso, os mapas para cada superfície precisam de 64MB para serem armazenados. Considerando-se mapas de profundidade mais detalhados como 512x512 ou 1024x1024, seriam necessários respectivamente 1GB e 4GB de espaço para armazenamento. Para comprimir o tamanho do mapa utilizado, a técnica de VDM utiliza o algoritmo de SVD (Singular-value decomposition) (WANG et al., 2003a). Uma das vantagens desse algoritmo, segundo os autores, é poder decompor um mapa com várias dimensões em dois mapas com um número menor de dimensões. Utilizando o algoritmo de SVD é possível reduzir o tamanho dos mapas de 64MB para 4MB. Os autores demonstram que a qualidade visual obtida com o mapa comprimido é similar à obtida com o mapa sem compressão, como visto na Figura 3.17. Mas mesmo utilizando o algoritmo de SVD, ainda seriam necessários 64MB de memória na renderização de uma superfície que utilizasse um mapa de curva de nível com resolução 512x512.

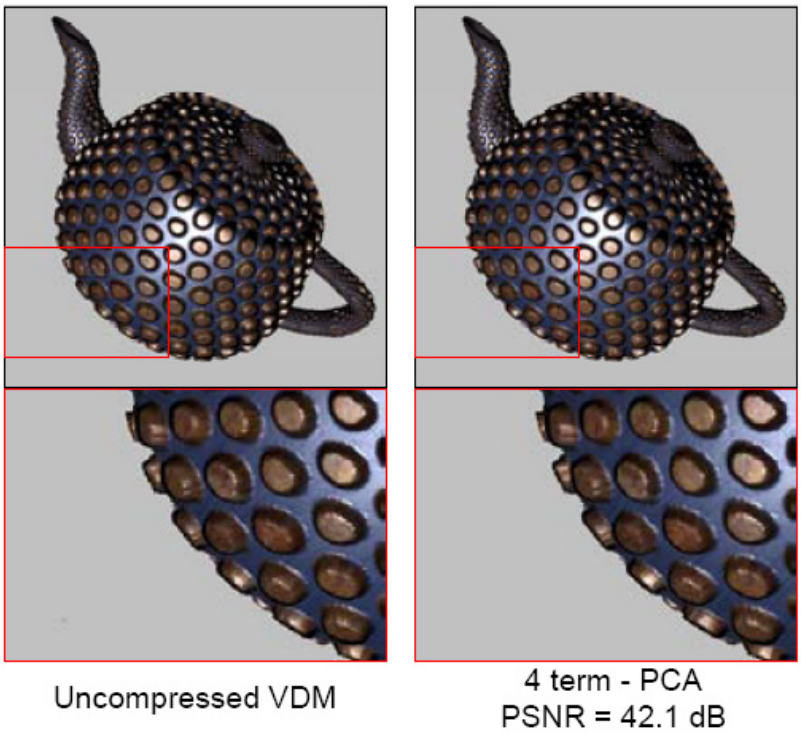


Figura 3.17: Renderização de uma chaleira utilizando a técnica de VDM. Renderização utilizando mapa sem compressão (esquerda), renderização utilizando mapa com compressão (direita) (WANG et al., 2003a).

A técnica de VDM demanda uma grande quantidade de memória para ser utilizada, mas consegue renderizar superfícies detalhadas com alta qualidade visual e alto desempenho.

As principais vantagens dessa técnica são:

- Permite simular malhas extremamente detalhadas com precisão;
- Trata corretamente os efeitos de *motion parallax*, *self-occlusion*, *self-shadow* e silhuetas;
- Implementação na GPU. No entanto, a mesma não é fornecida pelos autores;
- Baixo tempo de processamento.

As principais desvantagens dessa técnica são:

- Necessita do pré-cálculo de um mapa com cinco dimensões e compressão do mesmo, para cada superfície;
- Auto consumo de memória.

Um ambiente para o uso eficaz dessa técnica são aplicações em tempo real, na renderização de superfícies muito detalhadas. Apesar dessa técnica demandar um baixo tempo de processamento, sua aplicação pode ficar restrita a um pequeno número de malhas, devido ao alto consumo de memória no armazenamento dos mapas das superfícies.

3.6.2 Sphere Tracing

Sphere Tracing (HART, 1996) é uma técnica para aceleração do algoritmo de traçado de raios a partir do uso de mapas tridimensionais com distâncias seguras a serem percorridas para cada *voxel*[4]. Para gerar imagens com alta qualidade, a partir das técnicas de traçado de raios apresentadas na Seção 3.5, é necessário utilizar um grande número de iterações. Mesmo utilizando um grande número de iterações, e amostrando o mapa de profundidade da superfície várias vezes, não há garantia que seja encontrado o primeiro ponto de colisão entre o raio e o mapa de profundidade da superfície. Em mapas de profundidade de alta freqüência, por exemplo, é provável que algumas interseções sejam perdidas, mesmo utilizando um número alto de iterações. A Figura 3.18 ilustra uma situação onde é difícil detectar a colisão entre o raio e o mapa de profundidade.

Para resolver esse problema, a técnica de Sphere Tracing (HART, 1996) pré-calcula um mapa tridimensional, onde cada *voxel* do mapa é o centro de uma esfera e armazena o raio da mesma. O raio de cada esfera é definido pela menor distância entre seu centro e o mapa de profundidade da superfície. Portanto, a partir de um *voxel* do mapa, é possível viajar com o raio uma distância igual ao raio da esfera sem o risco de perder interseções com o mapa de

[4]Voxel é um ponto no espaço tridimensional.

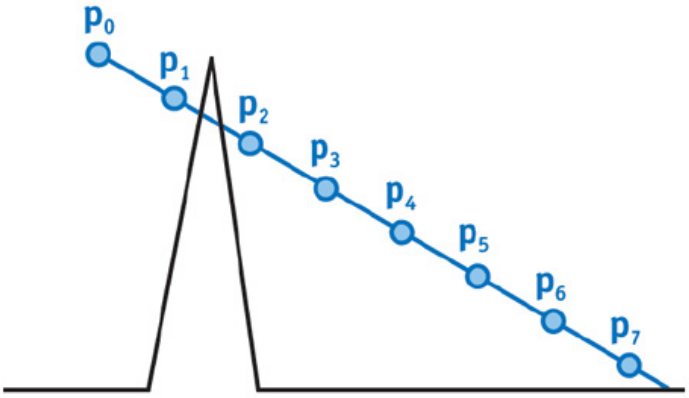


Figura 3.18: Detecção de colisão entre um raio e um mapa de profundidade de alta freqüência. Neste caso é difícil detectar a colisão com o mapa de profundidade da superfície (DONNELLY, 2005).

profundidade da superfície. Além disso, os raios das esferas armazenados permitem que o raio viaje maiores distâncias, sendo necessário um menor número de iterações para renderizar as superfícies com alta qualidade visual. A Figura 3.19 ilustra a técnica de Sphere Tracing.

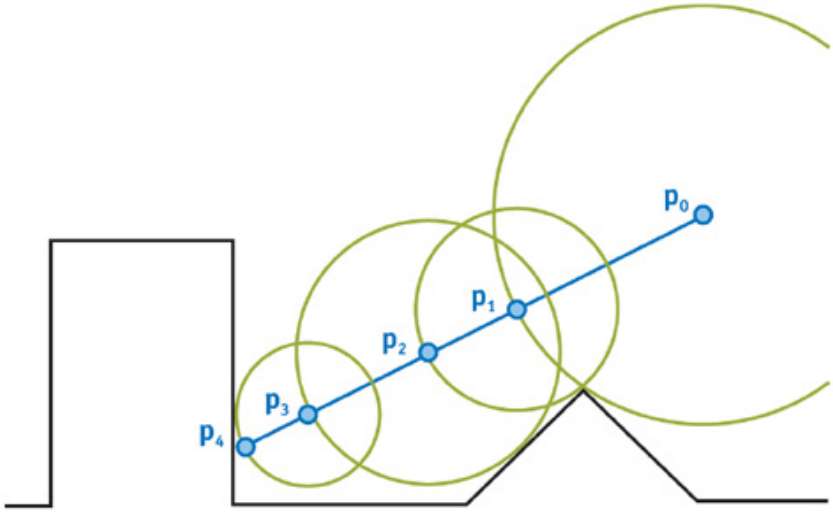


Figura 3.19: Ilustração da técnica de Sphere Tracing para detecção de colisão entre um raio e o mapa de profundidade da superfície (DONNELLY, 2005).

O algoritmo em alto nível 3.6.2 ilustra o funcionamento da técnica de Sphere Tracing proposta por Donnelly (DONNELLY, 2005).

Donnelly (DONNELLY, 2005) apresenta uma implementação da técnica de Sphere Tracing explorando o *hardware* gráfico programável. Em sua implementação são tratados os efeitos de *motion parallax* e *self-occlusion*. Apesar de não ser explicitado em seu trabalho, também é possível tratar o efeito de *self-shadow* utilizando as técnicas propostas por Policarpo (POLICARPO; OLIVEIRA; COMBA, 2005) ou por Tatarchuk (TATARCHUK, 2006). Donnelly uti-

Algorithm 5 Algoritmo de Sphere Tracing (DONNELLY, 2005) utilizado no processamento de *pixels* da superfície.

$tex1 \leftarrow Textura2D_{(cor\ difusa)}$
$tex2 \leftarrow Textura2D_{(mapa\ normais)}$
$tex3 \leftarrow Textura3D_{(mapa\ esferas)}$
$numSteps \leftarrow numero\ de\ passos\ da\ busca$
$depth \leftarrow fator\ de\ profundidade$
$sphereMapWidth \leftarrow Largura\ do\ mapa\ de\ esferas$
$sphereMapDepth \leftarrow Profundidade\ do\ mapa\ de\ esferas$

for each $Pixel(x,y)$ **do**
  $v \leftarrow \vec{V}_{visao}$
  $l \leftarrow \vec{V}_{iluminacao}$
  $uv \leftarrow coordenada\ de\ textura$

  $rayPos \leftarrow (uv[0], uv[1], 0.0)$
  $sphereDepthWidth \leftarrow sphereMapDepth/SphereMapWidth$
  $rayVec \leftarrow (v[0] * sphereDepthWidth, v[1] * sphereDepthWidth, v[2])$
  $rayVec \leftarrow rayVec * depth$

  for $i = 1$ to $numSteps$ **do**
    $stepDist \leftarrow readTexture3D(tex3, rayPos)$
    $rayPos \leftarrow rayPos + rayVec * stepDist$
  end for

  $newUV \leftarrow (rayPos[0], rayPos[1])$
  $color \leftarrow readTexture2D(tex1, newUV)$
  $n \leftarrow readTexture2D(tex2, newUV)$
  $output(x,y) \leftarrow phongEquation(n, l, v, color)$
end for

liza mapas de esferas com resoluções de 256x256x16 e 512x512x16, sendo necessários respectivamente 1MB e 4MB de memória para armazenar esses mapas. Este valor é bastante inferior a memória necessária para armazenar os mapas da técnica de VDM (WANG et al., 2003a), considerando-se a mesma resolução.

O uso do mapa de esferas faz com que essa técnica seja livre de artefatos, mas podem haver distorções na imagem gerada quando o número de iterações utilizado não for suficiente para encontrar-se o ponto de colisão. No entanto, a presença de distorções na imagem é pode ser menos perceptível ao observador que a presença de artefatos. De maneira geral, a técnica de Sphere Tracing permite obter um alto desempenho e uma alta qualidade visual, mas requer a geração e armazenamento do mapa de esferas para cada superfície renderizada. A Figura 3.20 exibe uma superfície de alta freqüência renderizada com a técnica de Sphere Tracing.

Figura 3.20: Superfície renderizada utilizando a técnica de Sphere Tracing (DONNELLY, 2005), e um mapa de profundidade de alta frequência.

As principais vantagens dessa técnica são:

- Permite simular malhas extremamente detalhadas com precisão;
- Trata corretamente os efeitos de *motion parallax* e *self-occlusion*;
- Implementação na GPU, utilizando *shader model* 2.0;
- Imagens geradas são livres de artefatos;
- Pode ser facilmente estendida para tratar o efeito de *self-shadow*.

As principais desvantagens dessa técnica são:

- Necessita do pré-cálculo do mapa de esferas, para cada superfície;
- Imagens geradas podem apresentar distorções;
- Alto tempo de processamento;
- Alto consumo de memória.

Um ambiente para uso eficaz dessa técnica são aplicações em tempo real, na renderização de superfícies muito detalhadas que são observadas a uma pequena distância. Está técnica demanda um alto tempo de processamento e um alto consumo de memória, mas gera imagens sem a presença de artefatos, o que pode ser importante na renderização de superfícies de alta freqüência.

3.6.3 Cone Step Mapping

Cone Step Mapping (DUMMER, 2006) é outra técnica existente que visa otimizar o algoritmo de traçado de raios, através do uso de mapas pré-calculados com informações sobre a superfície. Os efeitos tratados pela técnica de Cone Step Mapping são *motion parallax* e *self-occlusion*; assim como na técnica de Sphere Tracing, também é possível estender o algoritmo para tratar o efeito de *self-shadow*. Enquanto a técnica de Sphere Tracing (DONNELLY, 2005) utiliza esferas para determinar distâncias que um raio pode viajar, a técnica de Cone Step Mapping utiliza cones para definir essa mesma distância. Para cada *texel* no mapa de profundidade da superfície é criado um cone de cabeça para baixo[5], define a distância máxima que qualquer raio que esteja localizado acima do mesmo pode percorrer. Esta técnica demanda menos memória que a técnica de Sphere Tracing (DONNELLY, 2005) onde é necessário armazenar os raios das esferas para cada *voxel*. A partir do raio e da altura do cone, pode-se calcular a distância que qualquer raio acima de cada cone pode percorrer através das equações 3.9, 3.10 e 3.11.

$$cone_{height} = tex_a - pos_z \tag{3.9}$$

$$distance = \frac{cone_{height} * cone_{radius}}{cone_{radius} + length(\vec{v}_{xy})} \tag{3.10}$$

$$pos' = pos' + \vec{v} * distance \tag{3.11}$$

A distância a ser percorrida por cada raio é definida pelas equações 3.9, 3.10 e 3.11, onde $\vec{v}$ é o vetor de visão pelo qual a superfície é observada, $cone_{height}$ é a altura do cone, definido como a diferença entre a altura do mapa de profundidade e o ponto acima do mesmo, $distance$ é a distância a ser percorrida pelo raio e pos' a nova posição de observação encontrada. A Figura 3.21 ilustra o funcionamento da técnica de Cone Step Mapping.

O algoritmo em alto nível 3.6.3 ilustra o funcionamento da técnica de Cone Step Mapping proposta por Dummer (DUMMER, 2006).

A partir de análises de histograma, Dummer (DUMMER, 2006) observa que a quantização dos raios dos cones faz com que grande parte dos raios sejam armazenados no mesmo intervalo de valores. Para melhorar a distribuição dos valores no espectro da imagem, ele propõe que as

[5] Invertido em relação ao eixo Y.

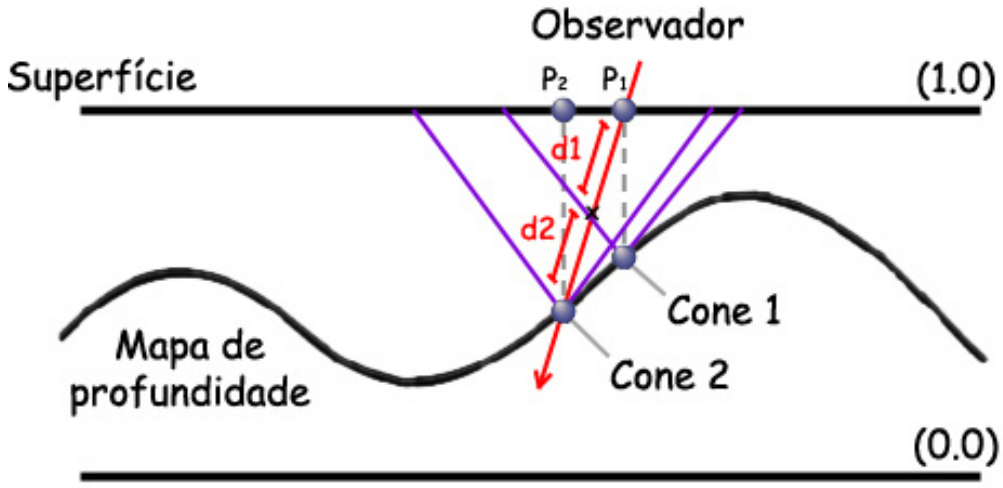


Figura 3.21: Ilustração do funcionamento da técnica de Cone Step Mapping (DUMMER, 2006). Na figura, $d1$ e $d2$ são as distâncias obtidas a partir da colisão do raio de visão com os cones 1 e 2.

raízes dos raios sejam armazenadas, o que aumenta a precisão dos raios.

Uma vantagem da técnica de Cone Step Mapping é poder ser utilizada no cálculo das distâncias a serem percorridas por um raio, sem a necessidade de armazenar mapas com um alto número de dimensões. Os cones criados para cada *texel* do mapa de profundidade precisam ter apenas seu raio armazenado, já que a altura dos mesmos pode ser calculada a partir do mapa de profundidade da superfície. Os raios dos cones podem ser quantizados e armazenados no canal *blue* da textura que contêm os mapas de normais e profundidade da superfície; onde antes estava armazenada a componente z da normal da superfície. Como a normal da superfície é unitária, pode-se calcular a componente z da mesma através de suas componentes x e y. Isso torna possível o armazenamento do mapa de cones da superfície sem aumentar o consumo de memória. Uma desvantagem dessa técnica é exigir que sejam realizados vários cálculos de colisão entre raio e cone para cada ponto renderizado da superfície. Assim como a técnica de Sphere Tracing (DONNELLY, 2005), essa técnica não apresenta artefatos, mas pode distorcer a superfície renderizada. A Figura 3.22 ilustra uma superfície sendo renderizada com a técnica de Cone Step Mapping (DUMMER, 2006).

As principais vantagens dessa técnica são:

- Permite simular malhas extremamente detalhadas com precisão;
- Trata corretamente os efeitos de *motion parallax* e *self-occlusion*;
- Implementação na GPU, utilizando *shader model* 2.0;

Algorithm 6 Algoritmo de Cone Step Mapping (DUMMER, 2006) utilizado no processamento de *pixels* da superfície.

$tex1 \leftarrow Textura2D_{(cor\ difusa)}$
$tex2 \leftarrow Textura2D_{(mapa\ normais)}$
$tex3 \leftarrow Textura2D_{(mapa\ profundidade)}$
$tex4 \leftarrow Textura2D_{(mapa\ cones)}$
$numSteps \leftarrow numero\ de\ passos\ da\ busca$
$depth \leftarrow fator\ de\ profundidade$

for each $Pixel(x,y)$ **do**
  $v \leftarrow \vec{V}_{visao}$
  $l \leftarrow \vec{V}_{iluminacao}$
  $uv \leftarrow coordenada\ de\ textura$

  $rayPos \leftarrow (uv[0], uv[1], 0.0)$
  $rayVec \leftarrow v * depth$

  for $i = 1$ to $numSteps$ **do**
    $tempUV \leftarrow (rayPos[0], rayPos[1])$
    $mapHeight \leftarrow readTexture2D(tex3, tempUV)$
    $coneHeight \leftarrow max(0, mapHeight - rayPos[2])$
    $coneRatio \leftarrow readTexture2D(tex4, tempUV)$
    $coneRatio \leftarrow coneRatio^2$
    $rayPos \leftarrow calculateConeCollisionPoint(coneHeight, coneRatio, rayPos, rayVec)$
  end for

  $newUV \leftarrow (rayPos[0], rayPos[1])$
  $color \leftarrow readTexture2D(tex1, newUV)$
  $n \leftarrow readTexture2D(tex2, newUV)$
  $output(x,y) \leftarrow phongEquation(n, l, v, color)$
end for

- Imagens geradas são livres de artefatos;
- Pode ser facilmente estendida para tratar o efeito de *self-shadow*.

As principais desvantagens dessa técnica são:

- Necessita do pré-cálculo do mapa de cones, para cada superfície;
- Imagens geradas podem apresentar distorções;
- Alto tempo de processamento.

Um ambiente para uso eficaz dessa técnica são aplicações em tempo real, na renderização de superfícies muito detalhadas que são observadas a uma pequena distância. Assim como

Figura 3.22: Malha renderizada utilizando a técnica de Cone Step Mapping. Diferentes mapas de normais e profundidade são combinados e utilizados na renderização da malha (DUMMER, 2006).

a técnica de Sphere Tracing (DONNELLY, 2005), esta técnica demanda um alto tempo de processamento, mas gera imagens sem a presença de artefatos, o que pode ser importante na renderização de superfícies de alta freqüência.

3.7 Efeitos tratados pelas técnicas

	Motion Parallax	Self-Occlusion	Self-Shadow	Silhueta
Bump Mapping				
Parallax Mapping	X			
Relief Mapping	X	X	X	
POM	X	X	X	
VDM	X	X	X	X
Sphere Tracing	X	X	I	
Cone Step Mapping	X	X	I	

Tabela 3.1: Lista dos efeitos visuais tratados pelas técnicas de Bump Mapping (PEERCY; AIREY; CABRAL, 1997), Parallax Mapping (WELSH, 2004), Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), Parallax Occlusion Mapping (TATARCHUK, 2006), View-Dependent Displacement Mapping (WANG et al., 2003a), Sphere Tracing (DONNELLY, 2005) e Cone Step Mapping (DUMMER, 2006).

A Tabela 3.1 apresenta uma relação entre as técnicas analisadas e os efeitos visuais tratados pelas mesmas. Os campos marcados com *X* representam efeitos tratados nativamente pelas técnicas, e os campos marcados com *I* representam efeitos que podem podem ser tratados pelas técnicas, mas não abordados pelos autores das mesmas.

4 *Comparação entre técnicas para simulação de detalhes*

4.1 Metodologia de comparação

A comparação entre as técnicas foi realizada com base na qualidade visual e desempenho obtido a partir da aplicação das técnicas. A qualidade visual das imagens foi comparada baseada nos parâmetros:

- Precisão na renderização da superfície (Diferença entre a imagem esperada e a imagem obtida);
- Número de efeitos visuais tratados;
- Presença de artefatos na imagem;
- Presença de distorções na imagem.

Alguns desses parâmetros são comumente utilizados em trabalhos da área para apresentar a qualidade visual da técnica proposta.

Para comparação entre as técnicas foi criada uma aplicação utilizando a API gráfica Direct3D 9.0c. Nessa aplicação foram inseridas todas as técnicas de simulação de malhas detalhadas implementadas, assim como 5 mapas de textura que representam diferentes tipos de superfícies. Para cada superfície, todas as técnicas foram comparadas na geração de imagens para dois pontos de vista: o primeiro a uma distância mediana entre o observador e a superfície onde o ângulo de observação, entre a normal da superfície e o vetor de visão, aproxima-se de 0 graus (visão quase perpendicular a superfície) e o segundo a uma distância próxima entre o observador e a superfície onde o ângulo de observação aproxima-se de 0 graus (visão quase perpendicular a superfície). As superfícies foram renderizadas na resolução de 1024x1024 *pixels*. Os mapas de normais e profundidade utilizados pelas técnicas de Bump Mapping

(PEERCY; AIREY; CABRAL, 1997), Parallax Mapping (WELSH, 2004), Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005) e Parallax Occlusion Mapping (TATARCHUK, 2006) têm a resolução de 256x256 *texels*, exceto o último mapa utilizado na renderização de uma superfície com texto, que possui a resolução de 512x512 *texels*. Os mapas utilizados nas técnicas de Cone Step Mapping (DUMMER, 2006) e Sphere Tracing (DONNELLY, 2005) possuem a mesma resolução base que os mapas utilizados nas demais técnicas e profundidade de 16 camadas.

Na renderização das superfícies foram utilizadas 16 iterações para as técnicas de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), Cone Step Mapping (DUMMER, 2006) e Sphere Tracing (DONNELLY, 2005), sendo que na técnica de Relief Mapping foram 11 iterações para a busca linear e 5 iterações na busca binária. No algoritmo de Parallax Occlusion Mapping o número mínimo de amostras utilizadas foi 8 e o número máximo 32, portanto o número de iterações de cada raio variou entre 8 e 32 iterações. Definindo-se o intervalo de amostragem entre 8 e 32 espera-se que o número médio de iterações de cada raio seja 16, possibilitando assim uma comparação mais precisa com as demais técnicas. O número de iterações utilizado foi definido a partir de testes realizados na renderização de diferentes superfícies utilizando as técnicas implementadas. Este número de iterações pode ser utilizado em aplicações na renderização de malhas detalhadas com alta qualidade visual e permite comparar as imagens geradas e o desempenho obtido a partir das técnicas utilizadas. Se um número muito grande de iterações fosse utilizado seria difícil comparar as imagens geradas pelas técnicas, devido a alta similaridade das mesmas. As técnicas de Bump Mapping (PEERCY; AIREY; CABRAL, 1997) e Parallax Mapping (WELSH, 2004) utilizam apenas uma iteração.

As técnicas capazes de tratar o efeito de *self-shadow* foram modificadas para não tratarem esse efeito na renderização, não gerando sombras sobre a superfície. Isso possibilitou uma melhor comparação entre a qualidade visual e desempenho das técnicas, além de possibilitar uma melhor visualização dos detalhes da superfície. Antes alguns desses detalhes eram ocultados pelas sombras.

O ambiente utilizado para teste foi um computador com processador de 2.0 GHz, 1GB de memória RAM e placa de vídeo GeForce 6600 GT com 128MB de memória. As imagens geradas na renderização das superfícies para cada uma das técnicas comparadas são apresentadas na Seção 4.2. O desempenho obtido na geração dessas imagens é apresentado na Seção 4.3.

4.2 Comparação entre imagens geradas

Nesta seção será apresentada uma comparação entre as imagens geradas por cada uma das técnicas para diferentes superfícies avaliadas. O desempenho obtido na renderização das superfícies é apresentado na Seção 4.3.

As Figuras 4.1, 4.2, 4.3, 4.4 e 4.5, apresentam a renderização das 5 superfícies avaliadas, visualizadas a uma distância mediana e em que o ângulo de observação, entre a normal da superfície e o vetor de visão, aproxima-se de 0 graus (visão quase perpendicular a superfície). As Figuras 4.6, 4.7, 4.8, 4.9 e 4.10 apresentam a renderização das 5 superfícies avaliadas, visualizadas a uma distância próxima e em que o ângulo de observação, entre a normal da superfície e o vetor de visão, aproxima-se de 90 graus (visão quase paralela a superfície). As técnicas utilizadas para renderização foram: (a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005). As quatro primeiras superfícies renderizadas, apresentadas nas Figuras 4.1, 4.2, 4.3, 4.4, 4.6, 4.7, 4.8 e 4.9, possuem mapas de profundidade de baixa e média freqüência. A quinta superfície renderizada, apresentada nas Figuras 4.5 e 4.10 possui um mapa de profundidade de alta freqüência.

Para as superfícies renderizadas a uma distância mediana é possível observar que as técnicas de Relief Mapping, Parallax Occlusion Mapping, Cone Step Mapping e Sphere Tracing geram imagens com um grande grau de similaridade, onde é difícil perceber diferenças entre as imagens geradas. As técnicas de Bump Mapping e Parallax Mapping conseguem adicionar detalhes as superfícies renderizadas, mas o efeito visual obtido é muito inferior às demais técnicas avaliadas. Além disso, essas técnicas não conseguem simular malhas detalhadas de alta freqüência, como apresentado nas Figuras 4.5 e 4.10. Podem ser observadas maiores diferenças entre as técnicas de Relief Mapping, Parallax Occlusion Mapping, Cone Step Mapping e Sphere Tracing nas superfícies renderizadas a uma distância próxima e em que o ângulo de observação é próximo de 90 graus, como visto nas Figuras 4.6, 4.7, 4.8, 4.9 e 4.10.

A técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997) por não tratar a profundidade das superfícies apresenta imagens planas, sem grande extrusão, quando a superfície é observada em ângulos próximos de 90 graus. Isso pode ser observado nas Figuras 4.6, 4.7, 4.8, 4.9 e 4.10. Quando a técnica é aplicada na renderização de superfícies observadas a ângulos próximos de 0 graus, o resultado final é melhor pois não é necessário tratar oclusões na superfície, como pode ser observado nas Figuras 4.1, 4.2, 4.3, 4.4 e 4.5. A técnica de Par-

allax Mapping (WELSH, 2004) apresenta problemas similares a técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997), mas consegue gerar imagens com maior qualidade visual, apresentando maior extrusão nas superfícies renderizadas. Isso pode ser observado nas Figuras 4.6, 4.7, 4.8, 4.9 e 4.10. No entanto, ambas as técnicas não apresentam bons resultados na renderização de superfícies de alta freqüência.

A técnica de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005) consegue gerar imagens com um alto nível de extrusão, mesmo quando a superfície é observada a ângulos próximos de 0 graus. Isso pode ser observado nas Figuras 4.6, 4.7, 4.8, 4.9 e 4.10. No entanto, a técnica utiliza uma busca linear e uma busca binária para encontrar o correto ponto observado na superfície da malha, como citado na Seção 3.5.1. No entanto, quando a busca linear falha, não conseguindo achar um ponto abaixo e outro acima da superfície, são gerados artefatos na imagem final.

A técnica de Parallax Occlusion Mapping (TATARCHUK, 2006) consegue gerar imagens com a melhor qualidade visual entre todas as técnicas comparadas, sendo necessário para isso um grande número de iterações. Nas imagens geradas com o número de iterações variando entre 8 e 32 a qualidade visual apresentada pela técnica foi similar a apresentada pelas técnicas de Cone Step Mapping (DUMMER, 2006) e Sphere Tracing (DONNELLY, 2005).

Por fim, as técnicas de Cone Step Mapping (DUMMER, 2006) e Sphere Tracing (DONNELLY, 2005) conseguem gerar imagens com alta qualidade visual. Diferente das técnicas de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005) e Parallax Occlusion Mapping (TATARCHUK, 2006) essas técnicas são livres de artefatos, mas podem renderizar a superfície com deformações e menores níveis de extrusão. Os mapas de esfera e cone utilizados por essas técnicas impedem que colisões com o mapa de profundidade da superfície sejam perdidas. No entanto, alguns raios podem não conseguir chegar ao seu destino final, devido a um baixo número de iterações, o que gera distorções e menores níveis de extrusão na imagem final.

As Figuras 4.1, 4.2, 4.3, 4.4, 4.5, Figuras 4.6, 4.7, 4.8, 4.9 e 4.10 apresentadas nesse trabalho permitem ter uma visão geral da aplicação das técnicas de simulação de malhas detalhadas a diferentes tipos de superfícies, a partir de dois pontos diferentes de visão. No entanto, a resolução das figuras pode tornar difícil a visualização de diferenças entre as imagens geradas para algumas das superfícies. Essas diferenças podem ser percebidas com facilidade quando as imagens são observadas em monitores, ou em outros dispositivos com alta resolução. As Figuras 4.11, 4.12 e 4.13 foram geradas utilizando um *zoom* para visualização de uma pequena parte de uma das superfícies avaliadas. Essas figuras permitem visualizar com maior facilidade diferenças entre as técnicas de Relief Mapping, Parallax Occlusion Mapping, Cone Step Mapping e Sphere

Tracing.

Na Figura 4.11 é possível perceber que a técnica de Bump Mapping não consegue adicionar grandes detalhes a superfície, tendo um efeito similar a aplicação de uma textura a superfície. A técnica de Parallax Mapping consegue deixar partes da superfície mais extrusas, mas gera distorções na imagem final. Na Figura 4.12 é possível perceber que a técnica de Relief Mapping gera artefatos na renderização da superfície, apresentando descontinuidades na superfície renderizada. A técnica de Parallax Occlusion Mapping consegue renderizar a superfície com maior precisão, mas ainda podem ser percebidos pequenos cerrilhamentos na imagem. Na Figura 4.13 é possível perceber que a técnica de Cone Step Mapping renderiza a imagem sem a presença de artefatos, e gera uma semi-borda preta em partes da superfície. No entanto, o nível de extrusão da superfície é ligeiramente inferior ao obtido com as técnicas de Relief Mapping, POM e Sphere Tracing. A técnica de Sphere Tracing consegue renderizar a superfície sem a presença de artefatos ou deformações, mas a interpolação incorreta entre cores da superfície reduz a qualidade da imagem gerada.

4.3 Comparação de desempenho

As Tabelas 4.1 e 4.2 apresentam a comparação de desempenho entre as técnicas implementadas na renderização das superfícies apresentadas na Seção 4.2, a partir de dois pontos de vista. A Tabela 4.3 apresenta o desempenho na renderização das 5 superfícies avaliadas, visualizadas a uma distância mediana entre o observador e a superfície. A Tabela 4.3 apresenta o desempenho na renderização das 5 superfícies avaliadas, visualizadas a uma distância mediana entre o observador e a superfície.

É possível observar que a técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997) e Parallax Mapping (WELSH, 2004) possuem um desempenho similar e muito superior às demais técnicas. Isto ocorre devido a ambas as técnicas realizarem apenas uma iteração. Dentre as demais técnicas, as técnicas de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005) e Sphere Tracing (DONNELLY, 2005) também possuem desempenho similar, e superior a técnica de Cone Step Mapping (DUMMER, 2006). Isto ocorre devido a essas técnicas realizarem o mesmo número de iterações, sendo que a técnica de Cone Step Mapping realiza um maior número de cálculos em cada iteração. Por fim, a técnica de Parallax Occlusion Mapping apresenta um desempenho muito inferior às demais técnicas. Isto ocorre devido ao número de iterações da técnica ser variável (entre 8 a 32 iterações), o que requer o tratamento de desvios dinâmicos pelos *hardwares* gráficos e resulta em um alto tempo de processamento. Mesmo nas

renderizações de superfícies observadas em ângulos próximos de 0 graus, que faria o número de iterações de cada raio ser aproximadamente 8, a técnica apresenta um desempenho muito inferior as demais técnicas.

Todas as técnicas comparadas podem ser utilizadas na renderização em tempo real de malhas detalhadas, visualizadas a distâncias próximas ou medianas. No entanto, a técnica de POM requer *hardwares* gráficos modernos para que a mesma possa ser aplicada em tempo real.

A partir dos dados de desempenho das técnicas é possível perceber que as técnicas mais precisas demandam um alto tempo de processamento, por isso pode ser necessário reduzir a qualidade visual do ambiente criado para conseguir uma interação em tempo real com o mesmo.

	1º Sup.	2º Sup.	3º Sup.	4º Sup.	5º Sup.
Bump Mapping	805	818	809	810	817
Parallax Mapping	805	809	803	804	788
Relief Mapping	300	269	280	246	180
POM	35	29	36	32	23
Cone Step Mapping	208	199	204	180	139
Sphere Tracing	318	268	293	219	166

Tabela 4.1: Comparação de desemepenho entre as técnicas de simulação de superfícies detalhadas, na renderização das superfícies apresentadas nas Figuras 4.1, 4.2, 4.3, 4.4, 4.5. Os valores apresentados representam o desempenho de cada uma das técnicas medido em quadros por segundo (fps).

	1º Sup.	2º Sup.	3º Sup.	4º Sup.	5º Sup.
Bump Mapping	480	479	480	480	480
Parallax Mapping	419	420	420	417	416
Relief Mapping	134	130	130	123	110
POM	11	18	10	12	8
Cone Step Mapping	86	86	85	83	76
Sphere Tracing	143	142	140	128	113

Tabela 4.2: Comparação de desemepenho entre as técnicas de simulação de superfícies detalhadas, na renderização das superfícies apresentadas nas Figuras 4.6, 4.7, 4.8, 4.9, 4.10. Os valores apresentados representam o desempenho de cada uma das técnicas medido em quadros por segundo (fps).

(a) Bump Mapping

(b) Parallax Mapping

(c) Relief Mapping

(d) Parallax Occlusion Mapping

(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.1: Comparação entre as técnicas de renderização para a primeira superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância mediana. (a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).

(a) Bump Mapping

(b) Parallax Mapping

(c) Relief Mapping

(d) Parallax Occlusion Mapping

(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.2: Comparação entre as técnicas de renderização para a segunda superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância mediana. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).

(a) Bump Mapping

(b) Parallax Mapping

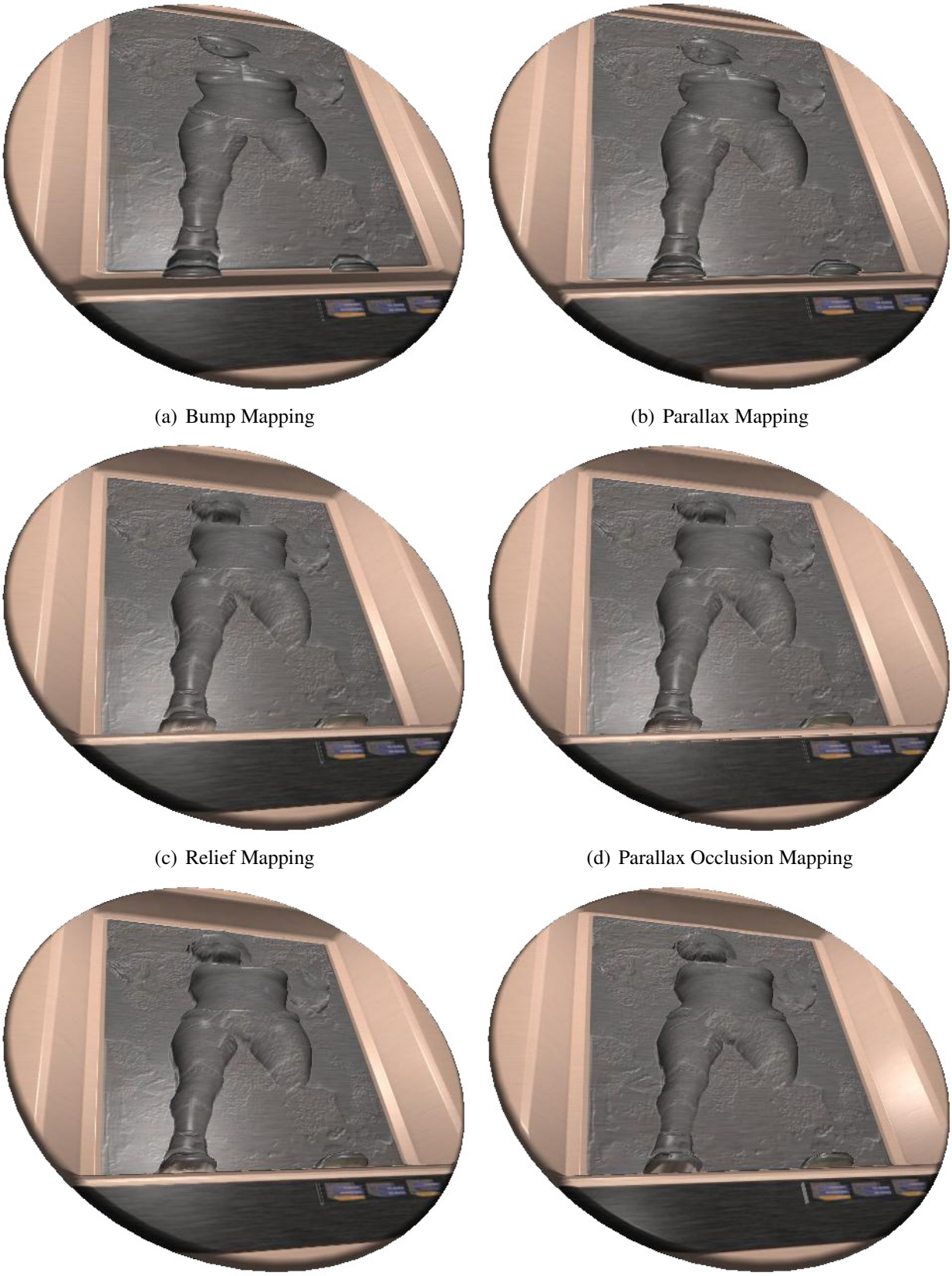
(c) Relief Mapping

(d) Parallax Occlusion Mapping

(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.3: Comparação entre as técnicas de renderização para a terceira superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância mediana. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).



(a) Bump Mapping

(b) Parallax Mapping

(c) Relief Mapping

(d) Parallax Occlusion Mapping

(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.4: Comparação entre as técnicas de renderização para a quarta superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância mediana. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).

(a) Bump Mapping

(b) Parallax Mapping

(c) Relief Mapping

(d) Parallax Occlusion Mapping

(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.5: Comparação entre as técnicas de renderização para a quinta superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância mediana. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).

(a) Bump Mapping

(b) Parallax Mapping

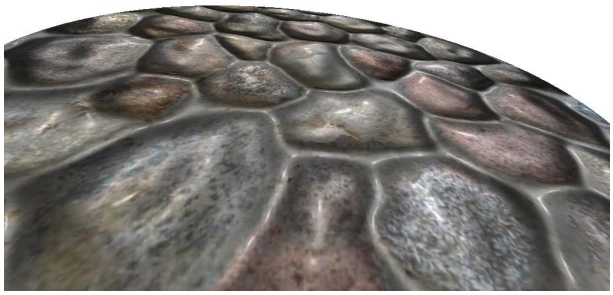
(c) Relief Mapping

(d) Parallax Occlusion Mapping

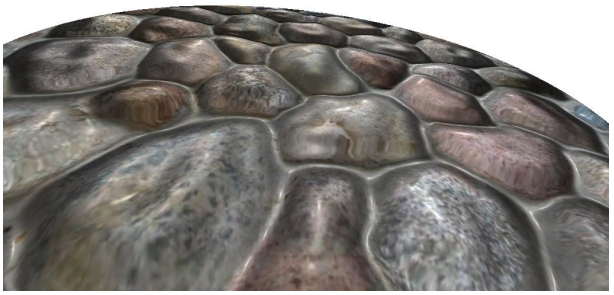
(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.6: Comparação entre as técnicas de renderização para a primeira superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância extremamente próxima. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).



(a) Bump Mapping

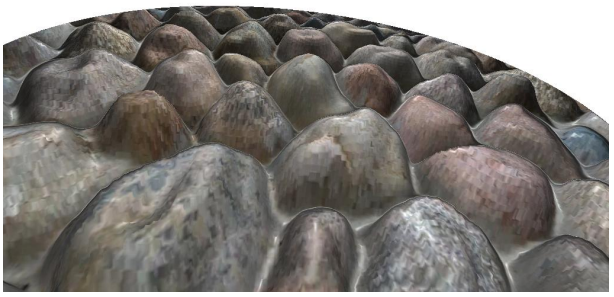


(b) Parallax Mapping

(c) Relief Mapping

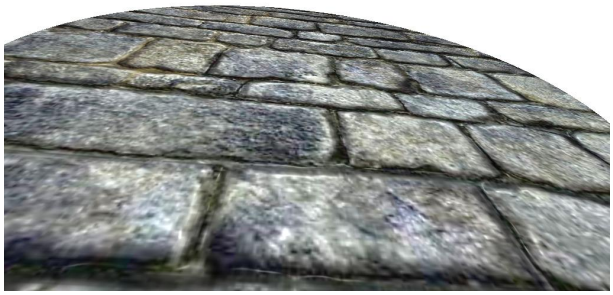
(d) Parallax Occlusion Mapping

(e) Cone Step Mapping

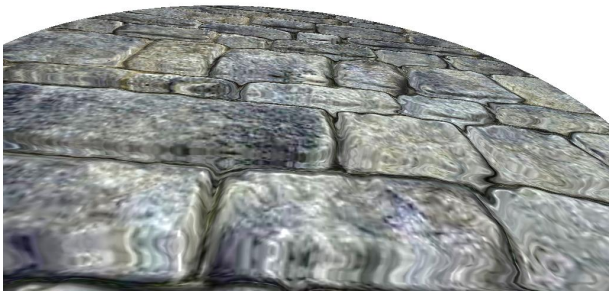


(f) Sphere Tracing

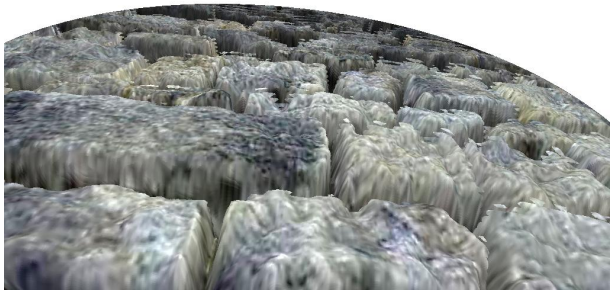
Figura 4.7: Comparação entre as técnicas de renderização para a segunda superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância extremamente próxima. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).



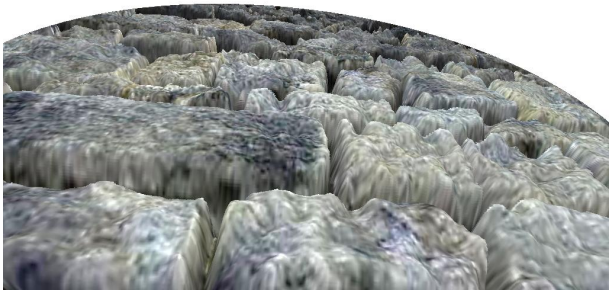
(a) Bump Mapping



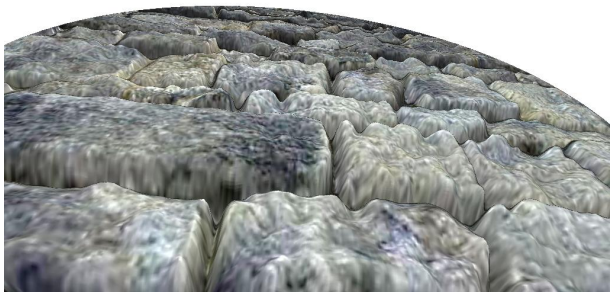
(b) Parallax Mapping



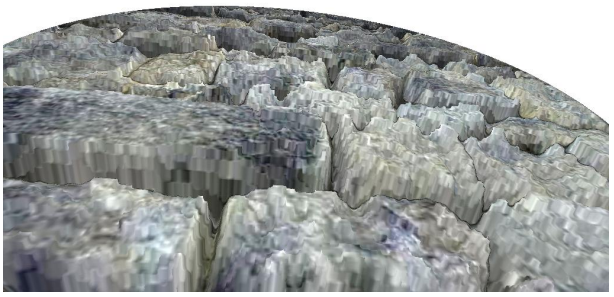
(c) Relief Mapping



(d) Parallax Occlusion Mapping

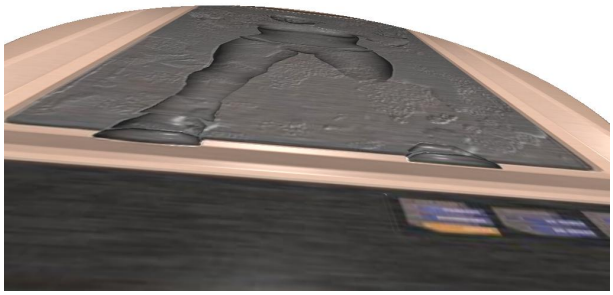


(e) Cone Step Mapping

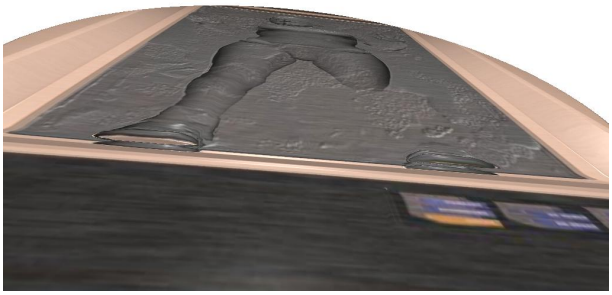


(f) Sphere Tracing

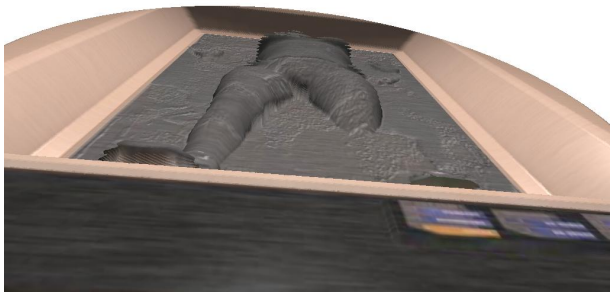
Figura 4.8: Comparação entre as técnicas de renderização para a terceira superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância extremamente próxima. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).



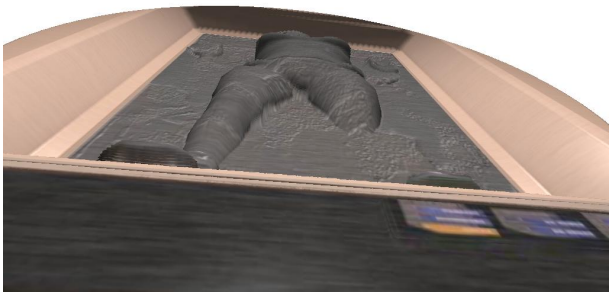
(a) Bump Mapping



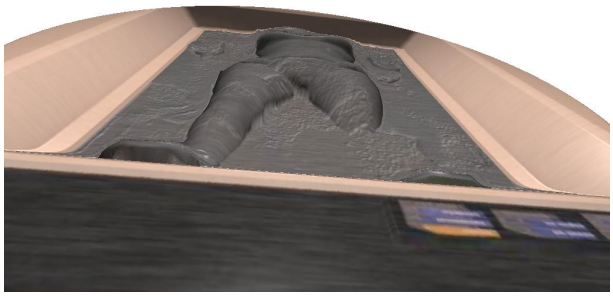
(b) Parallax Mapping



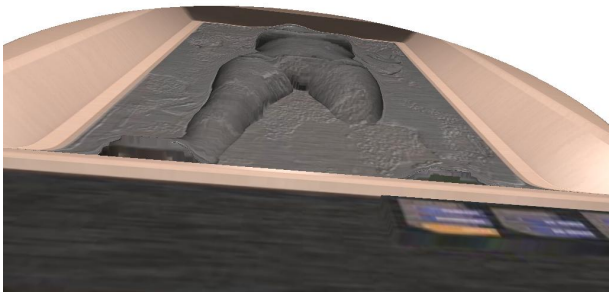
(c) Relief Mapping



(d) Parallax Occlusion Mapping



(e) Cone Step Mapping

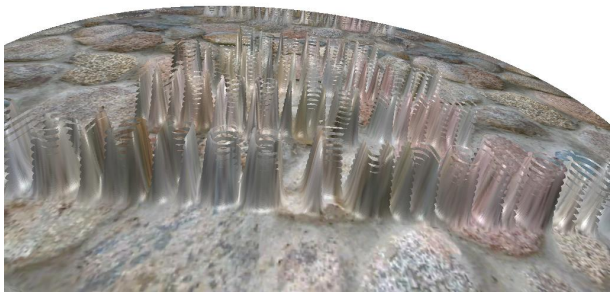


(f) Sphere Tracing

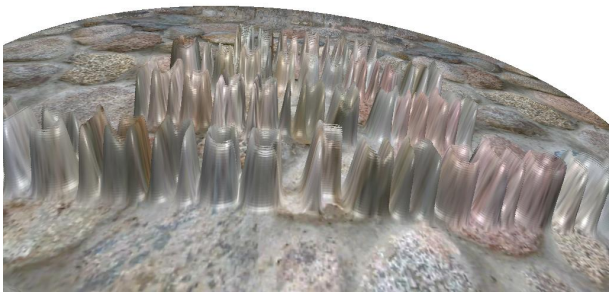
Figura 4.9: Comparação entre as técnicas de renderização para a quarta superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância extremamente próxima. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).

(a) Bump Mapping

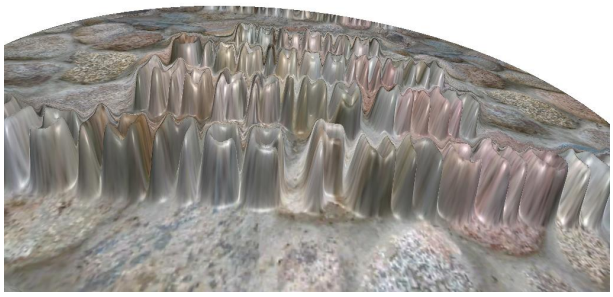
(b) Parallax Mapping



(c) Relief Mapping



(d) Parallax Occlusion Mapping

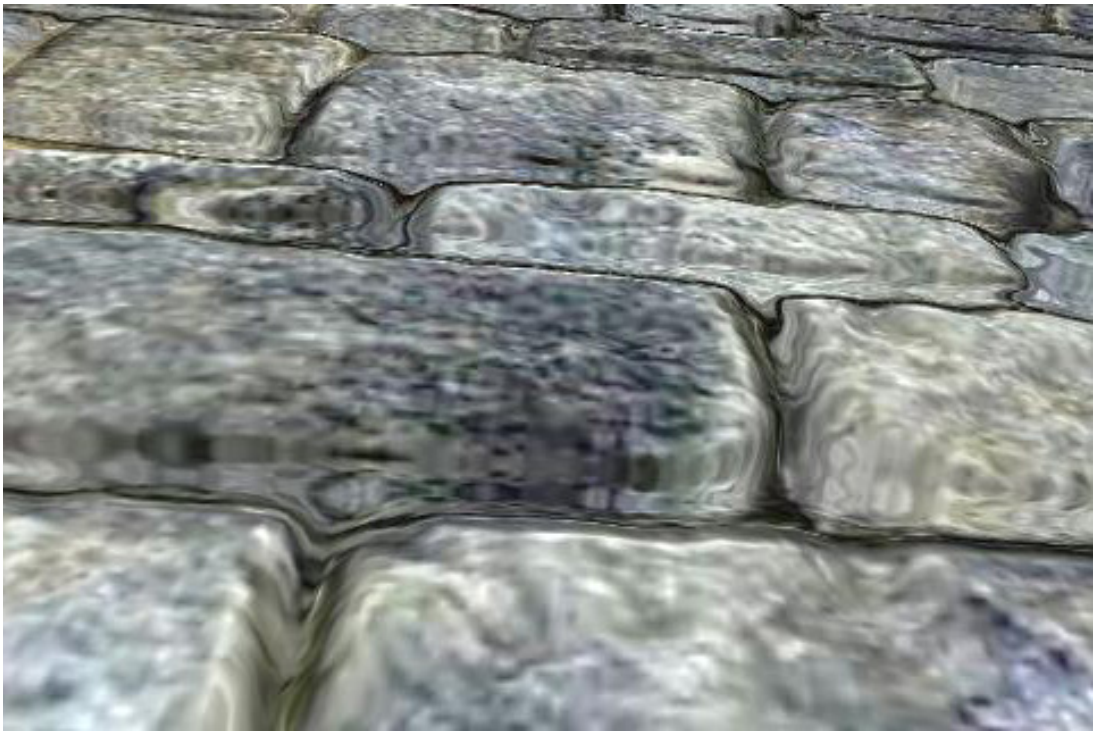


(e) Cone Step Mapping

(f) Sphere Tracing

Figura 4.10: Comparação entre as técnicas de renderização para a quinta superfície avaliada. Imagens renderizadas da superfície visualizada a uma distância extremamente próxima. a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004), (c) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (d) Parallax Occlusion Mapping (TATARCHUK, 2006), (e) Cone Step Mapping (DUMMER, 2006), (f) Sphere Tracing (DONNELLY, 2005).

(a)



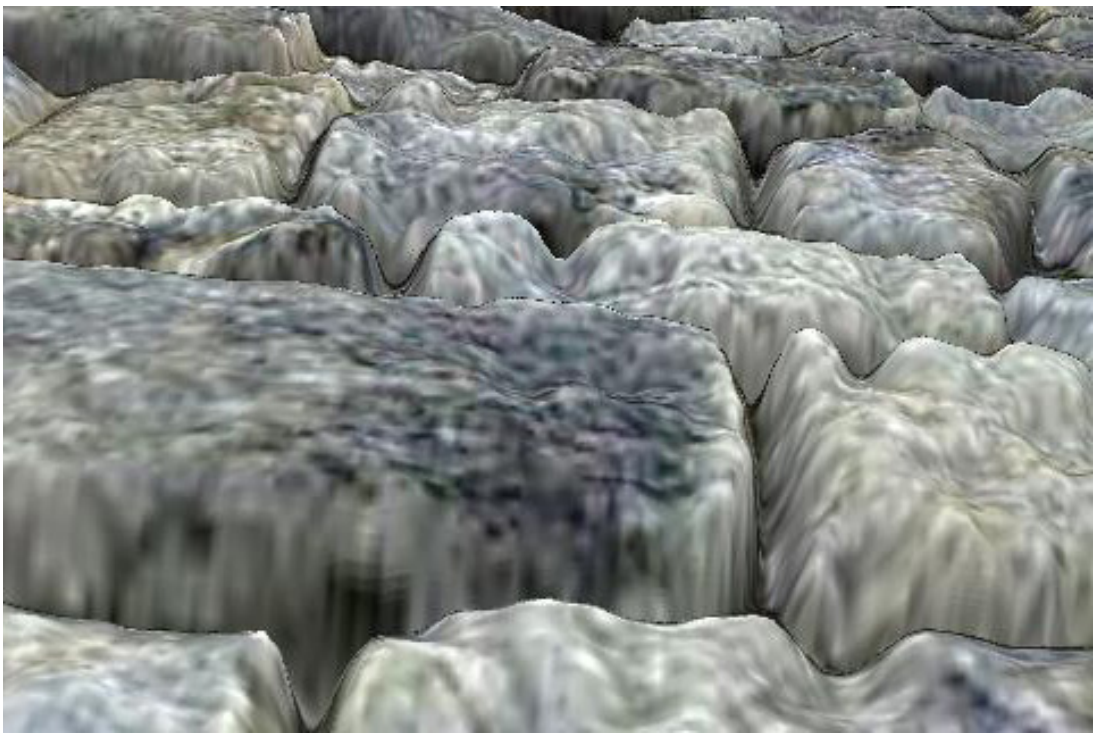
(b)

Figura 4.11: Comparação entre as técnicas de renderização para a terceira superfície Zoom em parte da imagem gerada a partir da renderização da superfície visualizada a uma distância extremamente próxima. (a) Bump Mapping (PEERCY; AIREY; CABRAL, 1997), (b) Parallax Mapping (WELSH, 2004).

(a)

(b)

Figura 4.12: Comparação entre as técnicas de renderização para a terceira superfície avaliada. Zoom em parte da imagem gerada a partir da renderização da superfície visualizada a uma distância extremamente próxima. (a) Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), (b) Parallax Occlusion Mapping (TATARCHUK, 2006).



(a)

(b)

Figura 4.13: Comparação entre as técnicas de renderização para a terceira superfície avaliada. Zoom em parte da imagem gerada a partir da renderização da superfície visualizada a uma distância extremamente próxima. (a) Cone Step Mapping (DUMMER, 2006), (b) Sphere Tracing (DONNELLY, 2005).

5 *Conclusão e trabalhos futuros*

Neste trabalho foi apresentada uma análise e comparação entre as principais técnicas existentes para simulação de malhas detalhadas. Na análise dessas técnicas, foi apresentado o funcionamento de cada uma das técnicas, efeitos visuais tratados, vantagens e desvantagens de sua utilização e ambiente e situação para uso eficaz das técnicas. Na comparação das técnicas, foi apresentada uma comparação entre as imagens geradas e desempenho obtido com cada uma das técnicas analisadas, que puderam ser implementadas. As técnicas apresentadas podem ser utilizadas como solução para o problema de renderização de malhas detalhadas, com a presença de micro e macro deformações, em tempo real.

A técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997) é a mais simples das técnicas analisadas e consegue adicionar pequenos detalhes as superfícies, como ilustrado na Seção 3.3.1. Essa técnica possui um alto desempenho e pode ser utilizada sem a necessidade da aquisição de *hardwares* gráficos modernos. Atualmente essa técnica ainda é muito utilizada em jogos.

A técnica de Parallax Mapping (WELSH, 2004) é implementada em conjunto com a técnica de Bump Mapping e consegue aumentar a deformação, ou extrusão, das superfícies. Essa técnica é uma heurística e sua aplicação geralmente é restrita a mapas com padrões de repetição, como paredes de tijolos. Esta técnica também é utilizadas em jogos.

A técnica de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005) é uma técnica mais precisa que consegue adicionar maiores detalhes as superfícies. No entanto, essa técnica exige a aquisição de *hardwares* gráficos modernos. Esta técnica foi publicada recentemente, mas já é utilizadas em jogos de computador de última geração, como Quake 4.

A técnica de Parallax Occlusion Mapping (TATARCHUK, 2006) é a técnica mais precisa para adição de detalhes em superfícies. Essa técnica trata a renderização das superfícies da forma mais precisa possível, sem se preocupar com o desempenho. Como resultado, utilizando um grande número mínimo e máximo de amostras, por exemplo, 16 e 128, essa técnica gera imagens extremamente similares a renderização das malhas reais. O desempenho obtido com

essa técnica é muito inferior às demais técnicas apresentadas.

A técnica de Sphere Tracing (DONNELLY, 2005) foi proposta recentemente e aparece como uma solução alternativa a técnica de Relief Mapping para simulação de superfícies detalhadas. Essa técnica elimina a presença de artefatos nas imagens geradas, que ocorre na técnica de Relief Mapping, mas insere pequenas deformações na mesma. O problema dessa técnica é a necessidade de geração e armazenamento do mapa de esferas para cada uma das superfícies.

A técnica de Cone Step Mapping (DUMMER, 2006) foi outra técnica proposta recentemente como uma solução alternativa a técnica de Relief Mapping. Assim como a técnica de Relief Mapping, essa técnica elimina a presença de artefatos nas imagens, mas insere pequenas deformações na mesma. Essa técnica consome a mesma memória que a técnica de Relief Mapping, mas consegue gerar imagens com qualidade visual superior as técnicas de Relief Mapping e Sphere Tracing. A desvantagem dessa técnica é demandar um tempo de processamento um pouco superior a técnica de Relief Mapping.

Em resumo, alguns pontos de vistas e tipos de ambientes (interativos ou não interativos) recomendados para uso das técnicas são apresentados na Tabela 5.

	Distância	Ângulo de Observação	Tipo de Ambiente
Bump Mapping	Grande	Pequeno	Interativo
Parallax Mapping	Grande	Pequeno	Interativo
Relief Mapping	Pequena	Medio	Interativo
Parallax Occlusion Mapping	Pequena	Grande	Não interativo
Cone Step Mapping	Pequena	Grande	Interativo
Sphere Tracing	Pequena	Grande	Interativo

Tabela 5.1: Pontos de vista e ambiente recomendados para uso de cada uma das técnicas.

Nenhuma das técnicas comparadas trata corretamente o efeito de silhuetas dos objetos. Para tratar esse efeito é necessário considerar a curvatura dos objetos, o que não pode ser feito quando utiliza-se o espaço da tangente. No entanto, não utilizar o espaço da tangente, implica em tratar apenas superfícies planas, onde seria necessário estender a técnica para tratar superfícies curvas. Devido a esse problema torna-se mais simples não tratar a silhueta dos objetos e fazer com que o mesmo possa ser aplicado a qualquer tipo de superfície.

A partir da análise e comparação das técnicas apresentadas pode-se concluir que nenhuma das técnicas apresenta uma solução genérica para simulação de malhas detalhadas. Cada uma das técnicas apresentadas possui uma diferente relação entre qualidade visual e desempenho e podem tratar diferentes efeitos visuais como *motion parallax*, *self-shadow*, *self-occlusion* e silhuetas.

Outra solução para a renderização de superfícies detalhadas seria a criação de *hardwares* específicos para o tratamento da técnica de Displacement Mapping (COOK, 1984), permitindo assim a renderização das superfícies reais em tempo real (BLYTHE, 2006).

Referências Bibliográficas

APODACA, A. A.; GRITZ, L. *Advanced RenderMan: beyond the Companion*. [S.l.]: Morgan Kaufmann, 2000. APO a 00:1 1.Ex. ISBN 1558606181.

BECKER, B. G.; MAX, N. L. Smooth transitions between bump rendering algorithms. In: . [S.l.: s.n.]. p. 183–190.

BEN-ARTZI, A.; OVERBECK, R.; RAMAMOORTHI, R. Real-time brdf editing in complex lighting: Copyright restrictions prevent acm from providing the full text for this work. In: *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. New York, NY, USA: ACM Press, 2006. p. 54. ISBN 1-59593-364-6.

BLINN, J. F. Simulation of wrinkled surfaces. *Computer Graphics (Proceedings of SIGGRAPH 78)*, v. 12, n. 3, p. 286–292, August 1978. Held in Atlanta, Georgia.

BLYTHE, D. The direct3d 10 system. In: *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*. New York, NY, USA: ACM Press, 2006. p. 724–734. ISBN 1-59593-364-6.

BURGER, P.; GILLIES, D. *Interactive Computer Graphics. Functional, Procedural and Device-Level Methods*. [S.l.]: Addison-Wesley, 1989. BUR p 89:1 1.Ex. ISBN 0-201-17439-1.

COOK, R. L. Shade trees. In: *Computer Graphics and Interactive Techniques*. [S.l.: s.n.], 1984. v. 18, n. 3.

COOK, R. L.; TORRANCE, K. E. A reflectance model for computer graphics. v. 1, n. 1, p. 7–24, jan. 1982. ISSN 0730-0301.

DONNELLY, W. Per-pixel displacement mapping with distance functions. In: ____. [S.l.]: Addison-Wesley, 2005. cap. 8.

DUMMER, J. *Cone Step Mapping: An Iterative Ray-Heightfield Intersection Algorithm*. [S.l.], 2006.

EVANGELISTA, B. P. et al. Renderização de cenas tridimensionais não foto-realistas explorando hardware programável. *Sibgrapi WIC*, 2005.

FOLEY, J. et al. *Computer Graphics. Principles and Practice. 2nd Edition in C*. [S.l.]: Addison-Wesley, 1996. FOL j 96:1 1.Ex. ISBN 0-201-84840-6.

HART, J. C. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, v. 12, n. 10, p. 527–545, 1996. Disponível em: <citeseer.ist.psu.edu/hart94sphere.html>.

HASENFRATZ, J.-M. et al. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, v. 22, n. 4, p. 753–774, dec 2003. Disponível em: <http://artis.inrialpes.fr/Publications/2003/HLHS03a>.

HEIDRICH, W. et al. Illuminating micro geometry based on precomputed visibility. *Proceedings of SIGGRAPH 2000*, ACM Press / ACM SIGGRAPH / Addison Wesley Longman, p. 455–464, July 2000. ISBN 1-58113-208-5.

HOWARD, I.; ROGERS, B. *Binocular Vision and Stereopsis*. [S.l.]: Oxford University Press, 1995.

KAJIYA, J. T. The rendering equation. In: *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1986. p. 143–150. ISBN 0-89791-196-2.

KANEKO, T. et al. Detailed shape representation with parallax mapping. *Proceedings of the ICAT2001 (11th International Conference on Artificial Reality and Tele-Existence)*, 2001.

LENGYEL, E. Bump mapping construction. In: *Mathematics for 3D game programming and computer graphics, Second Edition*. [S.l.]: Charles River media, November 2003. ISBN 1-58450-277-0.

OLIVEIRA, M. M.; BISHOP, G.; MCALLISTER, D. Relief texture mapping. *Proceedings of SIGGRAPH*, 2000.

PEERCY, M.; AIREY, J.; CABRAL, B. Efficient bump mapping hardware. *Computer Graphics*, v. 31, n. Annual Conference Series, p. 303–306, 1997. Disponível em: <citeseer.ist.psu.edu/peercy97efficient.html>.

POLICARPO, F.; OLIVEIRA, M. M.; COMBA, J. L. D. Real-time relief mapping on arbitrary polygonal surfaces. *I3D*, 2005.

RISSER, E.; SHAH, M.; PATTANAIK, S. *Interval Mapping*. [S.l.], 2006.

SILVA, A. R. da; MARTINS, C. A. P. da S. Uma implementação híbrida de raytracing em processadores gráficos programáveis ou processadores de propósito geral. *IEEE Sibgrapi WIC*, 2005.

SLOAN, P.-P.; COHEN, M. F. Interactive horizon mapping. *Eurographics Rendering Workshop*, June 2000.

TATARCHUK, N. Dynamic parallax occlusion mapping with approximate soft shadows. In: *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM Press, 2006. p. 63–69. ISBN 1-59593-295-X.

WANG, L. et al. View-dependent displacement mapping. *ACM Trans. Graph.*, ACM Press, New York, NY, USA, v. 22, n. 3, p. 334–339, 2003. ISSN 0730-0301.

WANG, L. et al. View-dependent displacement mapping. *ACM Trans. Graph.*, ACM Press, New York, NY, USA, v. 22, n. 3, p. 334–339, 2003. ISSN 0730-0301.

WELSH, T. *Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces*. [S.l.], 2004.

WOO, A.; POULIN, P.; FOURNIER, A. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, v. 10, n. 6, p. 13–32, nov. 1990. ISSN 0272-1716.

APÊNDICE A – Funções utilizadas

Lista das funções utilizadas nos algoritmos em alto nível apresentados na Seção 3.

readTexture2D(a, b)

a Textura 2D a ser lida

b Endereço da textura a ser lido

Lê uma posição da textura e retorna o valor daquela posição.

readTexture3D(a, b)

a Textura 3D a ser lida

b Endereço da textura a ser lido

Lê uma posição da textura e retorna o valor daquela posição.

phongEquation(n, l, v, c)

n Vetor normal da superfície

l Vetor direção da luz

v Vetor direção de visão

c Cor difusa da superfície

Calcula a cor do *pixel* utilizando a equação de iluminação de Phong (BURGER; GILLIES, 1989).

linearSearch(p, v, t)

p Posição inicial da busca linear

v Vetor direção com tamanho do passo da busca

t Mapa de profundidade da superfície

Calcula a nova posição da busca linear.

binarySearch(p, v, t)

p Posição inicial da busca linear

v Vetor direção com tamanho do passo da busca

t Mapa de profundidade da superfície

Calcula a nova posição da busca binária.

calculateMipmap(uv)

uv Coordenada de textura

Calcula o nível de mipmap de um *pixel*.

dotProduct(a, b)

a Vetor A

b Vetor B

Calcula o produto escalar entre os vetores a e b.

lerp(min, max, fat)

min Valor mínimo

max Valor máximo

fat Fator de interpolação

Calcula a interpolação linear entre um valor mínimo e um máximo.

calculateConeCollisionPoint(ch, cr, rp, rd)

ch Altura do cone

cr Raio do cone

rp Poisção inicial do raio

rd Direção do raio

Calcula o ponto de colisão entre um raio e um cone.

APÊNDICE B – Implementações das técnicas

B.1 Declaração de variáveis

```
//-----------------------------------------------------------------------------------
// File: BumpTechniques.fx
// Autor: Bruno P. Evangelista (www.brunoevangelista.com)
//
// This shader implements many techniques in order to simulate detailed surfaces
//
// The techniques currently implemented are:
//  - Bump Mapping
// - Parallax Mapping
// - Relief Mapping
// - Parallax Occlusion Mapping
// - Cone Step Mapping
// - Sphere Mapping
//
//-----------------------------------------------------------------------------------

#define NUM_ITERATIONS 16
#define NUM_ITERATIONS_RELIEF1 11
#define NUM_ITERATIONS_RELIEF2 5

float3 lightPos;
float3 ambientColor = {0.3, 0.3, 0.3};
float3 diffuseColor = {0.8, 0.8, 0.8};
float3 specularColor = {0.3, 0.3, 0.3};
float shine = 64.0;
float tile = 1.0;

float depth = 0.1;
bool depth_bias = false;

// POM Configurations
```

```
int LODThreshold = 3;
int minSamples = (int)(NUM_ITERATIONS * 0.5);
int maxSamples = (int)(NUM_ITERATIONS * 2);

// Texture dimensions
int TEXTURE_WIDTH;
int TEXTURE_HEIGHT;
int TEXTURE_DEPTH;

texture color_map_tex;
texture cone_map_tex;
texture cone_map3D_tex;
texture sphere_map_tex;

sampler2D color_map = sampler_state
{
Texture = <color_map_tex>;
MinFilter = Linear;
MagFilter = Linear;
MipFilter = Linear;
};

sampler2D cone_map = sampler_state
{
Texture = <cone_map_tex>;
MinFilter = Linear;
MagFilter = Linear;
MipFilter = Linear;
};

sampler3D cone_map3D = sampler_state
{
Texture = <cone_map3D_tex>;
MinFilter = Linear;
MagFilter = Linear;
MipFilter = Linear;
};

sampler3D sphere_map = sampler_state
{
Texture = <sphere_map_tex>;
MinFilter = Linear;
MagFilter = Linear;
```

```
MipFilter = Linear;
};

// Matrix
// -----------------------------------------
float4x4 matWVP : WorldViewProjection;
float4x4 matW  : World;
float4x4 matVI  : ViewInverse;

// Structs
// -----------------------------------------
struct vertexInput
{
    float4 pos       : POSITION;
    float3 normal    : NORMAL;
    float2 uv0  : TEXCOORD0;
    float3 tangent   : TANGENT0;
    float3 binormal  : BINORMAL0;
};

struct pixelInput
{
float4 hpos : POSITION;
float2 uv0  : TEXCOORD0;
float3 eyeVec : TEXCOORD1;
float3 lightVec : TEXCOORD2;
};

struct POM_VS_OUTPUT
{
    float4 position          : POSITION;
    float2 texCoord          : TEXCOORD0;
    float3 vLightTS          : TEXCOORD1;
    float3 vViewTS           : TEXCOORD2;
    float2 vParallaxOffsetTS : TEXCOORD3;
    float3 vNormalWS         : TEXCOORD4;
    float3 vViewWS           : TEXCOORD5;

};

struct POM_PS_INPUT
{
   float2 texCoord          : TEXCOORD0;
```

```
   float3 vLightTS          : TEXCOORD1;
   float3 vViewTS           : TEXCOORD2;
   float2 vParallaxOffsetTS : TEXCOORD3;
   float3 vNormalWS         : TEXCOORD4;
   float3 vViewWS           : TEXCOORD5;
};
```

B.2 Processamento de vértices

Processamento de vértices utilizado nas técnicas de Bump Mapping (PEERCY; AIREY; CABRAL, 1997), Parallax Mapping (WELSH, 2004), Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005), Cone Step Mapping (DUMMER, 2006) e Sphere Tracing (DONNELLY, 2005).

```
pixelInput bump_vs(vertexInput IN)
{
    pixelInput OUT;

    // Vertex in clip space
    OUT.hpos = mul(IN.pos, matWVP);

    // Copy texcoord
    OUT.uv0 = IN.uv0;

    // Tangent space (Tangent, binormal, normal)
    float3x3 tangentMap = float3x3(IN.tangent, IN.binormal, IN.normal);
    tangentMap = mul(tangentMap, matW);

// World vertex
float3 vertexPos = mul(IN.pos, matW);

// View vector
float3 eyeVec = vertexPos - matVI[3].xyz;
OUT.eyeVec = mul(tangentMap, eyeVec);

// Light vector
float3 lightVec = lightPos - vertexPos;
OUT.lightVec = mul(tangentMap, lightVec);

    return OUT;
}
```

B.3 Bump Mapping

Processamento de *pixels* utilizado na técnica de Bump Mapping (PEERCY; AIREY; CABRAL, 1997).

```
float4 bump_ps(float2 uv0, float3 eyeVec, float3 lightVec) : COLOR
{
// View and light vectors
float3 v = normalize(eyeVec);
float3 l1 = normalize(lightVec);

// Diffuse texture color
float3 color = tex2D(color_map, uv0).xyz;

// Bump map
float3 n = tex2D(cone_map, uv0);
n.xy = n.xy * 2.0 - 1.0;
n.z = sqrt(1.0 - dot(n.xy, n.xy));

// Attenuation
float att1 = 1.0 - max(0, l1.z);
att1 = 1.0 - att1*att1;
// Diffuse
float diffInt1 = saturate(dot(l1, n));
// Specular
float3 h1 = normalize(l1-v);
float specInt1 = pow(saturate(dot(h1, n)), shine);

float4 finalColor;
finalColor.xyz = ambientColor*color +
att1*( diffInt1*color*diffuseColor + specInt1*specularColor);
finalColor.w = 1.0f;

return finalColor;
}

float4 normal_ps(pixelInput IN) : COLOR
{
return bump_ps(IN.uv0, IN.eyeVec, IN.lightVec);
}

technique normal_mapping
{
    pass p0
```

```
    {
VertexShader = compile vs_1_1 bump_vs();
PixelShader  = compile ps_2_a normal_ps();
    }
}
```

B.4 Parallax Mapping

Processamento de *pixels* utilizado na técnica de Parallax Mapping (WELSH, 2004).

```
float4 parallax_ps(pixelInput IN) : COLOR
{
float offset = tex2D(cone_map, IN.uv0).w
* 0.06 - 0.015;
IN.uv0 += normalize(IN.eyeVec) * offset;

return bump_ps(IN.uv0, IN.eyeVec, IN.lightVec);
}

technique parallax_mapping
{
    pass p0
    {
VertexShader = compile vs_1_1 bump_vs();
PixelShader  = compile ps_2_a parallax_ps();
    }
}
```

B.5 Relief Mapping

Processamento de *pixels* utilizado na técnica de Relief Mapping (POLICARPO; OLIVEIRA; COMBA, 2005).

```
void ray_intersect_relief(inout float3 p,inout float3 v)
{
v /= v.z*NUM_ITERATIONS_RELIEF1;

int i;
for( i=0;i<NUM_ITERATIONS_RELIEF1;i++ )
{
float4 tex = tex2D(cone_map, p.xy);
```

```
if (p.z<tex.w)
p+=v;
}

for( i=0;i<NUM_ITERATIONS_RELIEF2;i++ )
{
v *= 0.5;
float4 tex = tex2D(cone_map, p.xy);
if (p.z<tex.w)
p+=v;
else
p-=v;
}
}

float4 relief_ps(pixelInput IN) : COLOR
{
float3 rayPos, rayVec;

rayPos = float3(IN.uv0, 0.0);
rayVec = normalize(IN.eyeVec);
rayVec.z = abs(rayVec.z);
rayVec.xy *= depth;

ray_intersect_relief(p.xyz, v);
return bump_ps(p.xy, IN.eyeVec, IN.lightVec);
}

technique relief_mapping
{
    pass p0
    {
VertexShader = compile vs_1_1 bump_vs();
PixelShader  = compile ps_2_a relief_ps();
    }
}
```

B.6 Cone Step Mapping

Processamento de *pixels* utilizado na técnica de Cone Step Mapping (DUMMER, 2006).

```
void ray_intersect_cone(inout float3 p,inout float3 v)
```

```
{
float dist=length(v.xy);

for( int i=0;i<NUM_ITERATIONS; i++ )
{
float4 tex = tex2D(cone_map, p.xy);
float height = max(0, tex.w - p.z);
float cone_ratio = tex.z*tex.z;

float stepDist = height * cone_ratio / (v.z*cone_ratio + dist);
p += v * stepDist;
}
}

float4 cone_ps(pixelInput IN) : COLOR
{
float3 rayPos, rayVec;

rayPos = float3(IN.uv0, 0.0);
rayVec = normalize(IN.eyeVec);
rayVec.z = abs(rayVec.z);
rayVec.xy *= depth;

ray_intersect_cone(p,v);
return bump_ps(p.xy, IN.eyeVec, IN.lightVec);
}

technique cone_mapping
{
    pass p0
    {
VertexShader = compile vs_1_1 bump_vs();
PixelShader  = compile ps_2_a cone_ps();
    }
}
```

B.7 Sphere Tracing

Processamento de *pixels* utilizado na técnica de Sphere Tracing (DONNELLY, 2005).

```
void ray_intersect_sphere(inout float3 p,inout float3 v)
{
```

```
for(int i = 0; i < NUM_ITERATIONS; i++)
{
float stepDist = tex3D(sphere_map, p).x;
p += v * stepDist;
}
}

float4 sphere_ps(pixelInput IN) : COLOR
{
float3 p,v;

p = float3(IN.uv0, 0);

// Texture width MUST be equal to texture height
IN.eyeVec.xy *= depth * 15.5f/256.0f *
TEXTURE_WIDTH;

v = normalize(IN.eyeVec);
v.z = abs(v.z);
v.xy *= TEXTURE_DEPTH/(float)TEXTURE_WIDTH;

ray_intersect_sphere(p,v);

return bump_ps(p.xy, IN.eyeVec, IN.lightVec);
}

technique sphere_mapping
{
    pass p0
    {
VertexShader = compile vs_1_1 bump_vs();
PixelShader  = compile ps_2_a sphere_ps();
    }
}
```

B.8 Parallax Occlusion Mapping

Processamento de vértices utilizado na técnica de Parallax Occlusion Mapping (TATARCHUK, 2006).

```
POM_VS_OUTPUT bump_vs_pom(vertexInput IN)
{
    POM_VS_OUTPUT OUT;
```

```
    // Vertex in clip space
    OUT.position = mul(IN.pos, matWVP);

    // Copy texcoord
    OUT.texCoord = IN.uv0;

    // Tangent space (Tangent, binormal, normal)
    float3x3 tangentMap = float3x3(IN.tangent, IN.binormal, IN.normal);
    tangentMap = mul(tangentMap, matW);

    // Normal WS
    OUT.vNormalWS = mul(IN.normal, matW);

// World vertex
float3 vertexPos = mul(IN.pos, matW);

// View vector
float3 eyeVec = matVI[3].xyz - vertexPos;
OUT.vViewWS = eyeVec;
OUT.vViewTS = mul(tangentMap, eyeVec);

// Light vector
float3 lightVec = lightPos - vertexPos;
OUT.vLightTS = mul(tangentMap, lightVec);

// Compute initial parallax displacement direction:
    float2 vParallaxDirection = normalize(  OUT.vViewTS.xy );

    // The length of this vector determines the furthest amount of displacement:
    float fLength         = length( OUT.vViewTS );
    float fParallaxLength = sqrt( fLength * fLength - OUT.vViewTS.z * OUT.vViewTS.z ) / OUT.vViewT

    // Compute the actual reverse parallax displacement vector:
    OUT.vParallaxOffsetTS = vParallaxDirection * fParallaxLength;
    OUT.vParallaxOffsetTS *= depth;

    return OUT;
}
```

Processamento de *pixels* utilizado na técnica de Parallax Occlusion Mapping (TATARCHUK, 2006).

```
float4 bump_ps_pom( POM_PS_INPUT i ) : COLOR0
```

```
{
   //  Normalize the interpolated vectors:
   float3 vViewTS   = normalize( i.vViewTS  );
   float3 vViewWS   = normalize( i.vViewWS  );
   float3 vLightTS  = normalize( i.vLightTS );
   float3 vNormalWS = normalize( i.vNormalWS );

   float4 cResultColor = float4( 0, 0, 0, 1 );

   // Compute the current gradients:
   float2 fTexCoordsPerSize = i.texCoord * float2(TEXTURE_WIDTH, TEXTURE_HEIGHT);

   // Compute all 4 derivatives in x and y
   float2 dxSize, dySize;
   float2 dx, dy;
   float4( dxSize, dx ) = ddx( float4( fTexCoordsPerSize, i.texCoord ) );
   float4( dySize, dy ) = ddy( float4( fTexCoordsPerSize, i.texCoord ) );

   float  fMipLevel;
   float  fMipLevelInt;
   float  fMipLevelFrac;

   float  fMinTexCoordDelta;
   float2 dTexCoords;

   // Find min of change in u and v across quad: compute du and dv magnitude across quad
   dTexCoords = dxSize * dxSize + dySize * dySize;

   // Standard mipmapping uses max here
   fMinTexCoordDelta = max( dTexCoords.x, dTexCoords.y );

   // Compute the current mip level  (* 0.5 is effectively computing a square root before )
   fMipLevel = max( 0.5 * log2( fMinTexCoordDelta ), 0 );

   // Start the current sample located at the input texture coordinate, which would correspond
   // to computing a bump mapping result:
   float2 texSample = i.texCoord;

   if ( fMipLevel <= (float) LODThreshold )
   {
      int nNumSteps = (int) lerp( maxSamples, minSamples, dot( vViewWS, vNormalWS ) );

      float fCurrHeight = 0.0;
```

```
float fStepSize   = 1.0 / (float) nNumSteps;
float fPrevHeight = 1.0;
float fNextHeight = 0.0;

int    nStepIndex = 0;
bool   bCondition = true;

float2 vTexOffsetPerStep = fStepSize * i.vParallaxOffsetTS;
float2 vTexCurrentOffset = i.texCoord;
float  fCurrentBound     = 1.0;
float  fParallaxAmount   = 0.0;

float2 pt1 = 0;
float2 pt2 = 0;

float2 texOffset2 = 0;

while ( nStepIndex < nNumSteps )
{
   vTexCurrentOffset -= vTexOffsetPerStep;

   // Sample height map which in this case is stored in the alpha channel of the normal map:
   fCurrHeight = 1.0f - tex2Dgrad( cone_map, vTexCurrentOffset, dx, dy ).a;
   fCurrentBound -= fStepSize;

   if ( fCurrHeight > fCurrentBound )
   {
      pt1 = float2( fCurrentBound, fCurrHeight );
      pt2 = float2( fCurrentBound + fStepSize, fPrevHeight );

      texOffset2 = vTexCurrentOffset - vTexOffsetPerStep;

      nStepIndex = nNumSteps + 1;
      fPrevHeight = fCurrHeight;
   }
   else
   {
      nStepIndex++;
      fPrevHeight = fCurrHeight;
   }
}

float fDelta2 = pt2.x - pt2.y;
```

```
      float fDelta1 = pt1.x - pt1.y;

      float fDenominator = fDelta2 - fDelta1;

      if ( fDenominator == 0.0f )
      {
         fParallaxAmount = 0.0f;
      }
      else
      {
         fParallaxAmount = (pt1.x * fDelta2 - pt2.x * fDelta1 ) / fDenominator;
      }

      float2 vParallaxOffset = i.vParallaxOffsetTS * (1 - fParallaxAmount );

      // The computed texture offset for the displaced point on the pseudo-extruded surface:
      float2 texSampleBase = i.texCoord - vParallaxOffset;
      texSample = texSampleBase;
   }

   cResultColor = bump_ps(texSample, -vViewTS, vLightTS);
   return cResultColor;
}

technique parallax_occlusion_mapping
{
    pass P0
    {
        VertexShader = compile vs_3_0 bump_vs_pom();
        PixelShader  = compile ps_3_0 bump_ps_pom();
    }
}
```