

UNIVERSIDADE FEDERAL DO ABC
CENTRO DE MATEMÁTICA, COMPUTAÇÃO E CONIÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO

Gabriel Nobrega de Lima

Mapeamento de Geometrias a partir de Texturas em Tempo Real

Santo André
2011

Gabriel Nobrega de Lima

Mapeamento de Geometrias a partir de Texturas em Tempo Real

Monografia apresentada ao Curso de Ciência da Computação da UFABC, como requisito para a obtenção parcial do grau de BACHAREL em Ciência da Computação.

Orientador: João Paulo Gois

Doutor em Computação - USP

Santo André

2011

Lima, Gabriel

Mapeamento de Geometrias a partir de Texturas em Tempo Real

/ Gabriel Lima - 2011

xx.p

1.Computação Gráfica 2. Detalhamento de superfícies 3. Mapeamento de Textura.. I.Título.

CDU 536.21

*Aos meus pais e irmão
Aos amigos, pelo apoio e companheirismo.*

Resumo

Na Computação Gráfica os objetos 3D são geralmente modelados por malhas poligonais. Estas requerem um grande número de polígonos para a representação de superfícies realistas. Sobre tudo, o elevado número de polígonos demanda um alto processamento das GPUs (*Graphics Processing Unit*), o que acaba limitando a qualidade das imagens geradas.

As técnicas de Detalhamento de Superfícies simulam objetos 3D realistas com baixo número de polígonos, permitindo que as superfícies com altos níveis de detalhes possam ser geradas eficientemente. Compreendendo a importância e o elevado número de métodos de detalhamento criados nos últimos anos, este trabalho realiza um estudo, implementação e comparação das principais técnicas de Detalhamento de Superfícies, com o objetivo de averiguar as vantagens e desvantagens, bem como os ambientes onde cada uma destas são melhor aplicadas.

Palavras-chaves: Displacement Mapping, GPU, Mapeamento de Texturas.

Keywords: Displacement Mapping, GPU, Texture Mapping.

Agradecimentos

A todos os meus parentes, pelo encorajamento e apoio.

*“Sem suor e sem trabalho nenhuma obra
é terminada”.*

Schrevelius

Sumário

Notação	8
1 Introdução	9
1.1 Detalhamento de Superfícies	9
1.2 Objetivos	13
1.3 Divisão do Trabalho	14
2 Fundamentos Teóricos	15
2.1 Texturas	15
2.1.1 Mapeamento de Texturas	16
2.1.2 Mapa de Níveis	18
2.1.3 Mapa de Normais	19
2.2 Modelo de Iluminação Phong	22
2.3 Modelos de Tonalização	27
2.4 Espaço Tangente	29
3 Detalhamento de Superfícies	33
3.1 Métodos Clássicos	33
3.1.1 Displacement Mapping	33
3.1.2 Bump Mapping	34
3.1.3 Normal Mapping	36
3.1.4 Refinamento do Normal Mapping: Generalização de Problema . . .	39
3.2 Métodos Não Iterativos	41
3.2.1 Parallax Mapping	42

3.2.2	Parallax Mapping with Offset Limiting	47
3.2.3	Parallax Mapping with Slope Information	49
3.3	Métodos Iterativos	52
3.3.1	Iterative Parallax Mapping	53
3.3.2	Busca Linear	55
3.3.3	Busca Binária	58
3.3.4	Relief Mapping	61
3.3.5	Parallax Occlusion Mapping	64
3.3.6	Cone Step Mapping	68
4	Comparação das Técnicas de Detalhamento de Superfícies	72
4.1	Comparação Teórica	72
4.2	Comparação Prática	73
5	Considerações Finais	91
A	Detalhamento de Superfícies através do Mapeamento de Texturas	93
A.1	Mapeamento de Texturas	93
A.2	Normal Mapping	93
A.3	Parallax Mapping	95
A.4	Parallax Mapping with Offset Limiting	97
A.5	Parallax Mapping with Slope Information	99
A.6	Iterative Parallax Mapping	101
A.7	Busca Linear	103
A.8	Busca Binária	105
A.9	Relief Mapping	107
A.10	Parallax Occlusion Mapping	110
A.11	Cone Step Mapping	114

Notação

Símbolos Matemáticos

- [] Matriz quadrada ou retangular
- []^T Matriz transposta
- []⁻¹ Matriz inversa

Operadores

- Produto escalar
- \times Produto vetorial
- $+$ Adição de escalares e vetores
- $-$ Subtração de escalares e vetores

Funções Especiais

- $T(u, v)$ A cor RGBA nas coordenadas de textura (u, v)
- $H(u, v)$ O Nível nas coordenadas de textura (u, v) do Mapa de Níveis
- $\vec{N}(u, v)$ O vetor normal nas coordenadas de textura (u, v) do Mapa de Normais
- $\max(x, y)$ O maior valor entre x e y

1 Introdução

Este capítulo realiza uma introdução ao Detalhamento de Superfícies abordando seus conceitos básicos que servem como motivador para o desenvolvimento deste trabalho. Na seção 1.1 são abordados os principais fundamentos do Detalhamento de Superfícies, na seção 1.2 são expostos os objetivos a serem alcançados no decorrer desta abordagem e na seção 1.3 estabelece-se a organização adotada neste texto.

1.1 Detalhamento de Superfícies

As aplicações de Computação Gráfica utilizam extensivamente malhas poligonais para a representação de superfícies ou objetos 3D. Para se atingir altos níveis de detalhamento em malhas, é necessário a utilização de um grande número de polígonos, uma propriedade proibitiva para as arquiteturas de placas de vídeo. Sendo assim, as aplicações gráficas de tempo real como jogos e simuladores acabam reduzindo o número de polígonos necessários para representações realistas, mantendo a geração de imagens sob taxas interativas no processo de *Render*. Com isso passa a haver uma relação direta entre a velocidade de processamento das placas de vídeo e a qualidade gráfica exibida pelas aplicações. Conforme a velocidade de processamento aumenta, o número de polígonos podem ser ampliados permitindo um melhor detalhamento de objetos, mantendo taxas de geração de imagens adequadas. Como a maioria das aplicações deseja a representação de cenas com grande número de objetos exibindo altos níveis de detalhamento, o número de polígonos necessários se torna tão alto que as placas de vídeo não conseguem ter processamento adequado para a geração de imagens em altas taxas. Sendo assim, deveria-se encontrar outras maneiras de se realizar este detalhamento que fossem mais eficientes que a adição sucessiva de polígonos em malhas a medida que placas de vídeo mais potentes sejam produzidas.

A arquitetura de *GPU's* (*Graphic Processor Unit*) programáveis permitiu que determinadas partes do *pipeline* gráfico pudessem ser substituídos por *Shaders* [10, 4]. Os *Shaders* são programas que possibilitam a criação de efeitos personalizados, permitindo a artistas ou programadores especificarem o modo como um vértice de um polígono ou

fragmentos [5] devem ser sujeitos ao processo de *Render*. Isto contribuiu para uma maior liberdade na criação de um visual e estilos únicos em aplicativos gráficos. Esta arquitetura baseada em *Shaders* permitiu que novas estratégias de detalhamento de superfícies fossem criadas, sem a necessidade da utilização de malhas com um número elevado de polígonos.

A área de pesquisa em Mapeamento de Detalhes em Tempo Real utiliza extensivamente *Shaders* e texturas para modificar partes do *pipeline* gráfico, permitindo que detalhes sejam mapeados em superfícies com baixo número de polígonos. Esta área define três níveis de estrutura de detalhes: a macro-estrutura, meso-estrutura e a micro-estrutura. O nível de macro-estrutura representa os detalhes mais abruptos, sendo normalmente uma referência à malhas poligonais. O nível de meso-estrutura é definido como os detalhes geométricos que são relativamente pequenos e distinguíveis dependendo do ponto em que são observados, normalmente representados por pequenas deformações na superfície. O nível de micro-estrutura representa os detalhes da superfície que são indistinguíveis quando observados por diferentes pontos de observação, sendo assim, este nível é normalmente modelado por texturas convencionais.

As malhas ou macro-estruturas utilizadas na representação de objetos 3D com formas complexas, necessitam de um elevado número de polígonos para que a representação das pequenas deformações da superfície sejam perceptíveis. Os detalhes da meso-estrutura podem ser obtidos sem a necessidade de malhas finas através de um conjunto de técnicas baseadas no mapeamento de texturas. Estas técnicas mantêm armazenadas em texturas as propriedades da meso-estrutura, tais como, os níveis de altura e vetores normais do relevo, de forma que suas pequenas deformações possam ser criadas em tempo real no processo de tonalização de fragmentos. Isto permite a redução do número de polígonos na malha, o que diminui a alocação de memória e aumenta a disponibilidade de unidades de processamento nas placas de vídeo. Com isso pode-se obter altos padrões de detalhamentos na meso-estrutura, mantendo alta a geração de imagens no processo de *Render*, uma propriedade importante para aplicações gráficas de tempo real.

O processo de detalhamento de superfícies utilizando o Mapeamento de Texturas, pode ser compreendido com o esquema da Figura 1.1, a macro-estrutura é uma malha simplificada com pequeno número de polígonos e a meso-estrutura é definida por uma textura que armazena as informações de altura e vetor normal de cada um dos pontos no relevo. No processo de tonalização do *Render* as informações da meso-estrutura e do ponto de observação são levados em conta no cálculo da iluminação para a re-criação dos

detalhes produzidos por uma malha fina.

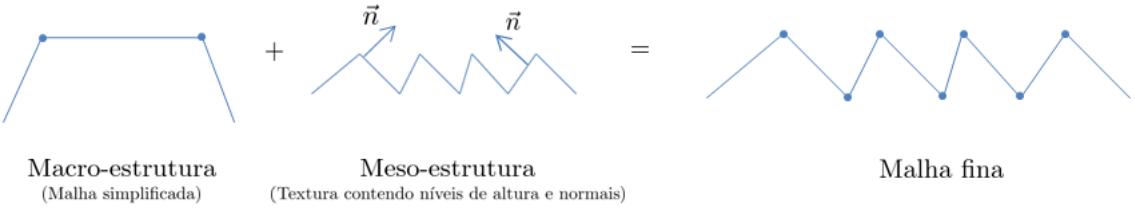


Figura 1.1: A utilização da macro-estrutura e meso-estrutura para a representação de malhas finas.

A Figura 1.2 apresenta duas malhas (*a*) e (*c*) para a representação de uma superfície irregular de pedras. A malha (*a*) possui grande número de polígonos, o que permite um detalhamento fino das características da superfície obtendo (*b*) a partir de um mapeamento de textura convencional. A malha (*c*) possui dois triângulos e um mapeamento de detalhes utilizando informações da meso-estrutura extraídas de (*b*) permite uma simulação de detalhes aproximada em (*d*). Como o mapeamento de detalhes são técnicas implementadas no domínio da imagem, uma maior percepção de realismo é obtido quando o observador esta em movimento em relação à superfície, sendo assim, a qualidade mostrada em (*d*) tende a ser mais bem evidenciadas em aplicações interativas.

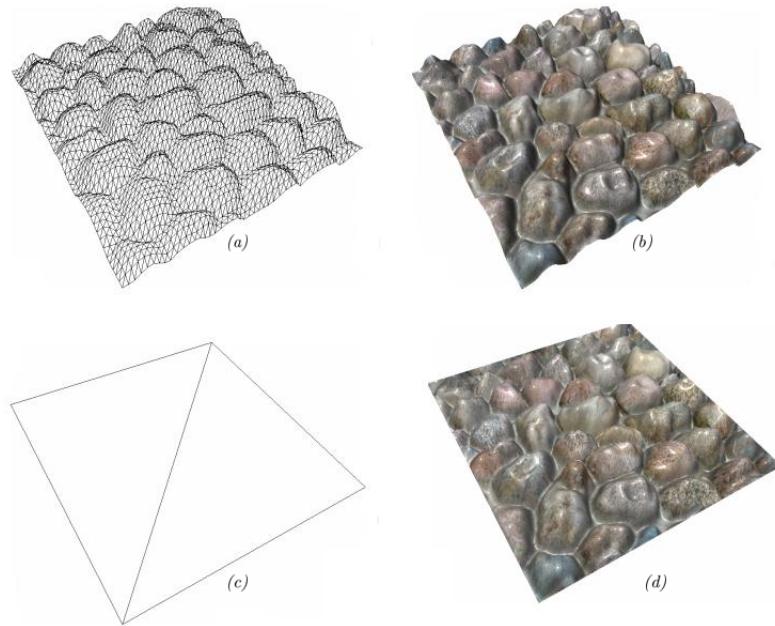


Figura 1.2: Duas malhas de complexidades antagônicas são mostradas. A malha (*a*) possui elevado número de polígonos que recebem um Mapeamento de Textura em (*b*). A malha (*c*) possui dois triângulos e uma técnica de Detalhamento de Superfícies em (*d*).

O mapeamento de detalhes em superfícies pode ser também explorado eficientemente utilizando a unidade programável de *Geometry* [28] e de *Tesselation* [28] presentes no *Shader Model 4*. As mesmas informações da meso-estrutura e posicionamento do observador são utilizados sobre macro-estruturas com baixo número de polígonos, de forma que as regiões que estão mais próximas e no campo de observação tenham acrestadas dinamicamente um maior número de polígonos. Sendo assim, o número de polígonos é reduzido e o detalhamento da meso-estrutura é conseguido de maneira dinâmica sob o domínio da malha.

Na Figura 1.3 um observador com visão superior visualiza uma região com uma superfície irregular a grandes distâncias em (a), como os detalhes da meso-estrutura não são perceptíveis a esta distância utiliza-se um baixo número de polígonos. Quando uma maior aproximação ocorre em (b), as regiões da superfície no campo de visão adquirem maior número de triângulos, utilizando as informações da meso-estrutura. Por fim, quando o observador se aproxima ainda mais da superfície em (c), a região observada apresenta uma adição de polígonos maior, permitindo que os detalhes ali contidos possam agregar maiores realismos. A idéia de realizar o mapeamento de detalhes ao nível das malhas é criar polígonos dinamicamente somente nas regiões próximas ao observador, ao passo que se decrementa o número de polígonos naquelas regiões que estão distantes. Isto permite a manutenção de um número de polígonos regular fluindo pelo *pipeline* gráfico, viabilizando a representação de ambientes altamente detalhados nas placas de vídeo atuais.

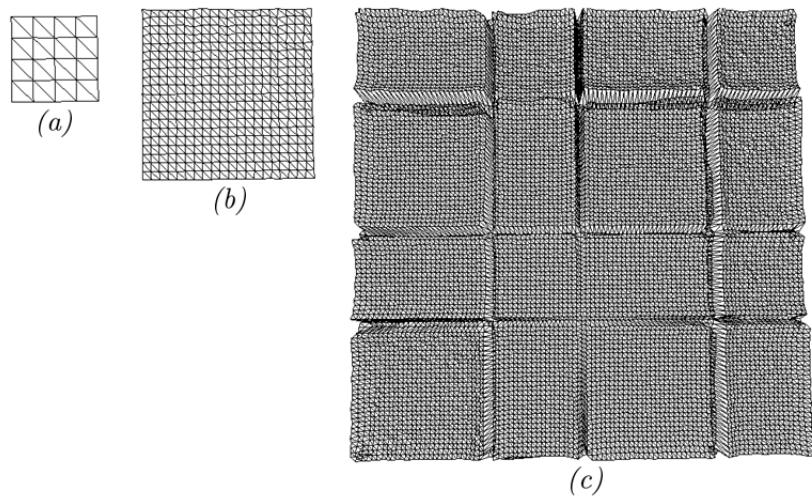


Figura 1.3: O observador visualiza uma superfície irregular a grandes distâncias em (a), aproximando-se cada vez mais em (b) e posteriormente em (c).

Seja qual for o método de detalhamento de superfícies utilizado, no domínio da imagem ou da malha, deseja-se simular os efeitos de silhueta, *motion parallax*, auto-oclusão e auto-sombreamento. A simulação de silhuetas permite que os contornos presentes em malhas finas sejam corretamente simulados. O efeito de *motion parallax* faz com que as partes da superfície mais próximas do observador se movimentem com maior velocidade que aquelas mais distantes. A auto-oclusão permite que partes da superfície mais próximas do observador possam ocultar as mais distantes. O efeito de auto-sombreamento possibilita que sombras sejam geradas por obstruções de luz do próprio relevo da superfície, ou seja, partes da superfície geram sombras sobre elas mesmas. Tais efeitos permitem que maiores níveis de realismo sejam adquiridos pelas técnicas de detalhamento de superfícies e são citados durante a abordagem das técnicas tratadas neste trabalho.

1.2 Objetivos

Nos últimos anos um elevado número de técnicas de Detalhamento de Superfícies foram desenvolvidas. Cada uma delas permite a geração de imagens exibindo diferentes qualidades visuais e desempenho. Com isso, este trabalho apresenta o funcionamento das principais técnicas de Detalhamento de Superfícies, os efeitos visuais tratados pelas mesmas e as principais vantagens e desvantagens de sua utilização. Ainda devem ser realizados os comparativos entre as técnicas apresentando as diferenças entre as imagens geradas e seu desempenho em taxa de geração de imagens.

As contribuições deste trabalho são:

- Abordar o tema de forma didática realizando um embasamento teórico mínimo, não disponível em artigos.
- A abordagem das técnicas através de uma notação homogênea que facilita sua compreensão.
- Relatar os problemas normalmente não mencionados nos artigos das técnicas.
- A comparação das técnicas por meio de qualidade visual e taxa de geração de imagens.
- A disponibilização de todos os códigos fontes das técnicas analisadas.

1.3 Divisão do Trabalho

Este trabalho está dividido em cinco capítulos, inicialmente foi realizada uma introdução no capítulo 1, expondo os principais motivadores para a utilização do Detalhamento de Superfícies. O capítulo 2 aborda os fundamentos teóricos necessários para a compreensão das técnicas tratadas neste trabalho. O capítulo 3 trata dos métodos de Detalhamento de Superfícies Clássicos, Não Iterativos e Iterativos. O capítulo 4 realiza um comparativo entre qualidade visual e o desempenho a partir dos métodos de detalhamento implementados. No capítulo 5 são apresentadas as considerações finais e por fim, no Apêndice A são mostrados os códigos fontes de cada uma das técnicas.

2 Fundamentos Teóricos

A compreensão das técnicas de Detalhamento de Superfícies exige um conhecimento adequado sobre texturas, modelos de iluminação, tonalização e Espaço Tangente. Tais temas são a base fundamental de toda e qualquer técnica tratada neste trabalho. Consequentemente este capítulo abordará inicialmente os conceitos de texturas na seção 2.1, utilizados para a representação da micro-estrutura e meso-estrutura. Em seguida, será abordado o modelo de iluminação de *Phong* na seção 2.2. Os diferentes modelos de tonalização na seção 2.3. E por fim, é abordado um sistema de coordenadas local da textura na seção 2.4, conhecido como Espaço Tangente, essencial para a conversão eficiente de atributos da meso-estrutura codificados em texturas.

2.1 Texturas

Uma imagem é uma matriz bidimensional, onde cada elemento é denominado por *picture element (pixel)*. Cada *pixel* é composto por um vetor, cuja as componentes são denominadas *canais*. Dependendo do formato de imagem, um *pixel* pode possuir de um a quatro *canais*, sendo cada um destes referidos por R, G, B e A. O primeiro canal R determina a intensidade de cor vermelha (*Red*) que o *pixel* possui, o segundo canal G (*Green*) a intensidade de verde, o terceiro canal B (*Blue*) a intensidade de azul e por fim, A o nível de transparência (*Alpha*), geralmente utilizada para a combinação de multiplas imagens em efeitos de transparência. Cada canal possui um valor inteiro entre 0 e 255, representando a intensidade de cor associada. Com isso um *pixel* contendo o vetor (a, b, c, d) terá uma intensidade a para o canal R, a intensidade b para o canal G, a intensidade c para o canal B e a intensidade d para o canal A. A cor final visualizada neste *pixel* será a composição das intensidades de cor de cada um dos canais.

Uma textura é uma imagem utilizada para ser mapeada para uma superfície. Elas são utilizadas para agregar realismo a objetos 3D, como pode ser visto na Figura 2.1. Como as texturas são imagens, todas as características definidas para as imagens permanecem válidas, contudo, a referência aos *pixels* de uma textura poderão ser referenciados

por *texels*.

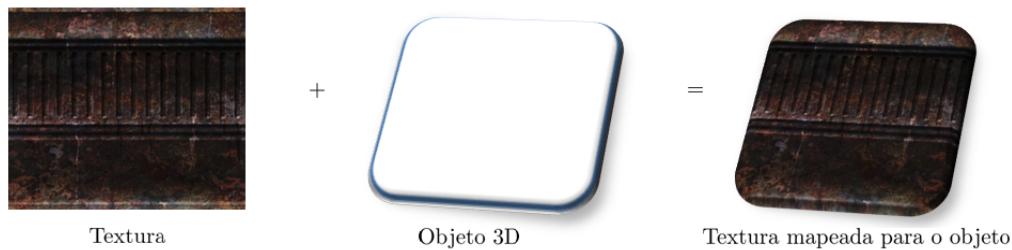


Figura 2.1: Uma textura sendo mapeada para um objeto.

As *GPUs* modernas permitem que operações matemáticas possam ser realizadas sobre os canais dos *texels* de maneira eficiente. Isto acaba motivando a utilização de texturas como entrada de dados para cálculos. Como um *texel* pode armazenar números em cada um dos canais que o compõem, pode-se abstrair sua semântica de cor para armazenar outros atributos, aproveitando assim da eficiência no acesso e processamento paralelo das *GPUs* às texturas. Isto é, os *canais* RGBA de uma textura podem ser utilizadas para armazenar dados que nada tem haver com a representação de cores, podendo servir como variáveis de entrada para problemas computacionais tratados nas *GPU's* com *Shaders*. Sendo assim, iremos abordar nas próximas seções os Mapas de Níveis e Mapas de Normais, que armazenam respectivamente, alturas de pontos e vetores normais de uma superfície nos *canais* RGBA de uma textura. Tais informações são utilizadas frequentemente para a descrição do nível da meso-estrutura e são usados em todas as técnicas de Detalhamento de Superfícies.

Como visto existem texturas que armazenam dados que não definem cor, isto pode gerar em alguns casos confusão. Para isso definimos como textura difusa àquela que mantém a mesma abstração de imagem, ou seja, é construída para representar a cor de um objeto 3D e não como entrada de dados para o cálculo com *Shaders*.

2.1.1 Mapeamento de Texturas

O Mapeamento de Texturas é o processo de envolver uma textura ao redor de um objeto fazendo com ele adquira um visual mais realista no nível da micro-estrutura. Neste caso é conveniente utilizar uma função $T(u, v)$ que mapeia um ponto (u, v) em coordenadas de textura para uma cor RGBA. As coordenadas de textura posicionam sua origem na base inferior esquerda da textura e têm seus *texels* obtidos variando-se os parâmetros u e

v entre 0 e 1, onde 0 é a origem e 1 é a extremidade oposta para cada um dos eixos u na horizontal e v na vertical. A Figura 2.2 mostra a relação entre as coordenadas de textura (u, v) e os *texels*, em $T(0, 0)$ obtém-se o *texel* do canto inferior esquerda, com $T(1, 0)$ o *texel* do canto inferior direito, com $T(1, 1)$ o *texel* no canto superior direito, com $T(0, 1)$ o *texel* do canto superior esquerdo e em $T(0.5, 0.5)$ o *texel* no centro da textura.

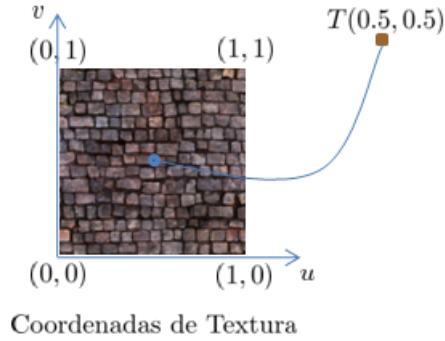


Figura 2.2: Sistema de coordenadas de textura e a função $T(u, v)$.

Como a textura é uma matriz bidimensional, as coordenadas de textura não são utilizadas para a obtenção de *texels* diretamente, sendo assim, o *texel* (i, j) é obtido fazendo $(i, j) = (uw, vh)$, onde w é a largura e h é altura da textura em número de *texels*. A multiplicação dos parâmetros u e v pela largura e altura geralmente irão resultar em valores fracionários que são descartados para o acesso ao *texel* (i, j) .

Na prática quando se define uma malha, cada um dos vértices que compõem os polígonos recebe coordenadas de textura, que no processo de *Render* são interpolados linearmente para que as regiões do interior do polígono adquiram coordenadas de textura, e por fim obtenham uma cor do *texel* correspondente.

As aplicações gráficas utilizam extensivamente o Mapeamento de Texturas, como pode ser visto na Figura 2.3. Nesta cena todo o ambiente utiliza o mapeamento de uma única textura, enquanto que o personagem utiliza multiplas texturas para compor os detalhes da micro-estrutura com maior nível de realismo.

O Mapeamento de Texturas pode ser realizado em quaisquer linguagens de construção de *Shader*, estas utilizam objetos *Samplers* para que *texels* de uma textura sejam obtidas nas coordenadas informadas. A implementação de um *Shader* em linguagem *OpenGL Shading Language* (GLSL) [22], que realiza o Mapeamento de Texturas com interpolação linear pode ser consultado no Apêndice A.1.

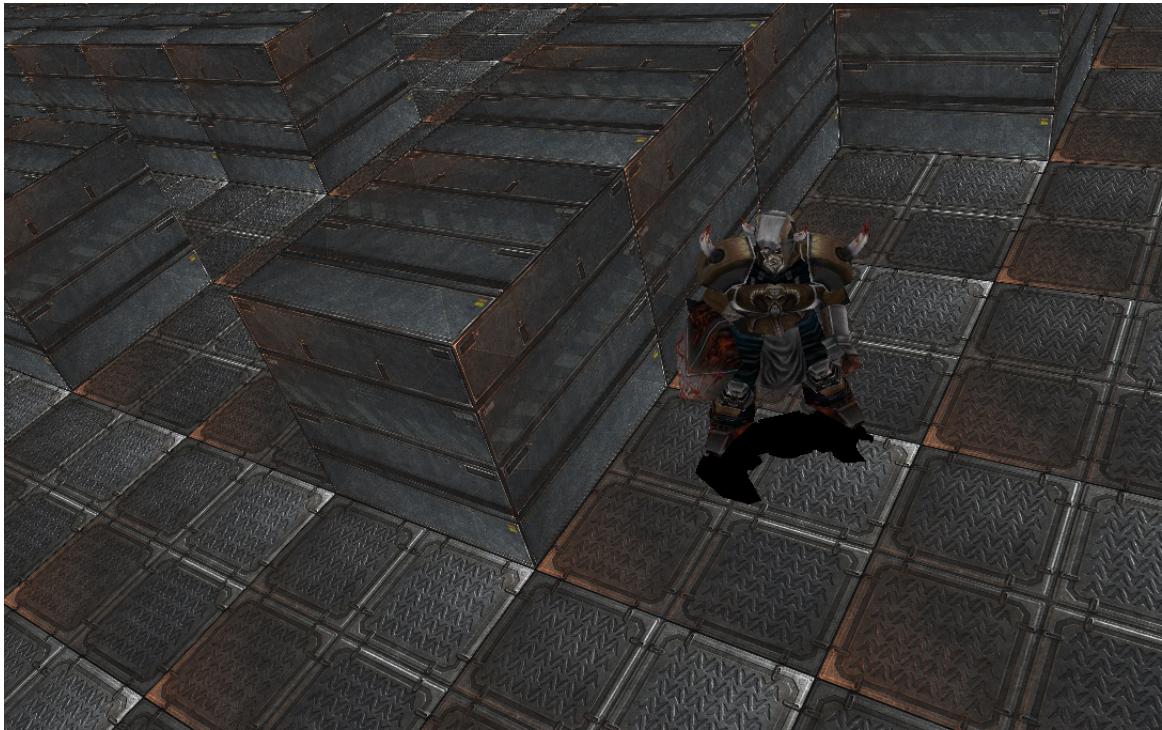


Figura 2.3: Um simulador de algoritmos de busca que utiliza o Mapeamento de Texturas para a geração de cenas mais realistas. O personagem e o ambiente contam com o mapeamento de multiplas texturas sobre suas superfícies.

2.1.2 Mapa de Níveis

O Mapa de Níveis [9, 24] é uma textura, geralmente composta por *pixels* com um único canal. Sua função é armazenar os níveis de altura que descrevem o relevo da meso-estrutura de uma superfície. A Figura 2.4 mostra uma textura difusa e um Mapa de Níveis. Para cada coordenada (u, v) da textura difusa tem-se um nível de altura $H(u, v)$ correspondente no Mapa de Níveis. Tal informação será utilizada em todas as técnicas de Detalhamento de Superfícies para a re-criação dos detalhes da meso-estrutura em tempo real, tais como, rugosidades e pequenas distorções no relevo.

Os níveis de altura do Mapa de Níveis são costumeiramente adotados com altura máxima de valor 1 e mínima como 0. Isto permite interpretá-lo graficamente, tendo as regiões de maior altura com cores mais próximas do branco, enquanto as mais baixas do preto.

Os Mapas de Níveis podem ser gerados a partir de texturas difusas. Para isso, utiliza-se a variação de cores nas redondezas de cada um dos *texels* de uma textura difusa como estimativa da altura. As ferramentas *Crazybump* e *ShaderMap* realizam este tipo

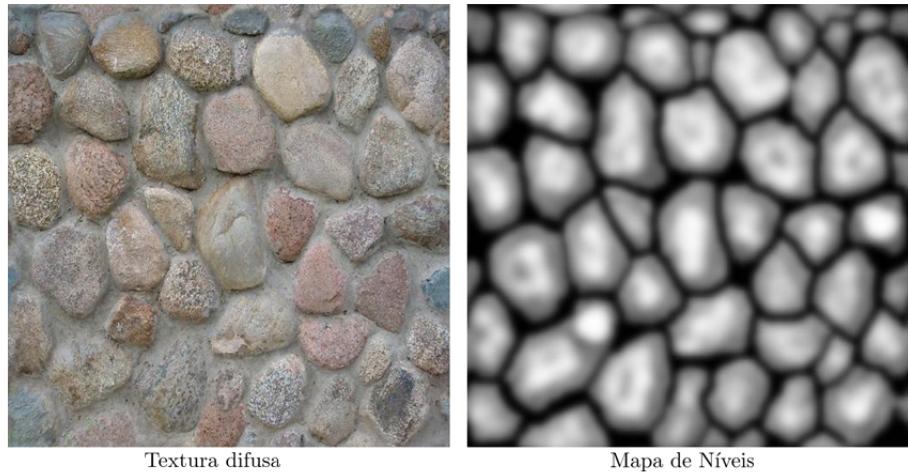


Figura 2.4: A representação das alturas da meso-estrutura em um Mapa de Níveis.

de procedimento facilitando o processo de produção artística.

2.1.3 Mapa de Normais

O Mapa de Normais [9, 24] é uma textura composta geralmente por *texels* com *canais* RGB, onde cada um destes armazena as coordenadas do vetor normal unitário associados a uma textura difusa. Ou seja, dado um vetor normal $\vec{n} = (x, y, z)$, a coordenada x é armazenada no *canal R*, a coordenada y no *canal G* e a coordenada z no *canal B* de um *texel*.

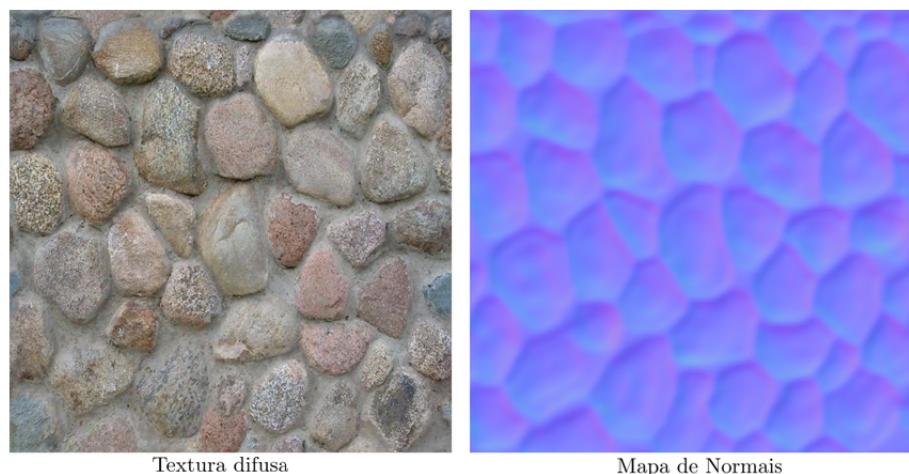


Figura 2.5: A representação dos vetores normais de uma meso-estrutura em um Mapa de Normais.

A Figura 2.5 mostra uma textura difusa e um Mapa de Normais, para cada coordenada (u, v) da textura difusa tem-se um vetor normal $\vec{N}(u, v)$ correspondente no

Mapa de Normais. Assim como no Mapa de Níveis o Mapa de Normais será utilizado extensivamente como fonte de informação do relevo da superfície, possibilitando a criação dos detalhes da meso-estrutura em tempo real.

Os vetores normais do Mapa de Normais informam a inclinação da superfície da qual estão sendo mapeados, sendo assim, seus vetores estão normalmente direcionados para fora dos objetos. Como será tratado na seção de Espaço Tangente estes vetores normais são codificados sobre um sistema de coordenadas local, onde o eixo Z é perpendicular a superfície da textura. Com isso os *canais* B obtêm frequentemente os maiores valores dentre todos os canais, fazendo com que os Mapas de Normais possuam uma coloração azulada característica.

Como as imagens não suportam *canais* com valores negativos e as texturas tratadas em *APIs* gráficas como *OpenGL* [23] e *DirectX* [6] possuem *canais* sendo representados no intervalo entre 0 e 1. O armazenamento de vetores normais não pode ser realizado diretamente, visto que vetores unitários possuem coordenadas no intervalo de -1 a 1. Então quando se utiliza uma destas *APIs* para armazenar as normais em texturas, deve-se realizar um processo de mapeamento prévio dado pela Eq. (2.1) [9], onde c_p é um *texel* que obtém o vetor normal cuja as coordenadas estão no intervalo entre 0 e 1.

$$c_p = (R, G, B) = 0.5\vec{n} + (0.5, 0.5, 0.5) \quad (2.1)$$

O inverso também é preciso, quando se trabalha com *Shaders* os *canais* RGB do Mapa de Normais estão no intervalo entre 0 e 1. Torna-se assim necessário um remapeamento das coordenadas dos vetores normais c_p codificados como *texels* para o intervalo dado entre -1 e 1, de maneira que os vetores normais extraídos do Mapa de Normais possam ser utilizados corretamente. Isto é realizado através da Eq. (2.2) [9].

$$\vec{n} = (x, y, z) = 2(\vec{c}_p - (0.5, 0.5, 0.5)) \quad (2.2)$$

Os mapeamentos previamente tratados têm implicações no desempenho e devem ser utilizados com cautela, principalmente aquele dado pela Eq. (2.2) que mapeia os *texels* de um Mapa de Normais para os vetores normais já que este processo costuma ocorrer para todos *pixels* do Mapa de Normais a cada nova imagem gerada pelo processo de *Render* nas técnicas de Mapeamento de Detalhes.

Um Mapa de Normais pode ser construído através do cálculo dos vetores normais de uma malha ou a partir de uma Mapa de Níveis. Em uma malha são obtidos de forma direta, através do cálculo dos vetores normais de cada vértice seguidos por interpolações lineares e mapeamento pela Eq. (2.1). Tal processo é exemplificado na Figura 2.6 onde os vetores normais \vec{n}_i da malha são obtidos a partir de interpolações lineares dos vetores normais $\vec{n}_1, \vec{n}_2, \vec{n}_3$ e \vec{n}_4 pertencentes aos vértices e armazenados no Mapa de Normais nas mesmas coordenadas de textura. A extração do Mapa de Normais a partir do Mapa de Níveis é realizado calculando-se para cada um dos *pixels* (u, v) do Mapa de Níveis os vetores $\vec{q} = (1, 0, H(u + 1, v) - H(u, v))$ e $\vec{t} = (0, 1, H(u, v + 1) - H(u, v))$. O vetor normal $\vec{N}(u, v)$ será obtido através de um produto vetorial como mostra a Eq. (2.3).

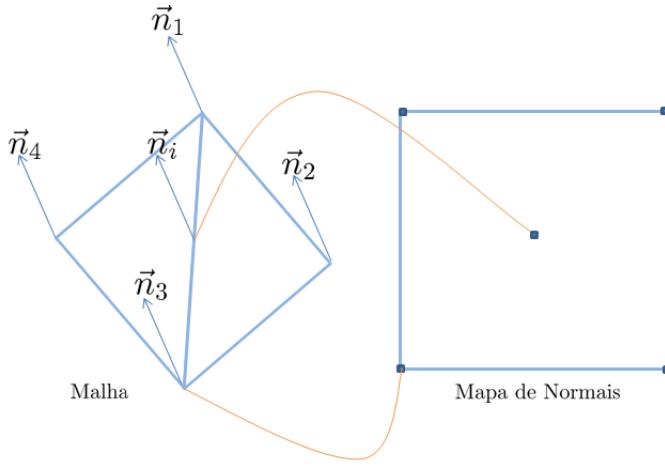


Figura 2.6: Mapeamento dos vetores normais \vec{n} de uma malha para o Mapa de Normais.

$$\begin{aligned}
 \vec{n} &= \vec{q} \times \vec{t} \\
 &= (1, 0, H(u + 1, v) - H(u, v)) \times (0, 1, H(u, v + 1) - H(u, v)) \\
 &= \frac{(H(u, v) - H(u, v + 1), H(u, v) - H(u + 1, v), 1)}{\sqrt{(H(u, v) - H(u, v + 1))^2 + (H(u, v) - H(u + 1, v))^2 + 1}}
 \end{aligned} \tag{2.3}$$

Atualmente já existem dezenas de ferramentas que geram Mapas de Normais a partir de Mapas de Níveis, as mais utilizadas pelos artistas são o *Crazybump*, *Shadermap* e o *plugin DDS* da NVidia desenvolvido para o *Photoshop*. A extração de Mapa de Normais a partir de malhas geralmente está disponível na grande maioria de programas de modelagem como o *Blender* e o *3D Studio Max*.

Os Mapas de Normais são comumente utilizados com *canais* RGB. Contudo, existem situações onde se deseja inserir outros dados em conjunto com os vetores normais,

permitindo que um acesso a textura retorne o máximo de dados possíveis. Por exemplo, em uma aplicação poderia ser necessário o acesso através de um *Shader* ao vetor normal, nível de altura e transparência de um *texel*. Como os formatos de imagens padrões possuem normalmente até 4 canais uma solução seria utilizar a codificação de vetores normais e níveis de altura em uma única textura RGBA comumente chamada de *Relief Map* [14] e a transparência em uma outra textura com apenas um canal. Isto é uma solução, contudo não a mais eficiente já que serão necessários dois acessos de memória por *texel* para que todos os dados estejam disponíveis. Utilizando as propriedades matemáticas de vetores unitários pode-se compactar o Mapa de Normais em apenas dois canais, liberando um canal para a inserção de outros tipos de informações. Isto é viável pois $\vec{n} = (x, y, z)$ têm seu módulo dado por $|\vec{n}| = \sqrt{x^2 + y^2 + z^2} = 1$, o que nos permite dizer que $z = \sqrt{1 - x^2 - y^2}$. Com isto pode-se utilizar a Eq. (2.1) desconsiderando o *canal* B para a compactação das normais no Mapa de Normais de dois canais. O processo inverso de descompactação, isto é, a obtenção dos vetores normais a partir do Mapa de Normais de dois *canais* é dado pela Eq. (2.2) desconsiderando-se o valor obtido para a coordenada z , sendo obtida posteriormente por $z = \sqrt{1 - x^2 - y^2}$.

O procedimento de compactação é útil em alguns casos, contudo sua descompactação realizada em tempo real requisita o cálculo da coordenada z do vetor normal utilizando o cálculo de raízes quadradas, uma operação custosa na arquitetura das *GPUs* mais antigas. Desta forma deve-se avaliar se o custo da extração da componente z não é maior que um acesso a uma textura na arquitetura de *GPU* que está se trabalhando antes de sua utilização.

2.2 Modelo de Iluminação Phong

Os modelos de iluminação na Computação Gráfica simulam a interação da luz com a superfície de objetos. Segundo os modelos da Física a luz tem característica dual, ora comportando-se como uma partícula ora como uma onda, diz-se então que a luz apresenta uma dualidade onda-partícula. Quando a luz interage com a superfície de um objeto, ocorrem os fenômenos de reflexão, refração, emissão, absorção, difração, interferência e polarização da onda de luz incidente. O elevado número de fenômenos presentes nesta interação inviabiliza que uma modelagem fisicamente correta seja construída em aplicações de Computação Gráfica, devido ao seu elevado custo computacional. O mo-

do Phong criado por *Bui Tuong Phong* em 1975 simplificou o modelo físico supondo que a maior parte dos objetos não emitem luz própria, mas absorvem e refletem a luz incidente provida de outras fontes. Assim, um objeto com cor amarela reflete ondas na faixa de comprimento amarelo, absorvendo as demais. O modelo de *Phong* explora esta propriedade de interação da luz com a superfície mostrando que o modelo físico pode ser simplificado nas componentes de reflexão difusa, reflexão espelhada e pela interação com a luz ambiente, como mostra a Figura 2.7.

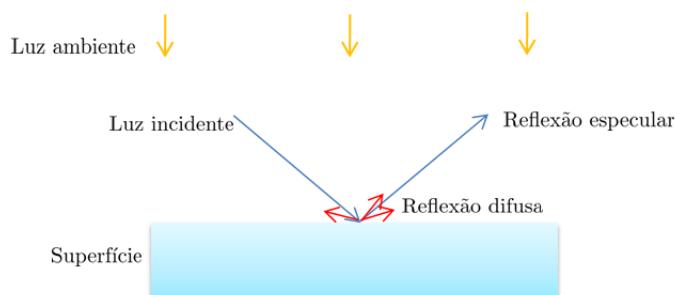


Figura 2.7: Componentes do modelo de *Phong* integrando com uma superfície.

A Figura 2.7 mostra que toda a luz incidente na superfície tem uma parcela, denominada reflexão difusa, que é espalhada uniformemente em todas as direções possíveis devido às micro irregularidades da superfície. Uma outra parcela, denominada reflexão espelhada é refletida pelo mesmo ângulo de incidência e modela o brilho visto em determinados pontos de um objeto. A luz ambiente simula uma iluminação indireta provida pela reflexão de luzes advindas de outros objetos em cena.

A reflexão difusa ou componente difusa do modelo de *Phong* considera que a luz incidente é espalhada com igual intensidade em todas as direções, independentemente da direção da visão. As superfícies que possuem estas características são chamadas de refletores Ideais ou refletores Lambertianos, visto que a intensidade da luz radiante de qualquer ponto na superfície é governada pela lei dos *Cossenos de Lambert*. Esta lei mostra que a energia radiante de qualquer pequena superfície dA em qualquer direção ϕ_n relativa a normal da superfície é proporcional a $\cos\phi_n$.

A quantidade de luz incidente depende da orientação da superfície em relação à direção da fonte luminosa. Na Figura 2.8 esta relação fica clara, em (a) a superfície é perpendicular à direção das luzes e assim recebe maior intensidade de luz. Enquanto em (b) a mesma superfície, mas com uma disposição oblíqua em relação as luzes incidentes,

recebe um menor intensidade luminosa.

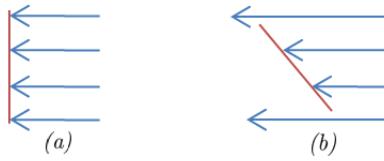


Figura 2.8: A superfície perpendicular a luz incidente (a) é mais iluminado que uma superfície de igual tamanho (b) sob um ângulo obliqua a luz incidente.

Assim, considerando um ponto da superfície com vetor normal \vec{n} e direção da luz dada por \vec{l} , ambos unitários, obtém-se a componente difusa I_{dif} do modelo *Phong* pela Eq. (2.4). A intensidade das cores da luz I_l é dada nas componentes RGB. A refletância difusa K_d dada também sobre as componentes RGB, é uma propriedade inerente ao material e determina quais níveis de intensidade nas componentes da luz I_l devem ser refletidas ou absorvidas pela superfície.

$$I_{dif} = I_l K_d \max(0, (\vec{n} \cdot \vec{l})) \quad (2.4)$$

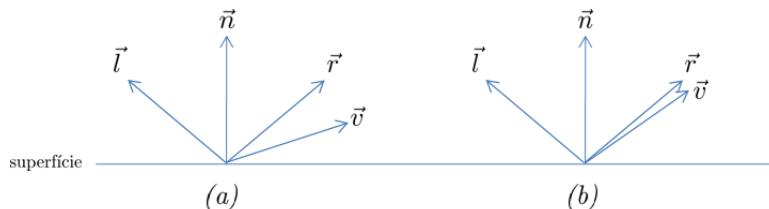


Figura 2.9: Uma mesma superfície interagindo com raios de luz (a) sob um ângulo grande e (b) sob ângulo pequeno entre a direção de visão \vec{v} e raio refletido \vec{r} .

A reflexão espelhada ou componente espelhada do modelo de *Phong* simula os pontos brancos vistos nas superfícies de objetos brilhantes. Trata-se da parcela de luz incidente à superfície que é refletida em relação ao vetor normal sob o mesmo ângulo. As intensidades de cor vistas nesta reflexão dependem do ângulo de observação no ponto. Sendo assim, observa-se componentes especulares com maior intensidade a medida que o ângulo diminui entre a direção da luz refletida \vec{r} e a direção do observador \vec{v} . Quando o ângulo entre \vec{r} e \vec{v} aumenta a intensidade das componentes especulares diminuem.

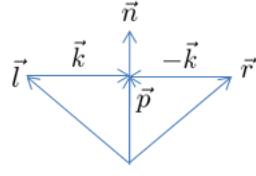


Figura 2.10: A construção vetorial utilizada para a dedução da direção de luz refletida \vec{r} .

A Figura 2.9 mostra a relação entre os vetores unitários \vec{v} , \vec{r} e \vec{n} , em (a) o ponto da superfície obterá menores intesidades de brilho que em (b) já que apresenta um menor ângulo entre os vetores \vec{r} e \vec{n} . O vetor de reflexão \vec{r} pode ser obtido fazendo-se a construção vetorial da Figura 2.10 que permite a obtenção de $\vec{p} = \vec{l} + \vec{k}$ e $\vec{p} = \vec{r} - \vec{k}$ que somados termo a termo resultam na expressão $\vec{p} = \vec{l} + \vec{r}$, como \vec{p} é paralelo ao vetor normal unitário \vec{n} , temos que $\vec{p} = (\vec{n} \cdot \vec{l})\vec{n}$ que substituídos na expressão resultante da soma nos permite obter a Eq. (2.5).

$$\vec{r} = 2\vec{n}(\vec{n} \cdot \vec{l}) - \vec{l} \quad (2.5)$$

A modelagem da componente especular I_{esp} do modelo de *Phong* é obtida pela Eq. (2.6).

$$I_{esp} = I_l K_e \max(0, (\vec{r} \cdot \vec{n}))^{n_s} \quad (2.6)$$

Na Eq. (2.6) da mesma maneira como apresentado para a componente difusa, tem-se a refletância especular K_e dada nas componentes RGB, trata-se de uma propriedade inerente ao material e determina quais níveis de intensidade nas componentes da luz I_l devem ser refletidas ou absorvidas pela superfície. O expoente n_s que têm seus valores dados de acordo com o material que se deseja simular, por exemplo, materiais brilhantes como metais devem possuir valores altos de n_s enquanto materiais opacos como a madeira valores baixos.

Na maioria dos casos as superfícies de objetos que não são atingidos diretamente por raios incidentes advindos de uma fonte de luz ainda são visíveis aos observadores. Isto ocorre porque a luz em um ambiente sofre reflexões sucessivas e acabam interagindo com o objeto não diretamente iluminado. Contudo, considerar o caminho destas reflexões em um modelo de iluminação seria muito custoso. Sendo assim, o modelo de

Phong propõe uma luz ambiente I_{amb} como um de seus componentes, dada pela equação Eq. (2.7).

$$I_{amb} = I_a K_a \quad (2.7)$$

A luz ambiente promove uma contribuição de cor constante a todas as superfícies em cena. A intensidade das cores da luz ambiente I_a e sua refletância é dada nas componentes RGB.

A intensidade final I nas componentes RGB dada pelo modelo de iluminação *Phong* descrito pelas componentes ambiente, difusa e especular é obtida pela Eq. (2.8).

$$\begin{aligned} I &= I_{amb} + I_{dif} + I_{esp} \\ &= I_a K_a + I_l (K_d \max(0, (\vec{n} \cdot \vec{l})) + K_e \max(0, (\vec{r} \cdot \vec{n}))^{n_s}) \end{aligned} \quad (2.8)$$

Pode-se ainda generalizar o modelo de *Phong* para n fontes de luzes pela Eq. (2.9).

$$I = I_a K_a + \sum_{i=1}^n I_{l_i} (K_d \max(0, (\vec{n} \cdot \vec{l}_i)) + K_e \max(0, (\vec{r}_i \cdot \vec{n}))^{n_s}) \quad (2.9)$$

A intensidade da energia luminosa que interage em uma superfície depende da distância em que a fonte de luz está deste ponto, quanto maiores as distâncias entre a fonte de luz e a superfície menores deverão ser as intensidades luminosas percebidas já que frequentemente a luz recebe atenuações atmosféricas. Para isso o modelo de iluminação *Phong* utiliza a Eq.(2.10) para atenuar as intensidades das luzes da Eq. (2.9), onde a , b e c são constantes empíricas e d é a distância do objeto à fonte de luz.

$$A_{tt} = \frac{1}{a + bd + cd^2} \quad (2.10)$$

Com isso pode-se agora definir a equação geral para o modelo de iluminação *Phong* nas componentes RGB de acordo com a Eq. (2.11).

$$I = I_a K_a + \sum_{i=1}^n \frac{1}{a + bd_i + cd_i^2} (I_{l_i}(K_d \max(0, (\vec{n} \cdot \vec{l}_i)) + K_e \max(0, (\vec{r}_i \cdot \vec{n}))^{n_s})) \quad (2.11)$$

O modelo de iluminação *Phong* permite que seja calculada a influência da intensidade I que uma fonte de luz causa sobre cada um dos pontos de uma superfície aumentando o realismo das cenas. A Figura 2.11 mostra o ganha de qualidade adquirido com a utilização do modelo de iluminação de *Phong* utilizado em conjunto com o Mapeamento de Texturas em (b), as regiões em passam a ter um sombreamento adequado em relação ao posicionamento da luz o que causa uma sensação de realismo não vista em (a).



Figura 2.11: Uma estátua com Mapeamento de Texturas em (a) e Mapeamento de Texturas com o modelo de iluminação de *Phong* em (b).

2.3 Modelos de Tonalização

Como visto na seção anterior o modelo de iluminação *Phong* determina a cor de um ponto levando em conta as propriedades ópticas da superfície do objeto, as características da luz incidente e a posição do observador. Os modelos de tonalização determinam onde e como este modelo de iluminação deve ser aplicado empregando para isso dois tipos de estratégia, a tonalização *Gouraud* e a *Phong*.

No modelo de tonalização *Gouraud*, o modelo de iluminação é aplicado nos

vértices e as cores destes vértices são interpoladas linearmente sobre a superfície do polígono. Enquanto que no modelo de tonalização *Phong* o modelo de iluminação é aplicado em cada ponto interior do polígono.

Nas malhas poligonais os vetores normais da superfície podem ser tratados segundo duas estratégias, a facetada ou com suavização geométrica. Na solução facetada, a normal do polígono é associada a cada vértice que o compõem. Na solução com suavização geométrica, um vértice tem sua normal obtida calculando-se a média das normais de todos os polígonos que o possuem, sendo assim, cada vértice adquire uma normal distinta.

A Figura 2.12 mostra um comparativo entre os modelos de tonalização *Gouraud* e tonalização *Phong* com a solução facetada de normais.

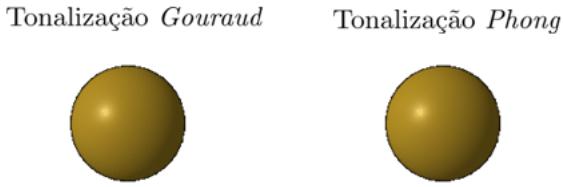


Figura 2.12: Um comparativo entre os modelos de tonalização *Gouraud* e *Phong* utilizando a solução facetada.

A observação do comportamento das esferas da Figura 2.12 mostra que o brilho do modelo de tonalização *Gouraud* se espalha pelo objeto. Enquanto que no modelo de tonalização *Phong* o brilho é mais definido. Isto ocorre porque na tonalização *Gouraud*, o modelo de iluminação é aplicado apenas nos vértices e as cores do interior do polígono são obtidas através da interpolação linear das cores atribuídas aos vértices. Enquanto no modelo de tonalização *Phong* a iluminação é calculada para cada ponto, fazendo com que haja a impressão de que o ponto de brilho se movimente conforme rotações são aplicadas sobre a esfera.

A Figura 2.13 mostra um comparativo entre os modelos de tonalização *Gouraud* e tonalização *Phong* com a solução de suavização geométrica.

A observação do comportamento das esferas da Figura 2.13 mostra que o brilho do modelo de tonalização *Gouraud* permanece se espalhando pela superfície. Contudo, o modelo de tonalização *Phong* torna-se mais bem definido, não havendo mais a impressão de que o brilho se movimenta conforme o objeto sofre rotação.

Na solução facetada todos os objetos apresentam uma característica descon-

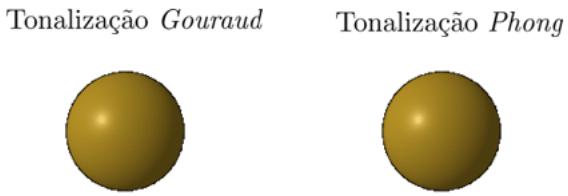


Figura 2.13: Um comparativo entre os modelos de tonalização *Gouraud* e *Phong* utilizando a solução de suavização geométrica.

tinua, uma vez que a normal da superfície utilizada na tonalização é igual em todos os pontos de um polígono. Dessa forma, na transição de um polígono para o outro, há uma mudança brusca na normal a ser usada no modelo de iluminação. Fazendo com que as superfícies que refletem a luz na direção perpendicular ao observador estejam mais claras. Na estratégia de suavização geométrica os objetos não apresentam transições bruscas nas transições entre os polígonos, pois as normais da superfície são calculadas através de interpolação das normais de seus vértices. Desta maneira, a transição de um polígono para o outro ocorre de suavemente. Como o modelo de tonalização *Phong* obtém imagens com melhor qualidade, ele será utilizado nas técnicas de Detalhamento de Superfícies.

2.4 Espaço Tangente

O Espaço Tangente [12] é um sistema de coordenadas local da textura utilizado como referência para os cálculos realizados pela maioria das técnicas de Detalhamento de Superfícies tratadas neste trabalho. Ele é definido através dos vetores tangente \vec{t} , bi-tangente \vec{b} e normal \vec{n} , como mostrado na Figura 2.14.

Com as coordenadas de textura dadas por (u, v) , \vec{t} é direcionado para onde u aumenta, \vec{b} é orientado para onde v aumenta e \vec{n} é perpendicular aos outros dois, formando assim a base do Espaço Tangente mostrada na Figura 2.14. Sua idéia fundamental é a criação de um sistema de coordenadas que permitam os vetores normais do Mapa de Normais descreverem a inclinação da superfície ao qual estão sendo mapeados, sem a necessidade de quaisquer transformações. Quando os vetores normais do Mapa de Normais estão nas Coordenadas de Mundo [5] e a superfície ao qual eles descrevem sofre uma transformação, todas as normais deverão sofrer transformações semelhantes para serem utilizadas nos cálculos de iluminação. Tal operação é computacionalmente custosa, já

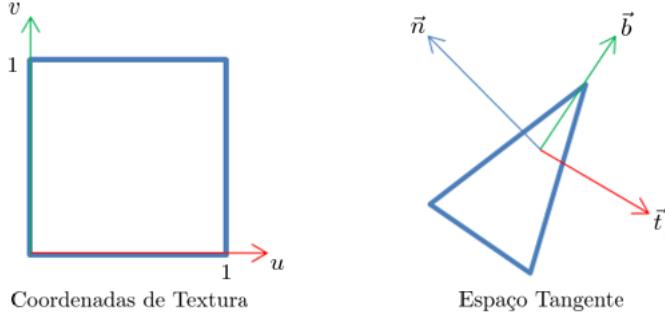


Figura 2.14: A esquerda o sistema de coordenadas da textura e à direita o Espaço Tangente composto pelos vetores Tangente \vec{t} , Bi-tangente \vec{b} e Normal \vec{n} .

que Mapas de Normais frequentemente possuem resolução de 256x256 ou 512x512 *pixels*. Contudo, quando Mapas de Normais utilizam o Espaço Tangente tais transformações não são necessárias. Em contraposição é preciso converter os vetores direção da luz e visão em Coordenadas de Mundo para as coordenadas do Espaço Tangente, uma tarefa bem menos custosa, permitindo a realização dos cálculos sob um único sistema. Para isto torna-se necessário uma matriz $M_{w,t}$, que permita a mudança de base de um vetor no sistema de Coordenadas de Mundo para o Espaço Tangente.

Supondo que a base ortogonal das Coordenadas de Mundo é dada pelos vetores canônicos $(\vec{u}, \vec{v}, \vec{w})$ e a base ortogonal do Espaço tangente é dada pelos vetores $(\vec{t}, \vec{b}, \vec{n})$, pode-se escrever os vetores base do Espaço Tangente em função dos vetores da base das Coordenadas de Mundo pelas Eqs. (2.12), (2.13) e (2.14).

$$\vec{t} = t_x \vec{u} + t_y \vec{v} + t_z \vec{w} \quad (2.12)$$

$$\vec{b} = b_x \vec{u} + b_y \vec{v} + b_z \vec{w} \quad (2.13)$$

$$\vec{n} = n_x \vec{u} + n_y \vec{v} + n_z \vec{w} \quad (2.14)$$

Escrevendo \vec{r} na base do Espaço Tangente em (2.15) e na base das Coordenadas de mundo em (2.16), deseja-se encontrar uma matriz mudança de base $M_{w,t}$ que transforme vetores nas Coordenadas de Mundo para as coordenadas do Espaço Tangente.

$$\vec{r} = f_1 \vec{t} + f_2 \vec{b} + f_3 \vec{n} \quad (2.15)$$

$$\vec{r} = e_1 \vec{u} + e_2 \vec{v} + e_3 \vec{w} \quad (2.16)$$

Colocando (2.12), (2.13) e (2.14) em (2.15), obtém-se (2.17).

$$\begin{aligned}\vec{r} &= f_1(t_x \vec{u} + t_y \vec{v} + t_z \vec{w}) + \\ &\quad f_2(b_x \vec{u} + b_y \vec{v} + b_z \vec{w}) + \\ &\quad f_3(n_x \vec{u} + n_y \vec{v} + n_z \vec{w})\end{aligned}\tag{2.17}$$

A Eq. (2.17) é re-arranjada obtendo-se a Eq. (2.18).

$$\begin{aligned}\vec{r} &= (f_1 t_x + f_2 b_x + f_3 n_x) \vec{u} + \\ &\quad (f_1 t_y + f_2 b_y + f_3 n_y) \vec{v} + \\ &\quad (f_1 t_z + f_2 b_z + f_3 n_z) \vec{w}\end{aligned}\tag{2.18}$$

Os coeficientes de (2.16) devem ser iguais aos de (2.18), sendo assim pode-se encontrar as Eqs. (2.19), (2.20) e (2.21).

$$e_1 = f_1 t_x + f_2 b_x + f_3 n_x\tag{2.19}$$

$$e_2 = f_1 t_y + f_2 b_y + f_3 n_y\tag{2.20}$$

$$e_3 = f_1 t_z + f_2 b_z + f_3 n_z\tag{2.21}$$

Re-escrevendo (2.19), (2.20) e (2.21) na forma matricial:

$$\begin{aligned}\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} &= \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \\ M_{t,w} &= \begin{bmatrix} t_x & b_x & n_x \\ t_y & b_y & n_y \\ t_z & b_z & n_z \end{bmatrix}\end{aligned}$$

Com isso é encontrada a matriz mudança de base $M_{t,w}$ que leva um vetor das coordenadas do Espaço Tangente para as Coordenadas de Mundo. A matriz $M_{w,t}$ que leva um vetor das Coordenadas de Mundo para o Espaço Tangente é obtida encontrando

$M_{t,w}^{-1}$. Como $M_{t,w}$ é uma matriz ortogonal $M_{t,w}^{-1} = M_{t,w}^T$, logo $M_{w,t} = M_{t,w}^T$ o que permite encontrar $M_{w,t}$ como:

$$M_{w,t} = \begin{bmatrix} t_x & t_y & t_z \\ b_x & b_y & b_z \\ n_x & n_y & n_z \end{bmatrix}$$

A matriz $M_{w,t}$ realiza a mudança de base de vetores nas Coordenadas de Mundo para o Espaço Tangente, possibilitando que os cálculos de iluminação e outros envolvidos nas técnicas de Detalhamento de Superfícies sejam realizados sobre um sistema local da textura. Como inicialmente tratado o Espaço Tangente permite uma redução abrupta dos cálculos que envolvem os vetores normais do Mapa de Normais, contudo se necessário outros vetores poderiam também ser descritos diretamente neste sistema.

3 Detalhamento de Superfícies

Neste capítulo são abordados todos os métodos de Detalhamento de Superfícies que simulam os detalhes da meso-estrutura. Os Métodos Clássicos na seção 3.1 apresentam as técnicas de detalhamento elementares e são tratados separadamente por motivos históricos. Os Métodos Não Iterativos na seção 3.2 propõem o detalhamento de superfícies através de deslocamento das coordenadas de textura em função do ponto em que são observados. Os Métodos Iterativos na seção 3.3 detalham a superfície utilizando estratégias que realizam iterações, como o *Ray Casting*.

3.1 Métodos Clássicos

Os Métodos Clássicos representam as técnicas de detalhamento elementares na construção de detalhes da meso-estrutura e são tratados separadamente por motivos históricos. Na seção 3.1.1 é abordado o *Displacement Mapping* [3], uma técnica que utiliza as informações do Mapa de Níveis e Mapa de Normais para mover os vértices de uma malha. Na seção 3.1.2 é tratado o método de *Bump Mapping* [1, 17] e na seção 3.1.3 o *Normal Mapping* [2, 9, 5] que consistem em uma extensão do modelo de tonalização *Phong*. A seção 3.1.4 aborda os principais problemas encontrados no *Bump Mapping* e *Normal Mapping*, motivando uma nova proposta de Detalhamento de Superfícies baseada no Mapeamento de Texturas.

3.1.1 Displacement Mapping

O *Displacement Mapping* foi criado em 1984 por *Cook* com o objetivo de criar detalhes em malhas. A técnica utiliza um *Displacement Map*, semelhante ao Mapa de Níveis, para modificar a posição dos vértices de uma malha em uma direção geralmente dada pelos vetores normais. Com o surgimento dos *Shaders* e da adição do *texture fetch*, uma instrução de acesso às coordenadas de textura a partir do *vertex shader* [22], o *Displacement Mapping* pode ser implementado nas *GPUs* modernas, como feito em *GPU based interactive displacement mapping* [25]. Onde um vértice com posição P_0 têm sua

coordenada de textura (u, v) utilizada para acessar o Mapa de Níveis e Mapa de Normais no *vertex shader*, obtendo respectivamente sua altura relativa $H(u, v)$ e seu vetor normal $\vec{N}(u, v)$. O posicionamento final P do vértice é dado pela Eq. (3.1).

$$P = P_0 + H(u, v)\vec{N}(u, v) \quad (3.1)$$

A Figura 3.1 mostra o detalhamento que uma malha plana ganha após a utilização da técnica. Os vértices são repositionados na direção das normais definidas por um Mapa de Normais e modularizados por um Mapa de Níveis.

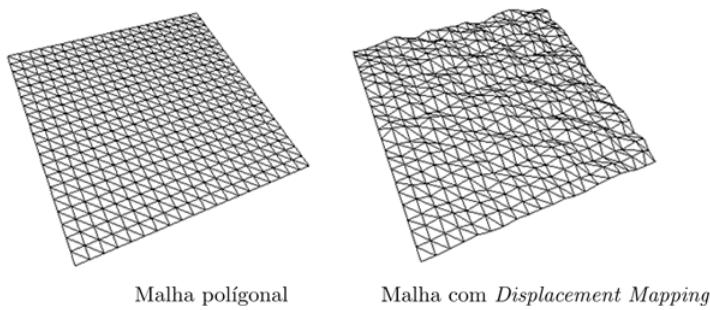


Figura 3.1: Os detalhamentos ganhos com a aplicação do *Displacement Mapping* sobre uma malha plana.

Esta técnica permite facilmente a construção de geometrias detalhadas, porém quando Mapas de Níveis de alta frequência são utilizados são necessárias malhas com número elevado de vértices para que bons resultados sejam obtidos, tornando a técnica computacionalmente custosa. Como discutido na introdução deste trabalho deseja-se a construção de técnicas que limitem o número de polígonos necessários para a modelagem da meso-estrutura. Contudo, o que vemos aqui é basicamente uma forma de adicionar detalhes dinamicamente, sem uma preocupação com a diminuição de polígonos. Sendo assim, o *Displacement Mapping* não é muito útil para a construção de meso-estruturas de alto nível de detalhamento, sendo aqui incluído apenas por motivos históricos.

3.1.2 Bump Mapping

O *Bump Mapping* proposto originalmente por *Blinn* é uma extensão do modelo de tonalização *Phong* que utiliza o Mapa de Níveis para a criação de detalhes no nível da

meso-estrutura. Ele foi o primeiro método que propunha a criação de detalhes na superfície sem a necessidade do uso de malhas finas, o que acabou influenciando a construção de praticamente todas as técnicas tratadas neste trabalho. No modelo de tonalização *Phong* a normal dos vértices de um polígono são linearmente interpolados e as cores em seu interior são obtidas calculando-se o modelo de iluminação em cada um dos pontos da superfície. Segundo a proposta do *Bump Mapping*, os detalhes da meso-estrutura poderiam ser modelados alterando-se os vetores normais interpolados em cada posição de acordo com o relevo descrito pelo Mapa de Níveis, permitindo a simulação de rugosidades e pequenas deformações. A Figura 3.3 mostra um triângulo com normal \vec{n}_0 no ponto (u, v) obtido através da interpolação linear das normais dos vértices e os vetores Tangente \vec{t} e Bi-tangente \vec{b} . O detalhamento é simulado alterando-se \vec{n}_0 de acordo com os gradientes de nível de altura horizontal Δx e vertical Δy do Mapa de Níveis que modificam o módulo respectivamente de \vec{t} e \vec{b} , permitindo obter a nova normal \vec{n}_1 de acordo com a Eq. (3.2). A nova normal \vec{n}_1 é então utilizada na computação do modelo de iluminação fazendo com que o relevo descrito no Mapa de Níveis seja perceptível na superfície do triângulo.

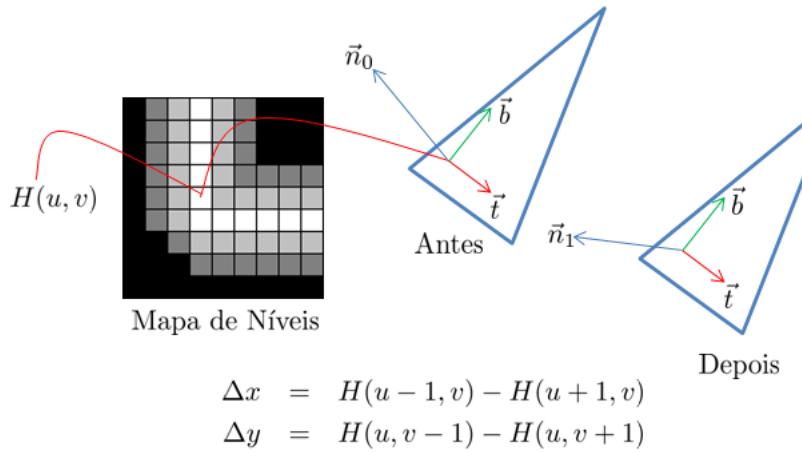


Figura 3.2: A modificação da normal \vec{n}_0 proposto pelo *Bump Mapping* a partir de um Mapa de Níveis.

Na Eq. (3.2) a normal \vec{n}_0 tem sua direção alterada de modo que variações dos gradientes de altura Δx e Δy do Mapa de Níveis influenciem em que proporção a nova normal \vec{n}_1 deve estar dirigida sobre os vetores \vec{t} e \vec{b} do Espaço Tangente.

$$\vec{n}_1 = \vec{n}_0 + \Delta x \vec{t} + \Delta y \vec{b} \quad (3.2)$$

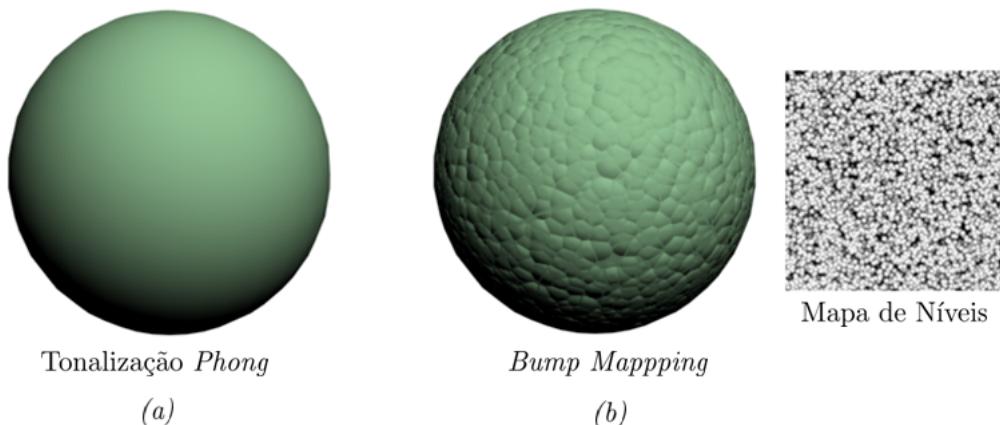


Figura 3.3: Uma esfera (a) submetida ao modelo de tonalização *Phong* e (b) ao *Bump Mapping* utilizando o Mapa de Níveis mostrado.

A Figura 3.3 mostra uma comparação entre o modelo de tonalização *Phong* e o *Bump Mapping* aplicados sobre uma esfera. A esfera em (a) é gerada utilizando a tonalização *Phong*, o que não permite a evidenciação de qualquer detalhamento, já que as normais usadas no cálculo de iluminação são simplesmente interpolações lineares daquelas definidas nos vértices. Em (b) o *Bump Mapping* é aplicado sobre a esfera, o que permite alteração das direções dos vetores normais que utilizados no cálculo de iluminação simulam aproximadamente o relevo descrito pelo Mapa de Níveis. Os detalhes criados pelo *Bump Mapping* permitem que uma sensação de rugosidade previamente não descrita na malha seja exibida sob baixos custos computacionais. O *Bump Mapping* é uma técnica simples que possibilita o detalhamento da meso-estrutura sem que o número de polígonos na malha interfira em sua qualidade final. Porém, os efeitos de silhueta, *motion parallax*, auto-oclusão e auto-sombreamento não são perceptíveis, já que a técnica utiliza essencialmente o modelo de tonalização *Phong*.

3.1.3 Normal Mapping

O *Normal Mapping* [2, 9, 5] semelhante ao *Bump Mapping* pode ser também encarado como uma extensão do modelo de tonalização *Phong*. A idéia é essencialmente a mesma, modificar as normais da superfície para que ao serem utilizadas no cálculo de iluminação, exibam um determinado detalhamento da meso-estrutura. A diferença entre as técnicas é a maneira como as normais são obtidas. No *Normal Mapping* um Mapa de Normais é

mapeado sobre uma superfície, de maneira que a cada ponto (u, v) que se deseja calcular a iluminação utilize a normal $\vec{N}(u, v)$. O *Bump Mapping* por sua vez usa um Mapa de Níveis para realizar estimativas do relevo e modificar a normal interpolada dos vértices. Sendo assim, o *Normal Mapping* obtém diretamente as normais da superfície utilizando-as nos cálculos de iluminação, fazendo com que ela tenha uma melhor eficiência do que o *Bump Mapping* nas arquiteturas de *GPUs* modernas.

A Figura 3.4 compara o Mapeamento de Textura utilizando a tonalização *Phong* com o *Normal Mapping*, considerando uma mesma textura difusa e superfície com 4 vértices. O *Normal Mapping* permitiu o aparecimento da sensação de um relevo com pequenos deslocamentos e rugosidades geometricamente não descritos.

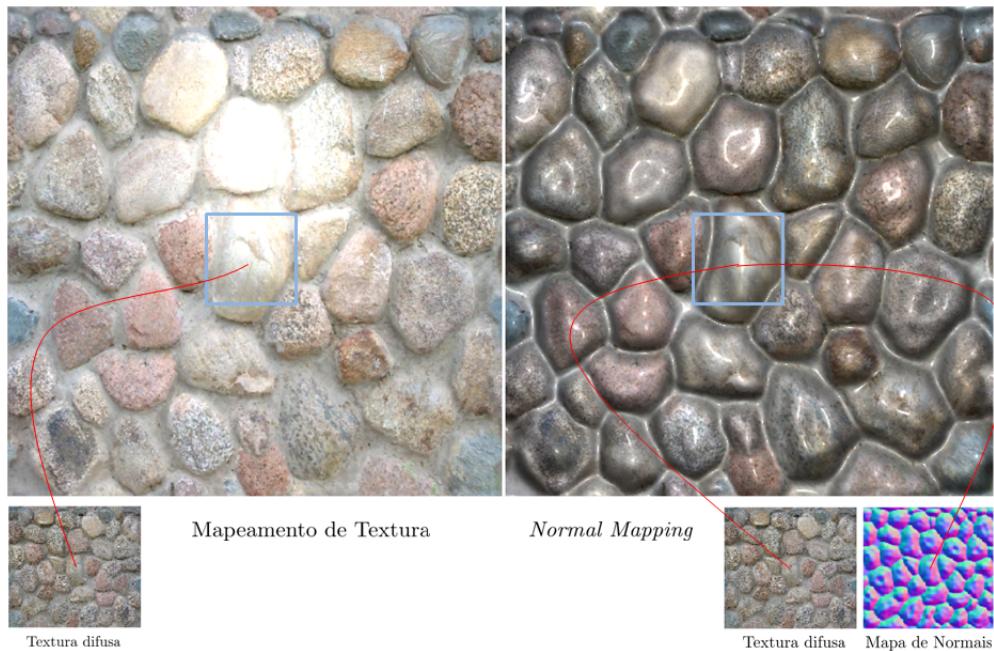


Figura 3.4: Comparativo de uma superfície com Mapeamento de textura utilizando tonalização *Phong* e o *Normal Mapping*.

A Figura 3.5 mostra outro comparativo entre um Mapeamento de Textura sem iluminação e o *Normal Mapping*, o que permite observar as enormes diferenças de qualidades conseguidas pela técnica.

Os comparativos realizados com o Mapeamento de Texturas e o modelo de tonalização *Phong* mostram os maiores níveis de qualidade obtidos com o *Normal Mapping*. Contudo, uma comparação entre os detalhes de uma malha com grande número de vértices se torna necessária. A Figura 3.7 mostra uma malha simples onde foi aplicado o *Normal Mapping* na tentativa de simular o relevo descrito pela malha fina da Figura 3.6. Neste

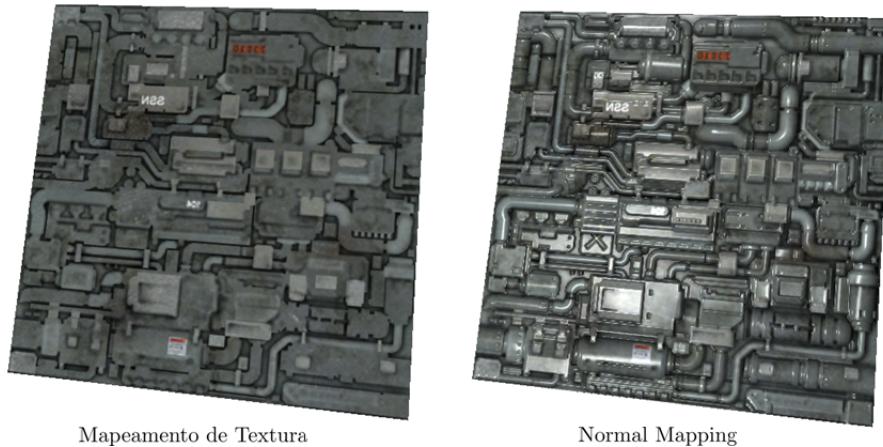


Figura 3.5: Comparativo entre o Mapeamento de Textura e o *Normal Mapping*.

caso, fica evidente que o *Normal Mapping* não consegue re-criar com perfeição todos os detalhes. Isto ocorre porque a técnica não realiza qualquer alteração no Mapeamento de Textura que leve em conta o ponto de observação. Fazendo com que os efeitos de silhueta, *motion parallax*, auto-occlusão e auto-sombreamento não estejam perceptíveis.

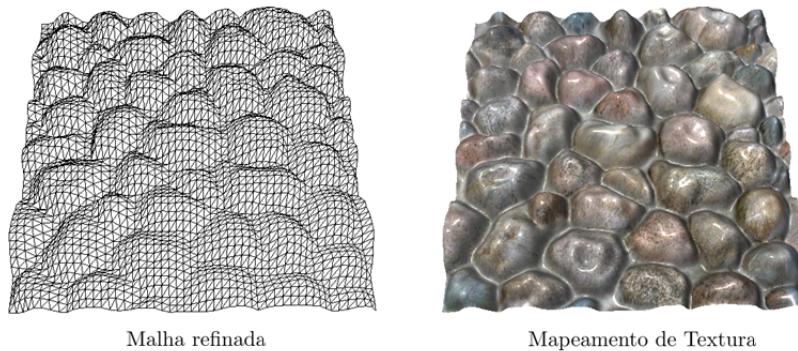


Figura 3.6: Representação de um relevo acidentado a partir de uma malha com número elevado de polígonos.

O *Normal Mapping* também apresenta níveis de qualidade variáveis que dependem da posição em que um observador visualiza a superfície. Quando a superfície é visualizada perpendicularmente, são obtidos os maiores graus de realismo. Porém, conforme o ângulo entre a direção de observação e a superfície diminui, passa haver uma degradação crescente dos detalhes. Isto é mostrado na Figura 3.8, onde as regiões mais próximas do observador e mais altas deveriam sobrepor as baixas. Como isto não ocorre, a ilusão de relevo se deteriora à medida que o ângulo entre o raio de visão e a superfície

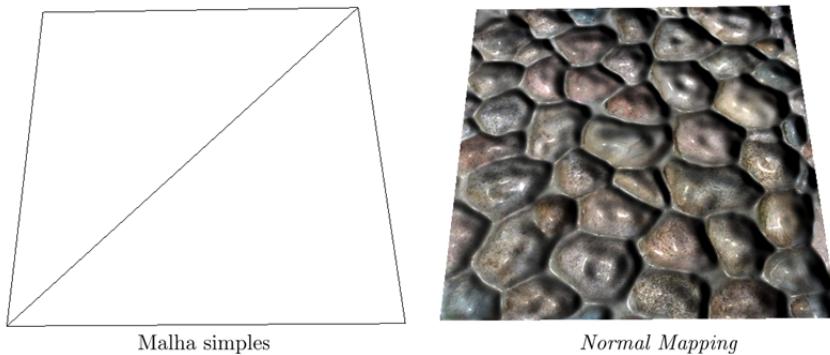


Figura 3.7: O *Normal Mapping* sendo aplicado em uma malha simples, com pequenos números de vértices.

são diminuídos. Tal deterioração não ocorre para visões superiores devido a poucas sobreposições. Mesmo com estas deficiências o *Normal Mapping* ainda é hoje muito utilizado por oferecer um baixo custo computacional e implementação simples nas *GPUs*, como pode ser visto em sua codificação no Apêndice A.2.



Figura 3.8: O *Normal Mapping* sob um pequeno ângulo entre o plano da textura e raio visão.

3.1.4 Refinamento do Normal Mapping: Generalização de Problema

O detalhamento provido pelo *Normal Mapping* causa bons resultados para visões superiores da superfície, contudo, conforme o ângulo entre a direção de observação e a superfície decai, evidencia-se grande perca de qualidade. Isto ocorre porque o *Normal Mapping* realiza o detalhamento da superfície utilizando exclusivamente o modelo de tonalização *Phong*, não levando em consideração os níveis do relevo no Mapeamento de Textura. Quando o observador está visualizando a superfície em ângulos grandes entre o raio de visão e o plano da textura, a maior parte do relevo está visível e a não percepção

das auto-ocluções não afeta tanto a sensação de detalhamento. Quando ângulos menores entre o raio de visão e o plano da textura são evidenciados, as auto-ocluções deveriam ocorrer com maior frequência, o que degrada a sensação de detalhamento. Outro efeito inexistente no *Normal Mapping* é o *motion-parallax* que permite transmitir a sensação de profundidade nos detalhes, fazendo com que, à medida que o observador se movimenta sobre a superfície, os detalhes que estão mais próximos movam-se com maior velocidade em relação àqueles mais distantes. Os efeitos de auto-oclução e *motion parallax* podem ser obtidos utilizando um detalhamento da meso-estrutura que leve em consideração o Mapa de Níveis no Mapeamento de Texturas.

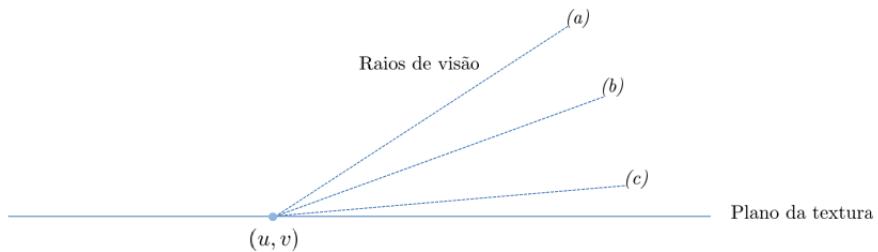


Figura 3.9: Um observador visualizando a superfície plana da textura em perfil sob três posições distintas (a) , (b) e (c) que implicam em sucessivos menores ângulos entre o raio de visão e o plano da textura.

O esquema da Figura 3.9 mostra uma superfície planar em perfil, onde fora mapeada uma textura difusa sob a visão de um observador em três momentos distintos (a) , (b) e (c) . Conforme o observador varia sua posição de (a) para (c) tomando ângulos cada vez menores entre o raio de visão e o plano da superfície, ele visualiza o mesmo *texel* sob as mesmas coordenadas de textura (u, v) , o que não permite a percepção de auto-oclução. Para que auto-ocluções passem ser perceptíveis, um observador em posições distintas teria que visualizar diferentes *texels* sob uma mesma coordenada de textura, simulando obstruções das regiões mais baixas pelas mais altas. Sendo assim, as auto-ocluções podem ser tratadas utilizando um Mapa de Níveis, que especifique todo o relevo a ser visualizado na textura difusa. Isto permite identificar as regiões da textura que deveriam estar visíveis ou não sob um ponto de vista. Para isso, introduzimos o problema que é a base para todas as técnicas aqui abordadas, através do esquema da Figura 3.10.

A Figura 3.10 mostra um esquema onde um observador visualiza uma superfície descrita por uma Mapa de Níveis. O raio traçado a partir do ponto de observação até o plano da textura, indica que a coordenada de textura (u, v) é aquela utilizada para se

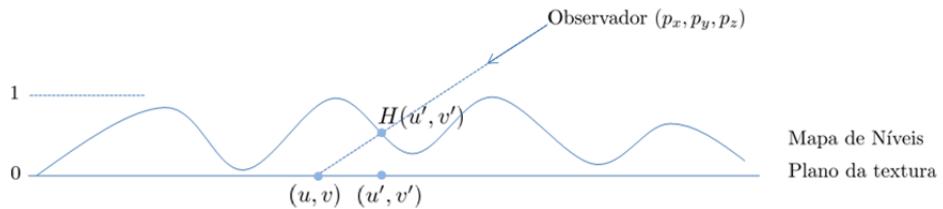


Figura 3.10: Esquema mostrando a relação do Mapa de Níveis, plano da textura e raio de visão do observador.

obter o *texel* naquele ponto. Com a adição do Mapa de Níveis ilustrado acima do plano da textura, torna-se possível identificar qual *texel* o observador deveria visualizar sob sua posição atual em relação à meso-estrutura definida pelo Mapa de Níveis. Para isso deve-se realizar uma interseção entre o raio de visão e o relevo descrito pelo Mapa de Níveis, obtendo a nova coordenada de textura (u', v') da qual o *texel* $T(u', v')$ e normal $\vec{N}(u', v')$ serão utilizados para a tonalização do *texel* nas coordenadas (u, v) . A interseção entre o raio de visão e o relevo estipulado pelo Mapa de Níveis, é dado pela Eq. (3.3), onde (p_x, p_y, p_z) é a direção atual do observador.

$$(u', v', H(u', v')) = (u, v, 0) + t(p_x, p_y, p_z) \quad (3.3)$$

Como o tempo computacional é fundamental para aplicações em tempo real, o problema passa a ser encontrar métodos eficientes para se estimar t na Eq. (3.3) e encontrar as coordenadas de textura (u', v') , onde ocorre a interseção entre o raio traçado na direção em que o observador visualiza a superfície e o relevo descrito pelo Mapa de Níveis. Este é o problema central para o detalhamento de superfícies baseados no Mapeamento de Texturas, dos quais as técnicas que permitem solucionar o problema serão tratadas nas seções de Métodos Não Iterativos e Métodos Iterativos.

3.2 Métodos Não Iterativos

Os Métodos Não Iterativos abordam o problema da interseção entre o raio de visão e o relevo descrito pelo Mapa de Níveis da seção 3.1.4 formulando uma heurística não iterativa que garanta bons resultados na maior parte dos casos. Estes métodos não convergem sempre ao encontro da interseção, o que proporciona a visualização de artefatos ou distorções em situações específicas. Contudo, suas implementações nas *GPUs* modernas apresentam

baixo custo computacional, podendo ser aplicadas em cenas de tempo real. Os métodos apresentados neste capítulo são o *Parallax Mapping* na seção 3.2.1, o *Parallax Mapping with Offset Limiting* na seção 3.2.2 e o *Parallax Mapping with Slope Information* na seção 3.2.3, cuja as implementações em *Shaders* podem ser consultadas respectivamente nos Apêndices A.3, A.4 e A.5.

3.2.1 Parallax Mapping

O *Parallax Mapping* [11, 13, 7] simula o efeito *motion parallax* no nível da meso-estrutura abordando o problema da interseção discutido na seção 3.1.4. Ele assume que o Mapa de Níveis possui alturas constantes nas proximidades de uma posição no plano da textura, ou seja, supõe que o relevo descrito têm baixa frequência na variação das alturas. Com isso a Eq. (3.3) poderia ser solucionada encontrando um t aproximado que permite deslocar as coordenadas de textura no Mapeamento de Textura de acordo com o relevo descrito pelo Mapa de Níveis.

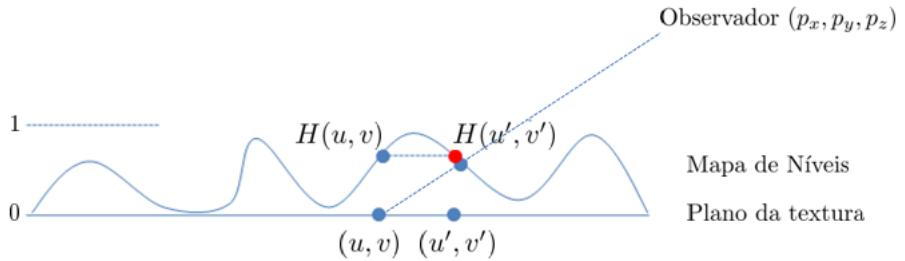


Figura 3.11: O esquema proposto pelo *Parallax Mapping*.

A Figura 3.11 mostra um observador visualizando o plano da textura em uma direção dada por (p_x, p_y, p_z) . A obtenção do ponto de interseção $(u', v', H(u', v'))$ entre o raio de visão e o relevo descrito pelo Mapa de Níveis no *Parallax Mapping* é alcançado supondo que o nível $H(u, v)$ é constante. Isto permite obter t na Eq.(3.3), dada por:

$$(u', v', H(u', v')) = (u, v, 0) + t(p_x, p_y, p_z)$$

Considerando $H(u', v') = H(u, v)$, têm-se:

$$H(u, v) = tp_z \quad (3.4)$$

$$t = \frac{H(u, v)}{p_z} \quad (3.5)$$

Substituindo t em (3.3) obtém-se a Eq. (3.6).

$$(u', v', H(u', v')) = (u, v, 0) + \frac{H(u, v)}{p_z}(p_x, p_y, p_z) \quad (3.6)$$

Como nos interessa somente obter as coordenadas (u', v') pertencentes ao plano da textura é possível eliminar a terceira componente dos cálculos, obtendo assim a Eq. (3.7) do *Parallax Mapping*.

$$(u', v') = (u, v) + \frac{H(u, v)}{p_z}(p_x, p_y) \quad (3.7)$$

O Mapa de Níveis está limitado sobre um intervalo de 0 a 1, o que pode ser inconsistente na simulação de determinados relevos. Por isso normalmente o Mapa de Níveis é re-mapeado através de um fator de s (*Scale*) e uma constante b (*Bias*) e utilizado de maneira empírica para obtenção de melhores resultados pela Eq.(3.8).

$$(u', v') = (u, v) + \left(\frac{sH(u, v) + b}{p_z} \right) (p_x, p_y) \quad (3.8)$$

Com a Eq. (3.8) obtém-se a nova coordenada de textura (u', v') da qual o texel $T(u', v')$ e normal $\vec{N}(u', v')$ serão utilizados para a tonalização do texel nas coordenadas (u, v) . Sendo assim, o *Parallax Mapping* realiza um deslocamento das coordenadas de textura à medida que um observador se movimenta sobre a superfície tornando possível o aumento da sensação de profundidade com o efeito de *motion parallax*. Vale ressaltar aqui que a técnica não implementa verdadeiramente o efeito de oclusão porque ela simplesmente realiza um deslocamento das coordenadas de textura em relação a posição do observador, não descartando quaisquer regiões que deveriam estar sobrepostas de acordo com o relevo descrito pelo Mapa de Níveis.

A Figura 3.12 mostra um comparativo entre o *Normal Mappping* e o *Parallax Mapping* onde pode-se evidenciar uma pequena vantagem na sensação de profundidade obtida pelo *Parallax Mapping*. As diferenças não são maiores pois sob este ângulo o efeito de *motion parallax* ou o deslocamento das coordenadas de textura do *Parallax Mapping* são pequenos.

A Figura 3.13 mostra um novo comparativo entre o *Normal Mappping* e o

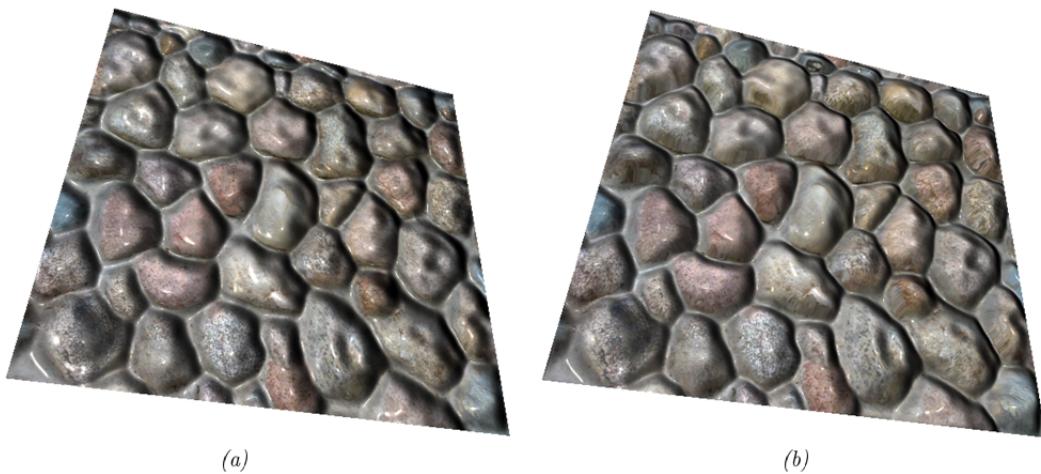


Figura 3.12: Aplicação do *Normal Mapping* em (a) e do *Parallax Mapping* em (b) sobre um mesmo plano a um ângulo grande entre o raio de visão e o plano da textura

Parallax Mapping agora sob ângulos menores entre o raio de visão e o plano da textura. Neste caso o *Parallax Mapping* aplica maiores deslocamentos nas coordenadas de textura o que permite aumentar a sensação do efeito *motion parallax*.

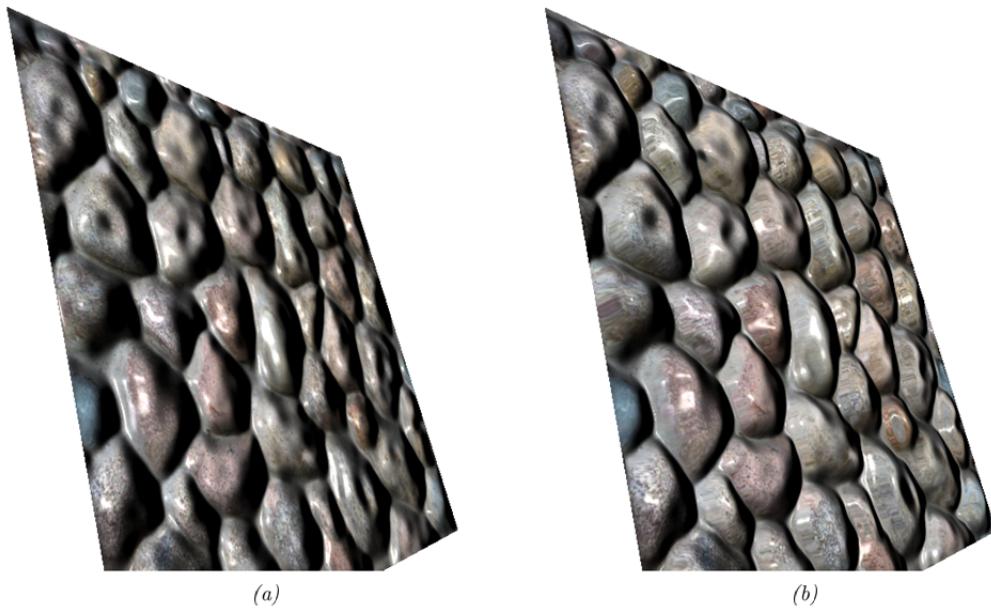


Figura 3.13: Aplicação do *Normal Mapping* em (a) e do *Parallax Mapping* em (b) sobre um mesmo plano a um ângulo médio entre o raio de visão e o plano da textura.

O *Parallax Mapping* melhora os resultados obtidos em relação ao *Normal Mapping*, pois é possível evidenciar uma sensação de profundidade exibida pelo efeito *motion parallax*, contudo, ocorrem uma série de problemas devido a sua simplicidade. A suposição de que o Mapa de Níveis têm alturas constantes nas proximidades de um

ponto é uma boa aproximação para ângulos grandes, no entanto, para pequenos ângulos ocorre uma perca de precisão indesejável. Isto pode ser visto facilmente no esquema da Figura 3.14, onde o ponto (u', v') se distancia da real interseção do raio de observação com o relevo do Mapa de Níveis. Este caso pode ser visto na Figura 3.15, onde pontos que não convergem nas proximidades da interseção passam a gerar distorções na textura.

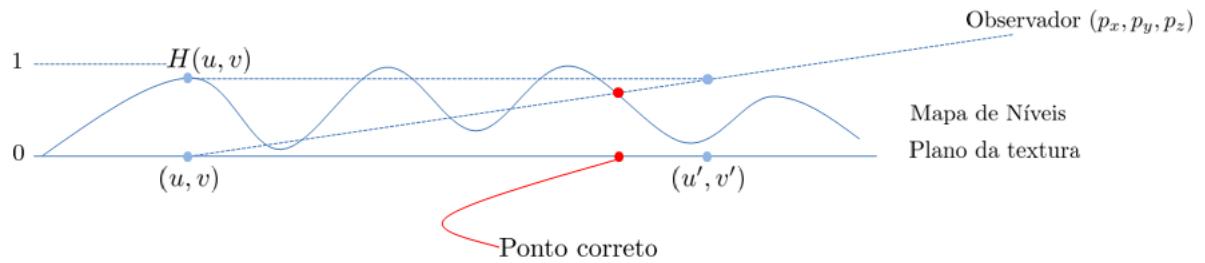


Figura 3.14: Um caso de não convergência ao ponto de interseção do *Parallax Mapping*: quando ocorrem ângulos pequenos entre o raio de visão e o relevo descrito no Mapa de Níveis a aproximação $H(u', v') = H(u, v)$ origina pontos de interseção distantes dos corretos.

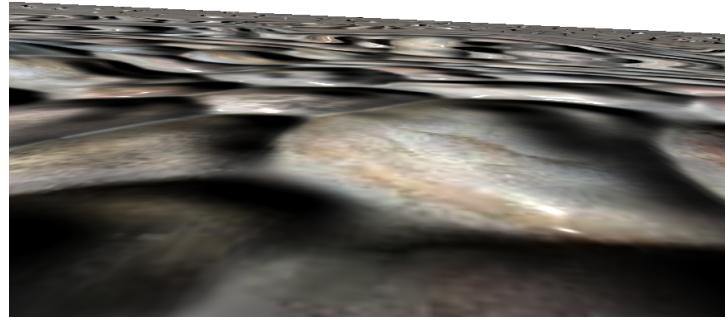


Figura 3.15: A ocorrência de distorções na textura frequentemente vista no *Parallax Mapping* quando são evidenciados ângulos pequenos entre o raio de visão e o plano da textura.

Uma desvantagem da aproximação utilizada é que esta apresenta bons resultados apenas quando aplicada a Mapas de Níveis com padrões no relevo, como paredes de tijolos e similares. Quando a técnica é aplicada a modelos com Mapas de Níveis sem padrões ou níveis de alta frequência, o efeito visual gerado é de baixa qualidade, como pode ser observado na parte superior esquerda da Figura 3.16.

Uma outra limitação do *Parallax Mapping* que ocorre nos casos onde se têm um pequeno ângulo entre o raio de visão e o plano da textura é a presença do $\frac{1}{p_z}$ na Eq.



Figura 3.16: O *Parallax Mapping* aplicado sobre um Mapa de Níveis sem padrões de altura repetidos ou aproximados.

(3.8). Nestes casos $\lim_{p_z \rightarrow 0} \frac{1}{p_z}$, fazendo com que as coordenadas (u', v') obtenha *texels* de forma aleatória, causando distorções e ocasionalmente o aparecimento de *pixels nadando*. Os *pixels nadando* são aqueles obtidos de maneira aleatória que não tornam continuas o detalhamento das texturas, parecendo um grupo de *pixels* que estão deslocados do restante em uma região da textura.

A aproximação da técnica também exibe problemas quando são utilizadas macro-estruturas não planares, pois os parâmetros s e b da Eq. (3.8) não conseguem ser adaptados sem que haja distorções na textura causadas pela observação de partes da superfície a pequenos ângulos. A Figura 3.17 mostra um cubo tendo duas faces visualizadas com *Parallax Mapping* e após uma manipulação exaustiva dos parâmetros não foi possível retirar totalmente as distorções. Isto mostra que a técnica foi desenvolvida para macro-estruturas planares estáticas e não para malhas que se movam pelo ambiente ou exibam uma forma complexa.

O *Parallax Mapping* apresenta uma melhora de qualidade em relação ao *Normal Mapping*, principalmente a ângulos médios entre o raio de visão e o plano da textura. Ele permite uma implementação eficiente nas *GPUs* modernas, contudo apresenta uma série de limitações para os casos onde se têm um pequeno ângulo entre o raio de visão e o plano da textura, bem como para Mapas de Níveis com relevos pouco padronizados e macro-estruturas não planares.



Figura 3.17: O *Parallax Mapping* aplicado sobre a superfície de um cubo.

3.2.2 Parallax Mapping with Offset Limiting

Um dos problemas do *Parallax Mapping* apresentado na seção 3.2.1 é a tendência de gerar distorções e *pixels nadando* conforme os ângulos de visão diminuem em relação ao plano da textura. O *Parallax Mapping with Offset Limiting* [27] propõe uma modificação na Eq. (3.8) que remove o termo $\frac{1}{p_z}$, fazendo com que os problemas encontrados quando $p_z \rightarrow 0$ sejam removidos. Para isso o método utiliza um deslocamento constante $H(u, v)$, modulando a distância que se percorre sobre a superfície da textura na obtenção da interseção (u', v') , como mostra a Eq. (3.9).

$$(u', v') = (u, v) + H(u, v)(p_x, p_y) \quad (3.9)$$

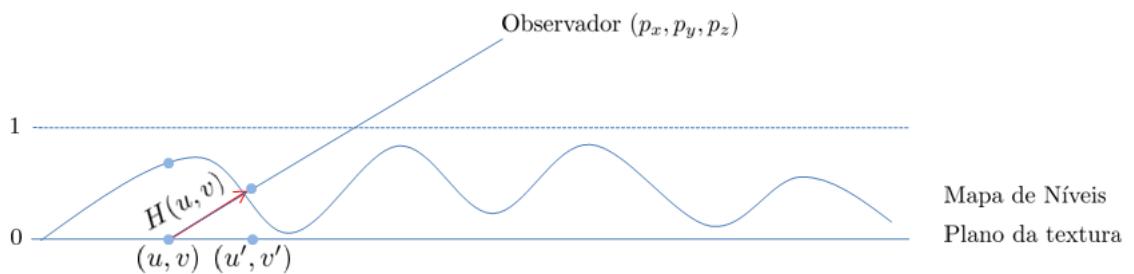


Figura 3.18: Esquema para um exemplo de deslocamento proposto pelo *Parallax Mapping with Offset Limiting*.

A Figura 3.18 mostra o nível de altura $H(u, v)$ modulando um vetor na direção de visão (p_x, p_y, p_z) com o objetivo de encontrar a interseção (u', v') aproximada entre o

raio de visão e o relevo do Mapa de Níveis. Para com prender porque isto é válido basta analisar a equação do *Parallax Mapping* representada por $(u', v') = (u, v) + \frac{H(u, v)}{p_z}(p_x, p_y)$. Quando a superfície é observada sob ângulos grandes $p_z \approx 1$ e o deslocamento é dado por $H(u, v)(p_x, p_y)$. Quando a superfície é observada a ângulos muito pequenos $p_z \ll p_x, p_y$ o que gera coordenadas (u', v') com valores altos e aleatórios já que $0 \leq p_z \leq 1$, então poderia se determinar um limitador para que não se gere deslocamentos muito grandes, sendo assim, adota-se $H(u, v)(p_x, p_y)$ como limitador. Com isso em qualquer caso o maior deslocamento possível será dado por $H(u, v)$ que no máximo é 1, fazendo com que deslocamentos muito grandes não possam mais ocorrer. Tal aproximação para encontrar a interseção pode parecer aleatória, com tudo o método utiliza o fator s (*Scale*) e uma constante b (*Bias*) para que o deslocamento sobre a textura seja adaptado de acordo com o relevo descrito pelo Mapa de Níveis, como mostrado na Eq. (3.10).

$$(u', v') = (u, v) + (sH(u, v) + b)(p_x, p_y) \quad (3.10)$$

Com a Eq. (3.10) obtém-se a nova coordenada de textura (u', v') da qual o texel $T(u', v')$ e normal $\vec{N}(u', v')$ serão utilizados para a tonalização do texel nas coordenadas (u, v) gerando o detalhamento da meso-estrutura definido pelo Mapa de Níveis.

A Figura 3.19 mostra a comparação entre o *Parallax Mapping* e o *Parallax Mapping With Offset Limiting* para o caso onde $p_z \rightarrow 0$. Fica evidente a diminuição das distorções quando se têm um ângulo pequeno entre o raio de visão e o plano da textura.



Figura 3.19: O *Parallax Mapping* em (a) e o *Parallax Mapping With Offset Limiting* em (b) a um pequeno ângulo entre o raio do observador e o plano da textura.

Com isso a técnica pode ser aplicada sobre macro-estruturas não planares e em movimento, já que as distorções não são mais exibidas. A Figura 3.20 mostra novamente um comparativo entre as duas técnicas sobre um cubo, em (a) o *Parallax Mapping* mesmo com uma exaustiva manipulação dos parâmetros s e b não permitiu a completa remoção

das distorções nos pequenos ângulos de visão. Em (b) o *Parallax Mapping with Offset Limiting* pode mapear os detalhes sem quaisquer distorções, mesmo naquelas regiões que apresentam um pequeno ângulo com a face do cubo.

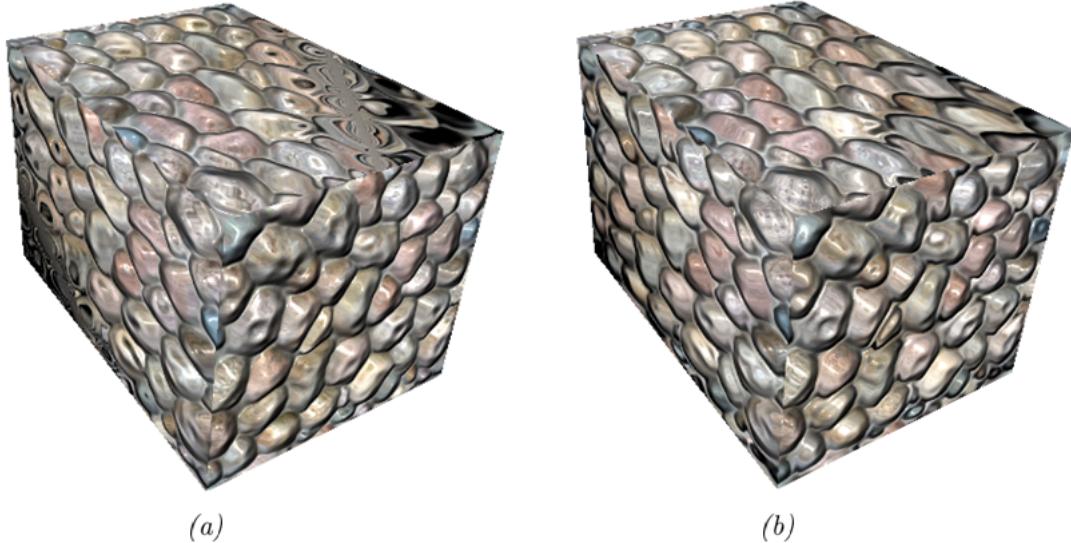


Figura 3.20: O *Parallax Mapping* em (a) e o *Parallax Mapping With Offset Limiting* em (b) aplicados sobre um cubo.

O *Parallax Mapping with Offset Limiting* propõe uma modificação simples ao *Parallax Mapping* que causa melhorias apenas para os casos onde $p_z \rightarrow 0$. Tal modificação ainda permite a obtenção de melhor desempenho, já que a Eq. (3.9) possui uma operação de divisão a menos que a Eq. (3.7) do *Parallax Mapping*. Contudo, o uso de Mapas de Níveis com relevos pouco padronizados ainda gera as mesmas distorções do *Parallax Mapping*, como tratado na seção 3.2.1.

3.2.3 Parallax Mapping with Slope Information

Os métodos de detalhamento Não Iterativos abordados até o momento assumem que as alturas do Mapa de Níveis são sempre constantes nas proximidades de uma coordenada de textura. No entanto, Mapa de Níveis e Mapa de Normais possuem mais informações que ainda não foram exploradas pelas técnicas anteriores, para o encontro da interseção do problema generalizado abordado na seção 3.1.4. Uma melhor aproximação consegue ser obtida se for assumido que a superfície permanece planar, mas seus vetores normais agora podem ser dados arbitrariamente pelo Mapa de Normais [Mátyás Premecz et al. 2006]. O *Parallax Mapping with Slope Information* [13] utiliza o vetor normal nas coordenadas de

textura (u, v) para obter uma estimativa da inclinação no relevo do Mapa de Níveis. Isto permite a obtenção aproximada do ponto de interseção entre o raio de visão e o relevo do Mapa de Níveis, como mostra a Figura 3.21. O método propõe que seja criado um plano com vetor normal $\vec{N}(u, v)$ que simultaneamente intercepte o raio de visão em uma coordenada que será a interseção estimada (u', v') .

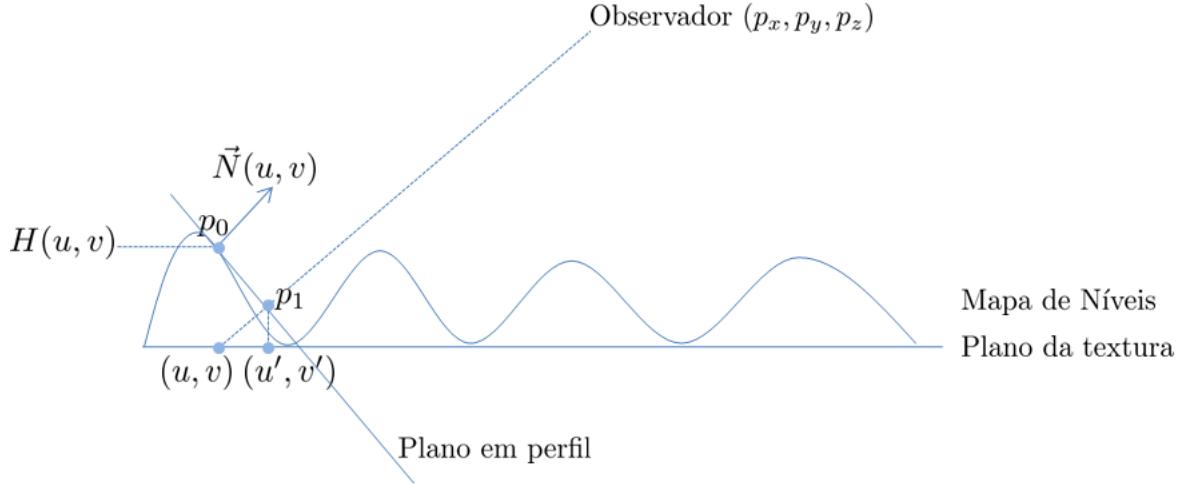


Figura 3.21: Esquema proposto pelo *Parallax Mapping with Slope Information*.

Orientando-se pela Figura 3.21 pode-se obter a interseção aproximada em (u', v') entre o raio de visão e o plano de textura como tratado na seção 3.1.4 encontrando t em $(u', v', H(u', v')) = (u, v, 0) + t(p_x, p_y, p_z)$. Para isso parte-se da propriedade do plano em que sua normal $\vec{N}(u, v)$ deve sempre ser perpendicular a qualquer vetor formado pela subtração de pontos pertencentes ao plano. Considerando $\vec{p} = (p_x, p_y, p_z)$ a direção do raio de visão unitário e fazendo $p_0 = (u, v, H(u, v))$ e $p_1 = (u, v, 0) + t(p_x, p_y, p_z)$ os pontos pertencentes ao plano pode-se obter t partindo da Eq. (3.11).

$$\vec{N}(u, v) \cdot [(u, v, 0) + t(p_x, p_y, p_z)] = \vec{N}(u, v) \cdot [(u, v, H(u, v))] \quad (3.11)$$

$$t = \frac{-uN_x(u, v) - vN_y(u, v) + uN_x(u, v) + vN_y(u, v) + N_z(u, v)H(u, v)}{\vec{N}(u, v) \cdot \vec{p}}$$

$$t = \frac{N_z(u, v)H(u, v)}{\vec{N}(u, v) \cdot \vec{p}} \quad (3.12)$$

Colocando (3.12) em $p_1 = (u, v, 0) + t(p_x, p_y, p_z)$, obtém-se o ponto de interseção estimado p_1 na Eq. (3.13).

$$p_1 = (u, v, 0) + \frac{N_z(u, v)H(u, v)}{\vec{N}(u, v) \cdot \vec{p}}(p_x, p_y, p_z) \quad (3.13)$$

O fator $\frac{1}{\vec{N}(u, v) \cdot \vec{p}}$, resulta na aparição de artefatos semelhantes ao do *Parallax Mapping* a pequenos ângulos entre o plano de textura e o raio de visão. Para que isso não ocorra a mesma metodologia adotada com *Parallax With Offset Limiting* é realizada removendo-se $\vec{N}(u, v) \cdot \vec{p}$, já que ele se torna muito grande a ângulos pequenos. Com isto obtém-se a interseção final (u', v') pela Eq. (3.14) retirando-se o fator problemático e colocando a Eq. (3.13) no espaço das coordenadas de textura.

$$(u', v') \approx (u, v) + N_z(u, v)H(u, v)(p_x, p_y) \quad (3.14)$$

Como nos métodos anteriores esta técnica depende de uma adaptação do Mapa de Níveis para uma melhor representação do relevo, para isso são utilizados o fator de escala s e a constante b , obtendo-se assim a Eq. (3.15) utilizada em sua implementação.

$$(u', v') \approx (u, v) + N_z(u, v)(sH(u, v) + b)(p_x, p_y) \quad (3.15)$$

O ponto de interseção (u', v') agora dado no espaço da textura que possui *texel* $T(u', v')$ e normal $\vec{N}(u', v')$ serão utilizados para a tonalização do *texel* nas coordenadas (u, v) fazendo com que os detalhes da meso-estrutura definidos no Mapa de Níveis sejam representados sobre na superfície. A técnica ainda utiliza uma heurística para encontrar a interseção, porém de maneira mais apurada que a realizada pelo *Parallax Mapping with Offset Limiting*, sendo assim somente o efeito de *motion parallax* pode ser observado.

Vale ainda ressaltar que nem sempre o método resulta em boas aproximações ao ponto de interseção entre o raio de visão e o relevo do Mapa de Níveis. A Figura 3.22 mostra o caso onde esta interseção não converge para a posição correta. Esta situação irá ocorrer em casos muito raros e com maior frequência sob ângulos pequenos entre o raio de visão e o plano da textura.

No comparativo da Figura 3.23 pode-se evidenciar maior detalhamento nos relevos mais distantes do ponto de visão no *Parallax Mapping with Slope Information* do que no *Parallax Mapping with Offset Limiting*. Isto ocorre porque o método converge

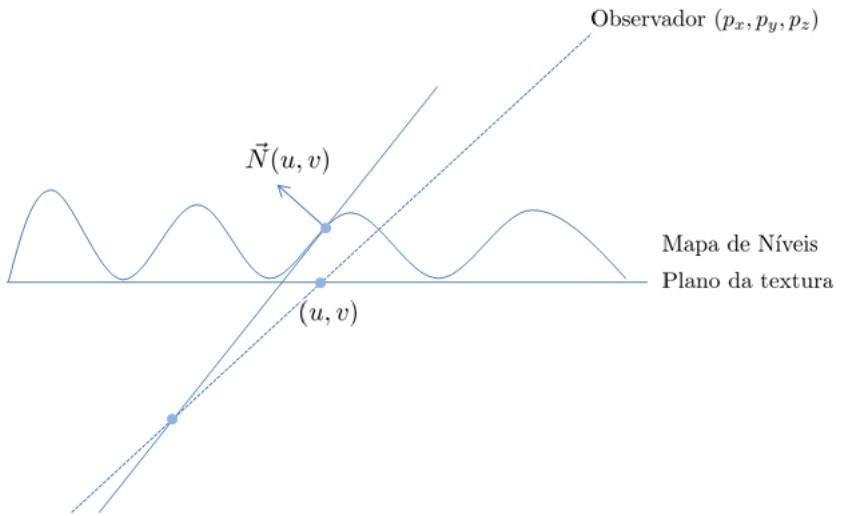


Figura 3.22: Um caso de não convergência: O vetor normal $\vec{N}(u, v)$ induz a formação do plano na direção errada.

para a interseção com maior precisão na maioria dos casos do que a heurística proposta pelo *Parallax Mapping with Offset Limiting*.

Entre todos os métodos Não Iterativos implementados àquele que obteve melhores qualidades gráficas foi o *Parallax Mapping with Slope Information* sob os maiores custos de processamento. Mesmo assim, sua implementação nas *GPUs* atuais mantém custos computacionais baixos, o que torna o método uma boa opção para o detalhamento de meso-estruturas em ambiente como jogos e simuladores.

3.3 Métodos Iterativos

O detalhamento de superfícies baseado em Métodos Iterativos tratam, o problema da interseção abordado na seção 3.1.4 com técnicas que iteram em busca da convergência ao ponto de interseção. Inicialmente será tratado o *Iterative Parallax Mapping* na seção 3.3.1, uma versão iterativa similar ao *Parallax Mapping with Slope Information*. Nas seções seguintes serão tratados os Métodos Iterativos baseados em *Ray Casting*, onde marchas são realizadas sobre os raios de visão em busca do encontro da interseção. Nesta categoria serão abordados os métodos de *Busca Linear* na seção 3.3.2, *Busca Binária* na seção 3.3.3, *Relief Mapping* na seção 3.3.4, *Parallax Occlusion Mapping* na seção 3.3.5 e o *Cone Step Mapping* na seção 3.3.6. Tais métodos garantem na maioria dos casos uma convergência



Figura 3.23: O *Parallax Mapping With Offset Limit* em (a) e o *Parallax Mapping with Slope Information* em (b).

aproximada ao ponto de interseção do problema generalizado, apresentando bons níveis de qualidade gráfica. Seus custos computacionais são superiores aos demais métodos, porém ainda viáveis a Computação de Tempo Real. As implementações de cada um dos métodos podem ser vistas consultando-se os Apêndices A.6, A.7, A.8, A.9, A.10 e A.11.

3.3.1 Iterative Parallax Mapping

O *Iterative Parallax Mapping* [20] utiliza a mesma idéia do *Parallax Mapping with Slope Information*, porém em uma versão iterativa. Este método possui convergência rápida e aproximada na maioria dos casos, apresentando uma melhora na qualidade gráfica obtida em sua versão original. Contudo, o método não garante que a interseção encontrada seja aquela mais próxima do observador e em casos extremos uma convergência se quer é garantida. De qualquer forma nos casos mais frequentes ele obtém bons resultados a um baixo custo computacional [Mátyas Premecz et al. 2006]. A Eq. (3.16) do *Iterative Parallax Mapping* é obtida diretamente da Eq. (3.15) do *Parallax Mapping with Slope Information*, apenas colocando-a de forma iterativamente.

$$(u'_{i+1}, v'_{i+1}) \approx (u_i, v_i) + N_z(u_i, v_i)(sH(u_i, v_i) + b)(p_x, p_y) \quad (3.16)$$

A Figura 3.24 mostra um comparativo entre o *Parallax Mapping with Slope Information* em (a) e o *Iterative Parallax Mapping* em (b) com 4 iterações. Como se pode observar em ângulos grandes entre o raio de visão e o plano da textura pouca diferença é

notada. Na Figura 3.25 é realizado o mesmo comparativo, porém sob ângulos pequenos entre o raio de visão e o plano da textura onde é possível se evidenciar grandes diferenças entre as duas abordagens. As pedras na superfície de (b) estão bem mais evidentes o que permite a visualização de detalhes com maior riqueza.



Figura 3.24: O *Parallax Mapping with Slope Information* em (a) e o *Iterative Parallax Mapping* em (b), sob ângulos grandes entre o raio de visão e o plano da textura.



Figura 3.25: O *Parallax Mapping with Slope Information* em (a) e o *Iterative Parallax Mapping* em (b), sob ângulos pequenos entre o raio de visão e o plano da textura

O *Iterative Parallax Mapping* gera imagens com boa qualidade gráfica através da simulação do efeito de *motion parallax*.

3.3.2 Busca Linear

A Busca Linear [14] é uma técnica utilizada integralmente pelo *Steep Parallax Mapping* [15] e como uma etapa inicial de todos os algoritmos que utilizam o conceito de *Ray Casting* para encontrar a interseção entre o raio de visão e o relevo do Mapa de Níveis tratado na seção 3.1.4. Ela encontra a interseção aproximada realizando uma marcha a partir do observador e na direção de visão a passos constantes até que se encontre um ponto onde a altura seja inferior ao relevo do Mapa de Níveis, considerando este como a interseção aproximada.

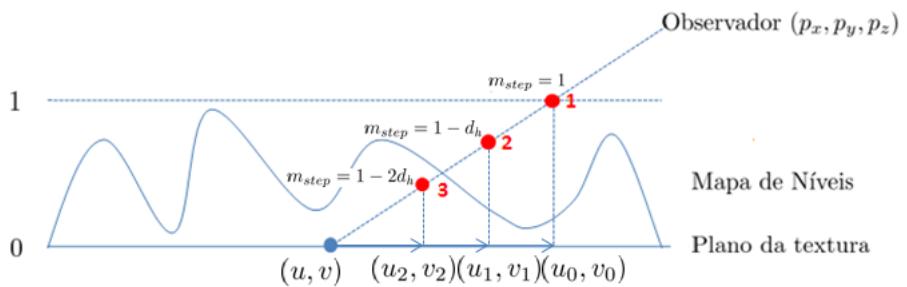


Figura 3.26: Marcha da Busca Linear para encontrar a interseção aproximada entre a direção do observador e o Mapa de Níveis.

A Figura 3.26 mostra um esquema sugerindo as iterações do algoritmo. O primeiro passo é iniciado na posição (u_0, v_0) de altura máxima do Mapa de Níveis na direção do raio de visão. Como a altura atual é superior a $H(u_0, v_0)$, o algoritmo procede realizando uma nova iteração. Obtém-se o ponto 2 onde a altura é superior a $H(u_1, v_1)$, a marcha prossegue com um novo passo, onde é encontrado o ponto 3 que possui altura inferior a $H(u_2, v_2)$, encontra-se assim a interseção aproximada. Logo, o *texel* $T(u_2, v_2)$ e a normal $\vec{N}(u_2, v_2)$ serão utilizados para a tonalização do *texel* nas coordenadas (u, v) .

Os pontos (u_i, v_i) com $i \geq 1$ obtidos a cada iteração da Busca Linear são dados pela Eq. (3.17), onde (u_0, v_0) e (u, v) são, respectivamente as coordenadas de textura dadas na posição em que o Mapa de Níveis alcança sua altura máxima e mínima na direção do raio de visão e m_{step} módula a busca, tendo seu valor decrementado de 1 a 0 pela contante d_h com $0 < d_h \leq 1$.

$$(u_i, v_i) = (u, v) + m_{step}(u_o - u, v_o - v) \quad (3.17)$$

A aproximação da interseção será tão boa quanto menor for d_h , contudo isto

irá exigir um número de iterações cada vez maior, o que prejudica sua eficiência. Deve-se então encontrar um meio termo entre d_h e o número de iterações para que se obtenha boas qualidades de imagem no detalhamento. Quando se aumenta o ângulo entre o raio de visão e o plano da textura menores serão os caminhos a serem percorridos pela marcha, o que permite a obtenção de boas aproximações da interseção com um número reduzido de iterações. Contudo, conforme se diminui o ângulo entre o raio de visão e o plano da textura maiores serão os caminhos a serem percorridos, o que requisita um número de iterações cada vez maior para que se mantenha bons níveis de qualidade.

A Busca Linear não garante convergência a interseção em todos os casos. Quando a marcha é realizada sobre um Mapa de Níveis com relevos apresentando pontas finas um passo do algoritmo pode simplesmente pular a interseção mais próxima do observador e convergir para um ponto qualquer, o que acaba gerando distorções. A Figura 3.27 mostra a marcha pulando a interseção mais próxima do observador entre o passo 2 e 3 e apenas parando no passo 4. Uma maneira de minimizar este problema é utilizar pequenos d_h de modo que tais situações sejam minimizadas. Contudo, quando os valores de d_h são diminuídos o número de passos aumenta, o que diminui a eficiência da técnica.

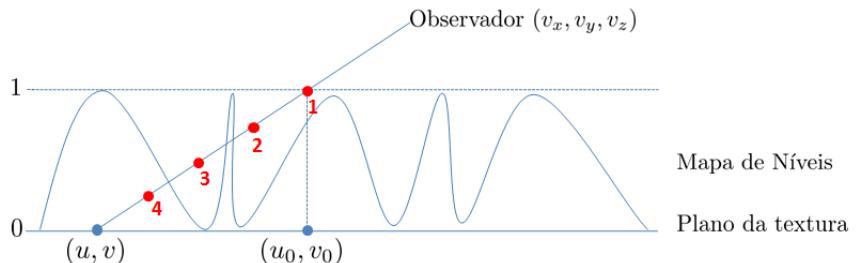


Figura 3.27: O caso em que a Busca Linear perde a interseção mais próxima do observador.

A Figura 3.28 mostra o caso onde se têm ângulos grandes entre o plano da textura e raio de visão, sendo apenas 10 iterações suficientes para gerar boa qualidade no detalhamento sem a presença de distorções. Na Figura 3.29 é analisado o caso onde se têm pequenos ângulos, em (a) percebe-se que as 10 iterações não são mais suficientes para que a interseção seja obtida com boa aproximação, visto a ocorrência de distorções na textura. Porém, aumentando-se bruscamente o número de passos em (b) percebe-se qualidades iguais ou superiores ao caso visto em ângulos grandes entre o raio de visão e o plano da textura.

A Busca Linear e a maior parte das técnicas de detalhamento de superfícies baseadas em *Ray Casting* possibilitam a geração de auto-sombreamento. Este efeito



Figura 3.28: Execução da Busca Linear com 10 iterações sob ângulos grandes entre o plano da textura e raio de visão.

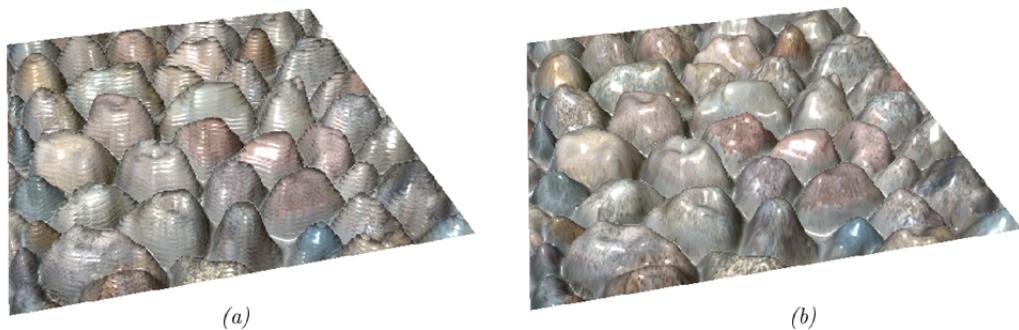


Figura 3.29: A execução da Busca Linear com (a) 10 iterações e (b) 60 iterações sob ângulos pequenos entre o plano da textura e o raio de visão.

cria sombras por obstruções da luz pelo próprio relevo da superfície, ou seja, partes da superfície geram sombras sobre elas mesmas. A construção deste efeito segue exatamente a mesma idéia da Busca Linear, mas neste caso a marcha irá ocorrer a partir da fonte de luz e na direção da interseção mais próxima do observador obtida anteriormente. Caso a marcha encontre uma interseção com o relevo do Mapa de Níveis, o ponto de interseção mais próximo do observador está bloqueado, consequentemente pertence à sombra. Quando a marcha atinge o ponto de interseção mais próximo do observador, o ponto não apresenta um relevo bloqueando a luz, logo não é um ponto de sombra. Ambos os casos são ilustrados na Figura 3.30, em (a) a marcha a partir da fonte de luz atinge o ponto de interseção mais próximo do observador estando este iluminado, em (b) ela encontra uma interseção com o relevo do Mapa de Níveis, logo o ponto de interseção mais próximo do observador está na sombra.

A Busca Linear apresenta níveis altos de qualidade no detalhamento de superfícies permitindo a simulação do *motion parallax*, auto-occlusão e auto-sombreamento.

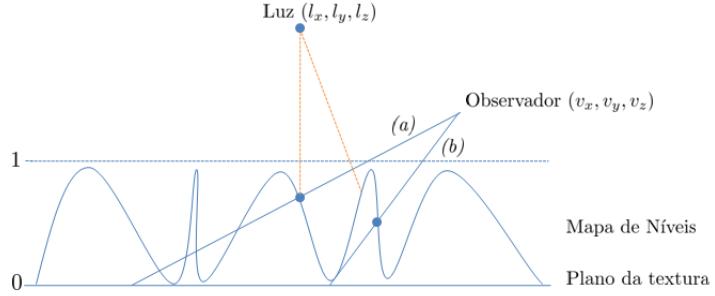


Figura 3.30: Os dois casos possíveis para a geração de auto-sombreamento. Em (a) o ponto de interseção mais próximo do observador é iluminado e em (b) está na sombra.

Porém, o método não garante convergência à interseção mais próxima do observador e a variação das distâncias que as marchas percorrem podem se tornar grandes o suficiente para que o número de iterações seja muito alto, o que pode inviabilizar sua utilização em aplicações de tempo real como jogos e simuladores.

3.3.3 Busca Binária

A Busca Binária [14] é outra técnica utilizada para encontrar a interseção entre o raio de visão e o relevo do Mapa de Níveis tratado no problema da seção 3.1.4. Ela é normalmente utilizada como uma fase de refinamento, como será abordado na seção 3.3.4 pois apresenta uma série de limitações quando aplicada sozinha. O algoritmo inicia com o intervalo formado pelas coordenadas (u, v) sobre a textura e (u_0, v_0) dada pela altura máxima do Mapa de Níveis na direção do raio de visão, como mostra a Figura 3.31. Neste intervalo são realizadas sucessivas divisões, sempre mantendo-se dois pontos extremos em lados opostos, um fóra e outro no interior do relevo descrito pelo Mapa de Níveis.

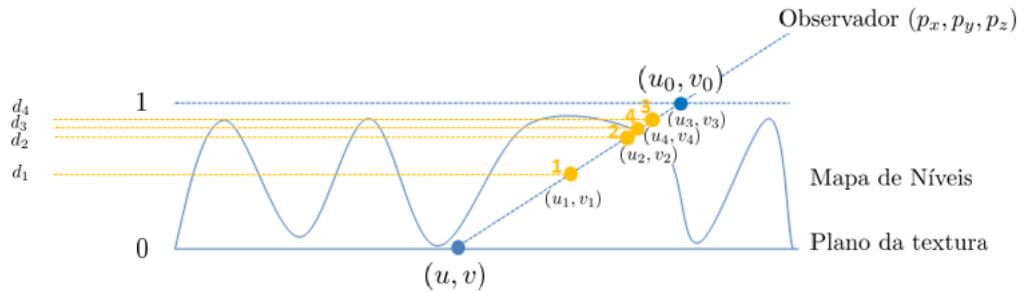


Figura 3.31: As iterações da Busca Binária para encontrar a interseção aproximada entre o relevo do Mapa de Níveis e o raio de visão na direção (p_x, p_y, p_z) .

A Figura 3.31 mostra um raio de visão atingindo o relevo descrito por um Mapa de Níveis e os passos pelo qual a Busca Binária realiza para encontrar a interseção aproximada. Para cada ponto de iteração numerado de 1 a 4 estão dispostos na parte lateral esquerda seus correspondentes valores de altura d_i e nas proximidades suas coordenadas de textura (u_i, v_i) . Na primeira iteração o raio de visão dado entre as coordenadas (u_0, v_0) e (u, v) é dividido ao meio obtendo-se o ponto 1 nas coordenadas (u_1, v_1) . Deve-se agora obter um novo intervalo de pontos extremos que deixem o relevo do Mapa de Níveis em seu interior. Como $H(u_1, v_1) \geq d_1$, o novo intervalo é dado pelas coordenadas (u_0, v_0) e (u_1, v_1) . Uma nova divisão é realizada obtendo-se o ponto 2, como $H(u_2, v_2) \geq d_2$ o novo intervalo será formado pelos pontos de coordenadas (u_0, v_0) e (u_2, v_2) . Realizando uma nova divisão obtém-se o ponto 3. Como $H(u_3, v_3) \leq d_3$ o novo intervalo deverá ser composto pelos pontos de coordenadas (u_3, v_3) e (u_2, v_2) , visto que estes mantêm o relevo do Mapa de Níveis em seu interior. Uma nova divisão é realizada obtendo-se o ponto 4 nas coordenadas (u_4, v_4) e o algoritmo termina, o considerando como o ponto de interseção aproximado. Logo, o texel $T(u_4, v_4)$ e a normal $\vec{N}(u_4, v_4)$ serão utilizados para a tonalização do *texel* nas coordenadas (u, v) . Neste caso se novas iterações fossem realizadas maiores seriam os níveis de precisão, contudo optou-se por um limite de quatro passos.

Nos casos onde o relevo do Mapa de Níveis possui baixas oscilação de alturas ou ângulos grandes entre o raio de visão e plano da textura a Busca Binária costuma convergir, obtendo maiores precisões com aumento do número de iterações, como é o caso da Figura 3.32.



Figura 3.32: Busca Binária sob ângulos grandes entre o plano da textura e o raio de visão.

A Figura 3.33 mostra um caso onde a Busca Binária converge para uma interseção que não é àquela mais próxima da posição do observador. O algoritmo inicia com os

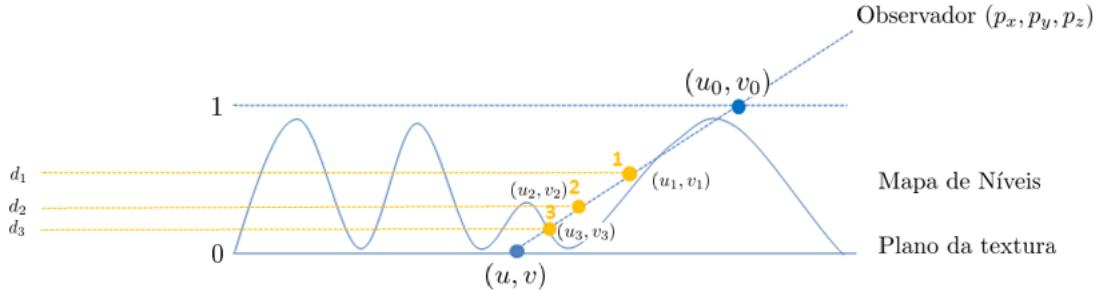


Figura 3.33: Caso de não convergência da Busca Binária.

pontos extremos dados pelas coordenadas (u_0, v_0) e (u, v) . Uma divisão ao meio é realizada neste intervalo obtendo-se o ponto 1 nas coordenadas (u_1, v_1) . Como $H(u_1, v_1) \leq d_1$ o novo intervalo formado é dado pelas coordenadas (u_1, v_1) e (u, v) . O que é incorreto com a tentativa de obter a interseção mais próxima do observador, visto que o intervalo esperado seria o das coordenadas (u_0, v_0) e (u_1, v_1) . As sucessivas iterações realizadas na Busca Binária são baseadas em uma falsa orientação. Isto irá gerar uma convergência no sentido contrário, podendo-se evidenciar artefatos na imagem final, como mostra a Figura 3.34. Tais artefatos, também conhecidos como *pixels nadando*, ocorrem quando não há convergência para a interseção correta, sendo assim observa-se um grupo de *pixels* descontinuados.

Nos casos onde ocorrem variações muito bruscas nas alturas do Mapa de Níveis ou ângulos de visão pequenos entre raio de visão e o plano da textura aumenta-se a probabilidade de iterações sucessivas não convergirem para a interseção mais próxima do observador, sem que um aumento do número de iterações ou qualquer medida possa ser tomada para contornar o problema.



Figura 3.34: Busca Binária sob ângulos pequenos entre o raio de visão e o plano da textura: ocorrem constantemente o aparecimento de *pixels nadando*.

A Busca Binária é um método pouco custoso quando se comparado a Busca Linear. Quando são evidenciados ângulos grandes entre o raio de visão e o plano da textura ambos os métodos produzem bons resultados, como mostra a Figura 3.35. Contudo, para

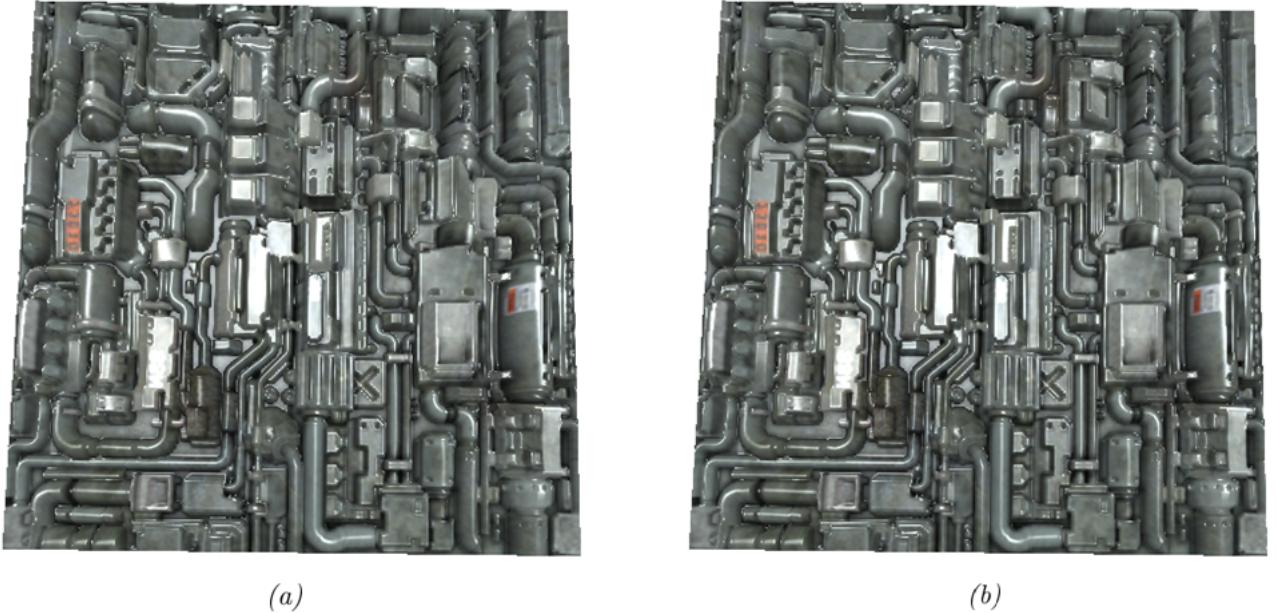


Figura 3.35: Busca Linear em (a) e Busca Binária em (b) limitados a 10 iterações sob ângulos grandes entre o raio de visão e o plano da textura.

os casos onde se tem pequenos ângulos a Busca Linear apresenta distorções que podem ser contornadas com o aumento elevado do número de passos. Na Busca Binária este caso gera distorções e o aumento do número de iterações não garante uma melhora, visto que o algoritmo costuma não convergir para a interseção em algumas regiões do Mapa de Níveis. Com tais limitações a técnica é mais aconselhada quando utilizada em conjunto com outros métodos, mesmo assim ela permite a simulação do efeito *motion parallax* e *auto-occlusão*.

3.3.4 Relief Mapping

O *Relief Mapping* [19] é um método iterativo que utiliza o conceito de *Ray Casting* para o detalhamento de meso-estruturas que obtêm bons níveis de qualidade gráfica simulando os efeitos de *motion parallax*, auto-occlusão e auto-sombreamento. Ele aborda o problema da interseção entre o raio de visão e o relevo do Mapa de Níveis da seção 3.1.4 em duas etapas. A primeira executa uma Busca Linear com passos constantes até encontrar uma altura que esteja abaixo do relevo no Mapa de Níveis. A segunda utiliza os dois últimos deslocamentos da Busca Linear para realizar uma Busca Binária que converge em poucos passos ao ponto de interseção.

A Figura 3.36 mostra os passos realizados pela Busca Linear da primeira etapa

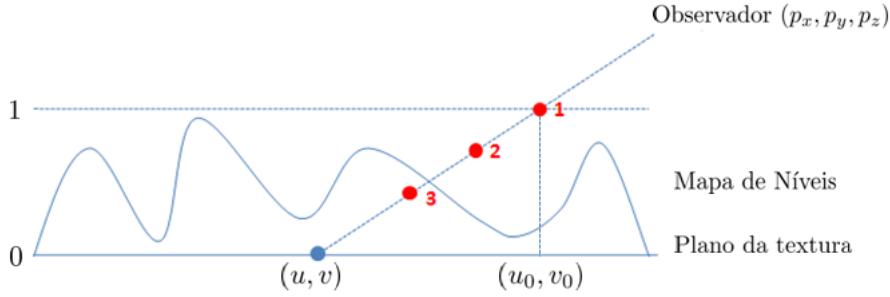


Figura 3.36: Realização da Busca Linear na primeira etapa.

do *Relief Mapping*. São obtidos sucessivamente os pontos 1, 2 e 3 na direção (p_x, p_y, p_z) do observador. Quando o ponto 3 é atingido a primeira etapa é finalizada, já que este se encontra abaixo do relevo do Mapa de Níveis.

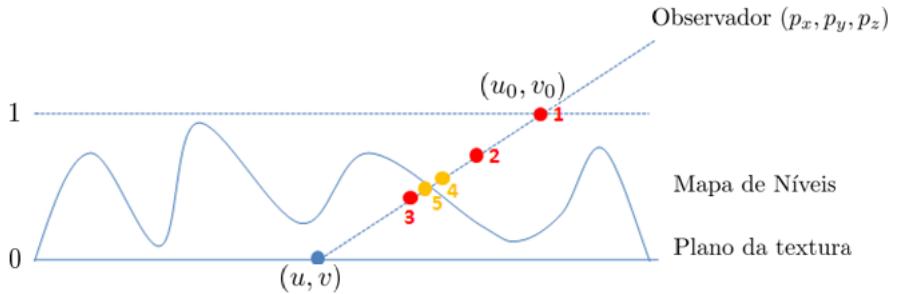


Figura 3.37: Realização da Busca Binária na segunda etapa.

Com o relevo do Mapa de Níveis entre os dois últimos pontos da Busca Linear, o *Relief Mapping* realiza na segunda etapa uma Busca Binária entre os pontos 2 e 3 convergindo rapidamente para a interseção entre o raio de visão na direção (p_x, p_y, p_z) e o relevo do Mapa de Níveis. Isto é mostrado na Figura 3.37, onde a Busca Binária realiza duas iterações, na primeira o ponto 4 é obtido e na segunda o ponto 5, o qual já está aproximadamente sobre a interseção. Aumentando-se as iterações da Busca Binária melhores aproximações ao ponto de interseção são obtidas. Neste caso vale ressaltar que a Busca Binária somente é utilizada na segunda etapa porque evitam-se os casos de não convergência tratados na seção 3.3.3 e aproveita-se sua grande velocidade de aproximação.

A Figura 3.38 mostra os bons níveis de detalhamento da meso-estrutura obtidos pelo *Relief Mapping* em ângulos grandes entre o plano da textura e o raio de visão.

Os testes realizados mostram que na maior parte das vezes uma implementação com 10 passos na Busca Linear e 4 na Busca Binária são suficientes para encontrar o ponto de interseção com boa precisão. Contudo, o método nem sempre converge para o ponto de

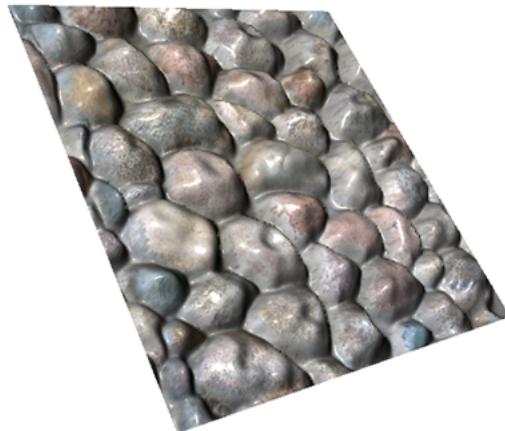


Figura 3.38: Relief Mapping sob ângulos grandes entre o plano da textura e o raio de visão.

interseção mais próximo do observador. Como a primeira etapa utiliza a Busca Linear a esta interseção pode ser perdida por causa do tamanho do passo e tipo de Mapa de Níveis, como já tratado na seção 3.3.2. Isto faz com que ocorram o aparecimento de distorções e "pixels nadando" quando o observador visualiza a superfície sobre pequenos ângulos, como mostra Figura 3.39.



Figura 3.39: As distorções e *pixels* nadando do *Relief Mapping* sob ângulos pequenos entre o raio de visão e o plano da textura.

A Figura 3.40 mostra em (a) o *Iterative Parallax Mapping*, o método não iterativo que obteve os melhores níveis de qualidade gráfico, e em (b) o *Relief Mapping* sendo aplicados para a obtenção dos detalhes da meso-estrutura de uma superfície de tijolos. Fica evidente a melhora da percepção de profundidade obtida em (b) em relação a (a). Isto ocorre porque *Relief Mapping* simula o efeito de oclusão, bem como apresenta as melhores aproximações do ponto de interseção para o problema da seção 3.1.4, contudo o *Relief Mapping* necessita de um processamento muito maior.

O auto-sombreamento pode ser conseguido da mesma maneira como discutido na seção 3.3.2, só que neste caso utiliza-se o *Relief Mapping* para realizar a verificação se o ponto de interseção está iluminado ou não.

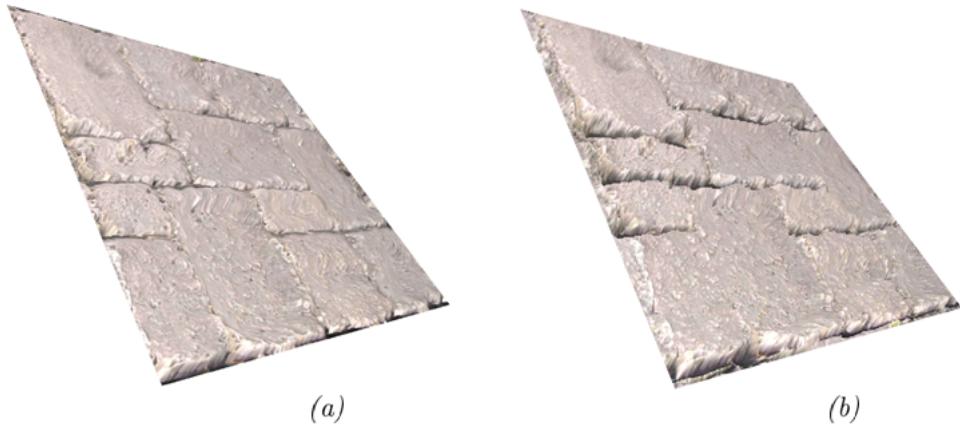


Figura 3.40: Um comparativo entre o (a) *Relief Mapping* e o (b) *Iterative Parallax Mapping*.

O *Relief Mapping* é uma técnica que utiliza o melhor da Busca Linear e Busca Binária para criar o detalhamento da meso-estrutura. Ele simula os efeitos de *motion parallax*, auto-oclusão e auto-sombreamento, o que oferece bons níveis de qualidade gráfico, contudo, exibe distorções em alguns casos, além de necessitar de grande processamento. A técnica não simula silhuetas, contudo, um aprimoramento realizado em *Relief Mapping of Non-Height-Field Surface Details* [8] tornou isto possível. Ele propôs a utilização de funções cônicas para a representação aproximada do relevo das superfícies em cada um dos vértices, permitindo a identificação e simulação silhuetas.

3.3.5 Parallax Occlusion Mapping

O *Parallax Occlusion Mapping* [26, 7, 16] é uma técnica iterativa de detalhamento da meso-estrutura que utiliza o conceito de *Ray Casting* para simulação dos efeitos de *motion parallax*, auto-oclusão e auto-sombreamento. Ele aborda o problema da interseção entre o raio de visão e o relevo do Mapa de Níveis da seção 3.1.4 em duas etapas. Na primeira executa uma Busca Linear que delimita um pequeno intervalo fechado de busca, semelhante ao utilizado pelo *Relief Mapping* e na segunda etapa realiza uma interseção aproximada entre a reta formada pelos dois últimos pontos obtidos na Busca Linear com o relevo descrito pelo Mapa de Níveis utilizando o método numérico da Falsa Posição [21]. Outra técnica que realiza exatamente o mesmo procedimento para gerar os detalhes da meso-estrutura é o *Interval Mapping* [21], contudo o *Parallax Occlusion Mapping* atingiu

maior popularidade por estar vinculada a área de jogos e definir algumas observações não presentes no trabalho do *Interval Mapping*.

A Figura 3.41 mostra os passos realizados pela Busca Linear da primeira etapa do *Parallax Occlusion Mapping*. São obtidos sucessivamente os pontos 1 e 2 na direção (p_x, p_y, p_z) do observador. Quando o ponto 2 é atingido a primeira etapa é finalizada, já que este se encontra abaixo do relevo do Mapa de Níveis.

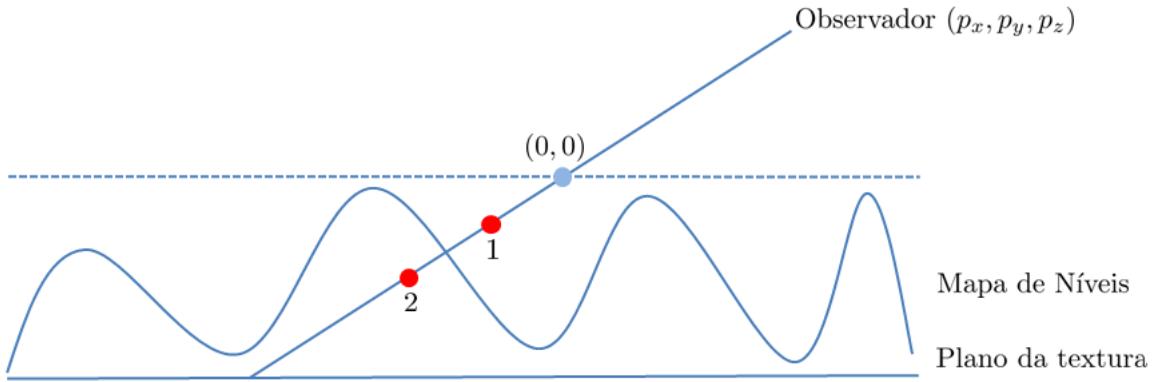


Figura 3.41: A realização da Busca Linear na primeira etapa.

Com o relevo do Mapa de Níveis entre os dois últimos pontos da Busca Linear, o *Parallax Occlusion Mapping* obtém a interseção aproximada entre o raio de visão e o Mapa de Níveis através do método da Falsa Posição. A partir dos pontos 1 e 2 da Figura 3.41 pode-se obter respectivamente os pontos $L_{inf} = (x_{inf}, y_{inf})$ e $L_{sup} = (x_{sup}, y_{sup})$ sobre o relevo do Mapa de Níveis como mostra a Figura 3.42.

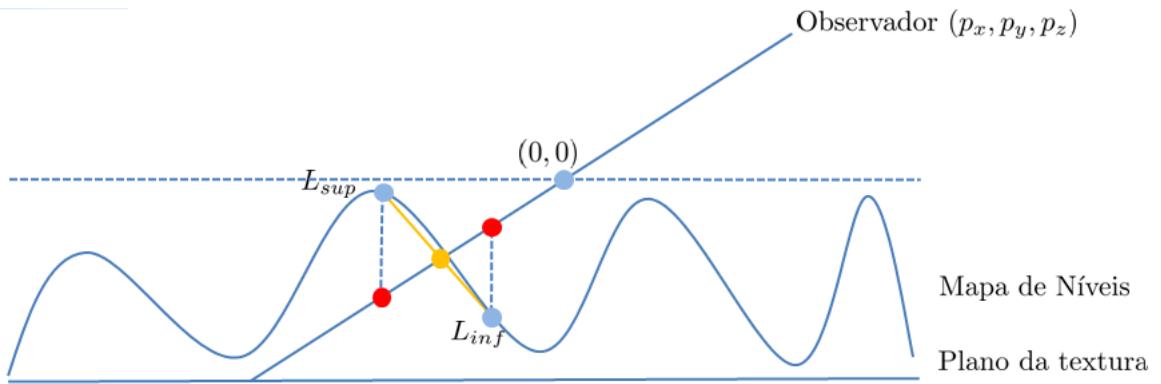


Figura 3.42: A obtenção da interseção aproximada através do método da Falsa Posição.

A interseção aproximada entre o raio de visão e o relevo do Mapa de Níveis pode ser obtido pelo método da Falsa Posição encontrando-se a interseção entre o raio de visão e a linha formada pelos pontos L_{inf} e L_{sup} como mostrado na Figura 3.42. Para

facilitar a compreensão do método definimos neste caso que o eixo horizontal é o x e o vertical o y com o raio de visão passa pela origem. Com isso o raio de visão pode ser definido pela Eq. (3.18) e a reta formada pelos pontos L_{inf} e L_{sup} pela Eq. (3.19).

$$A_vx + B_vy = 0 \quad (3.18)$$

$$A_lx + B_ly + C_l = 0 \quad (3.19)$$

Onde:

$$A_l = y_{sup} - y_{inf} \quad (3.20)$$

$$B_l = x_{inf} - x_{sup} \quad (3.21)$$

$$C_l = -B_ly_{sup} - A_lx_{sup} \quad (3.22)$$

Com isso pode-se resolver a interseção entre o raio de visão e a reta formada pelos pontos L_{inf} e L_{sup} pelas Eqs. (3.23) e (3.24).

$$x = C_l \frac{B_v}{A_vB_l - A_lB_v} \quad (3.23)$$

$$y = -C_l \frac{A_v}{A_vB_l - A_lB_v} \quad (3.24)$$

A interseção poderia ser obtida simplesmente definindo o raio de visão como a linha horizontal, sendo assim, $A_v = 0$ e $B_v = 1$, logo pode-se obter a interseção final aproximada através da Eq. (3.25).

$$x = C_l \frac{B_v}{A_vB_l - A_lB_v} = \frac{-C_l}{A_l} \quad (3.25)$$

O ponto no qual a reta intercepta o raio de visão irá definir um novo L_{inf} ou L_{sup} dependendo de sua localização em relação ao relevo do Mapa de Níveis. Caso ele esteja abaixo do relevo irá gerar um novo ponto L_{inf} , se estiver acima do relevo um ponto L_{sup} . Com isso pode-se realizar o procedimento descrito previamente obtendo aproximações cada vez melhores da interseção entre o raio de visão e o relevo do Mapa de Níveis.

A idéia da técnica em utilizar o método da Falsa Posição na segunda etapa do algoritmo ao invés da Busca Binária como no *Relief Mapping* é sua rápida convergência ao

ponto de interseção. Mesmo realizando uma divisão, uma multiplicação e três subtrações adicionais em relação ao *Relief Mapping* o *Parallax Occlusion Mapping* recupera este custo adicional convergindo à interseção com um número menor de iterações.

O *Parallax Occlusion Mapping* ainda define uma estimativa para a distância dos passos realizados pela marcha da Busca Linear. Como abordado na seção 3.3.2 a Busca Linear necessita de um número maior de passos quando o observador visualiza a superfície a pequenos ângulos para geração de imagens com boa qualidade. Sendo assim, propoem-se a definição de um valor máximo de mínimo de iterações possíveis para a Busca Linear. O número de iterações realizadas para cada tonalização de um *texel* em (u, v) será em função de uma interpolação linear entre $\vec{N}(u, v) \cdot \vec{p}$, onde \vec{p} é o vetor visão. Com isso a técnica realiza uma marcha a passos menores nas regiões com pequenos ângulos entre o raio de visão e o plano da superfície e uma marcha menos precisa em ângulos grandes.

A Figura 3.43 mostra o *Parallax Occlusion Mapping* sendo aplicado para detalhar uma superfície de pedras em ângulos médios entre o raio de visão e o plano da textura.



Figura 3.43: O *Parallax Occlusion Mapping* detalhando uma superfície de pedras utilizando um número mínimo de 5 e máximo de 10 iterações na primeira etapa e 2 iterações na segunda etapa.

3.3.6 Cone Step Mapping

O *Cone Step Mapping* é uma técnica de detalhamento da meso-estruturas que otimiza o método de Busca Linear utilizando texturas pré-calculadas com informações sobre os espaços vazios do relevo. Ele gera os efeitos de *motion parallax*, auto-occlusão e auto-sombreamento, permitindo a obtenção de níveis de qualidade de imagem altos.

Quando se deseja gerar o detalhamento da meso-estrutura com alta qualidade gráfica utilizando as técnicas baseadas em *Ray Casting* torna-se necessário um grande número de iterações. Contudo, mesmo com um elevado número de iterações não é possível garantir que o traçamento de raios realizado pela Busca Linear encontre a interseção entre o raio de visão e o relevo do Mapa de Níveis que esteja mais próximo do observador. Como tratado na seção 3.3.2 estes casos ocorrem geralmente quando o observador visualiza a superfície sob ângulos pequenos entre o raio de visão e o plano da textura ou quando são utilizados Mapas de Níveis que possuem alta frequência. Para solucionar este problema o *Cone Step Mapping* propõe que seja pré-calculada uma textura a partir do Mapa de Níveis que informe para cada um dos *texels* o maior raio de cone invertido sem que este penetre no interior do relevo do Mapa de Níveis, como mostra a Figura 3.44. Isto permite que a marcha da Busca Linear possa realizar passos a distâncias iguais àquela dada pela interseção entre o raio de visão e o cone, sem que se perca a interseção entre o raio de visão e o relevo do Mapa de Níveis mais próxima do observador. Sendo assim, cada *texel* do Mapa de Níveis possui um cone invertido definindo a distância máxima que qualquer raio que esteja localizado acima dele possa percorrer sem penetrar no interior do relevo do Mapa de Níveis.

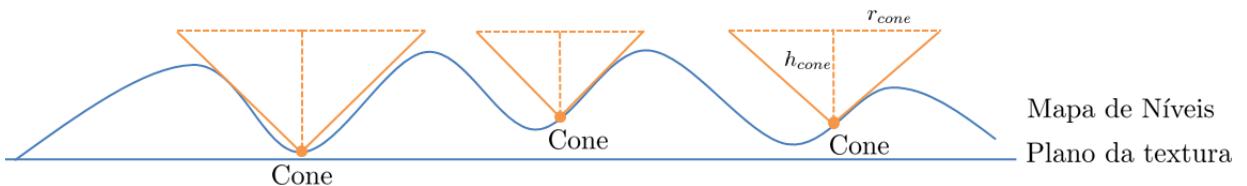


Figura 3.44: A amostragem de três cones invertidos de raio r_{cone} e altura h_{cone} mapeando os espaços livres do Mapa de Níveis.

A Figura 3.45 mostra um esquema simulando as iterações do *Cone Step Mapping* para solucionar o problema da interseção abordado na seção 3.1.4. O raio de visão na direção (p_x, p_y, p_z) parte das coordenadas de textura (u_0, v_0) onde o Mapa de Níveis atinge sua maior altura. O primeiro passo é obter o raio r_{cone} do cone invertido em (u_0, v_0)

e construi-lo sobre $H(u_0, v_0)$. Com isso pode-se realizar uma interseção entre o cone e o raio de visão obtendo a maior distância d_1 que se pode percorrer sem que o relevo do Mapa de Níveis seja alcançado. A distância d_1 permite que a marcha alcance as coordenadas de textura (u_1, v_1) , onde novamente um cone invertido é criado sobre $H(u_1, v_1)$ e sua interseção com o raio de visão permite encontrar a maior distância d_2 que se pode percorrer sem que o relevo do Mapa de Níveis seja alcançado. Neste caso a distância d_2 a partir de (u_1, v_1) na direção do raio de visão permite a obtenção da interseção entre o raio de visão e o relevo descrito pelo Mapa de Níveis.

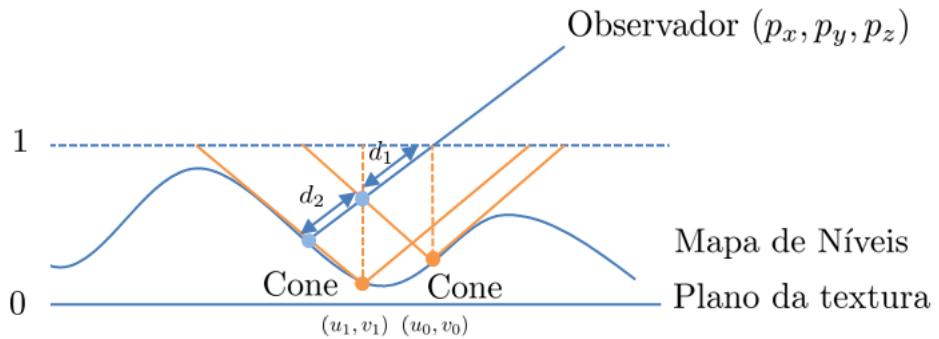


Figura 3.45: As iterações do *Cone Step Mapping* para encontrar a interseção entre o raio de visão na direção (p_x, p_y, p_z) e o relevo do Mapa de Níveis mais próximo do observador.

A interseção entre o raio de visão $\vec{p} = (p_x, p_y, p_z)$ e o cone invertido é dado pela Eq. (3.28), onde a altura do cone h_{cone_i} e as distâncias d_i de modulação da marcha são dadas respectivamente pelas Eqs. (3.26) e (3.27). Vale ressaltar que $F_i = (f_{x_i}, f_{y_i}, f_{z_i})$ é ponto dado no Espaço Tangente obtido a cada passo do algoritmo, sendo assim, f_{z_i} é altura do ponto em relação a superfície.

$$h_{cone_i} = H(u_i, v_i) - f_{z_i} \quad (3.26)$$

$$d_i = \frac{h_{cone_i} * r_{cone_i}}{r_{cone_i} + |\vec{p}_{xy}|} \quad (3.27)$$

$$F_{i+1} = F_i + \vec{p} * d_i \quad (3.28)$$

A Figura 3.46 mostra a boa qualidade visual obtida com utilização de apenas 2 iterações do *Cone Step Mapping*. A técnica permite convergência rápida ao ponto de interseção sem que o traço de raios perca a interseção mais próxima do observador.



Figura 3.46: A aplicação de duas iterações do *Cone Step Mapping* no detalhamento de uma superfície sob ângulos grandes entre o raio de visão e o plano da textura.

Mesmo com o mapeamento dos espaços vazios utilizando cones invertidos a técnica pode ainda gerar distorções, pois o número de iterações pode não ser suficiente para obter uma boa aproximação da interseção. Existe na verdade uma tendência de que os raios traçados parem antes de encontrar a interseção, o que pode gerar as distorções em destaque na Figura 3.47.

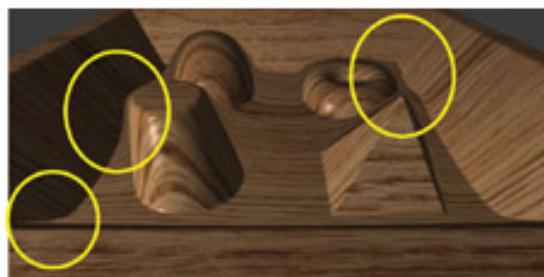


Figura 3.47: As distorções obtidas com o *Cone Step Mapping* quando o um número de iterações é inferior ao necessário para se chegar no ponto de interseção.

Uma técnica que corrige estas distorções foi abordada em *Relaxed Cone Step-*

ping for Relief Mapping [16] onde os cones invertidos podem penetrar uma única vez o relevo do Mapa de Níveis. Isto permite que os passos da marcha possam ser maiores e a convergência seja mais rápida, porém, exige uma segunda etapa de refinamento dada pela Busca Binária para encontrar a interseção. Como neste caso os cones invertidos penetram uma só vez o relevo, o algoritmo sabe exatamente onde deve iniciar a etapa de Busca Binária que converge em poucos passos ao ponto de interseção evitando as distorções da Figura 3.47.

Outra técnica que utiliza um conceito de otimização da Busca Linear é o *Per-Pixel Displacement Mapping with Distance Functions* [18]. Ela pré-calcula uma textura 3D [5], onde cada posição da textura armazenam o raio de uma esfera definido pela menor distância entre seu centro e o relevo do Mapa de Níveis. Sendo assim, pode-se realizar os passos da marcha garantindo que a primeira interseção nunca seja perdida. O inconveniente desta técnica é uso de texturas 3D que requerem um grande volume de memória.

4 Comparação das Técnicas de Detalhamento de Superfícies

Neste capítulo são comparadas as técnicas de Detalhamento de Superfície abordadas durante todo o trabalho. As comparações realizadas seguem duas vertentes, a teórica e a prática. A Comparação Teórica na seção 4.1, visa confrontar os tipos de efeitos que cada um dos métodos conseguem gerar, tais como, a simulação de silhuetas, *motion parallax*, auto-oclusão e auto-sombreamento. A Comparação Prática na seção 4.2, realiza um confrontamento entre as técnicas com base na qualidade visual e desempenhos obtidos na execução de cada um dos algoritmos. Tal comparação permite verificar a eficiência das técnicas sobre diferentes pontos de observação e Mapas de Níveis.

4.1 Comparação Teórica

As técnicas de Detalhamento de Superfícies abordadas ou referenciadas neste trabalho podem ou não gerar determinados efeitos no nível da meso-estrutura, permitindo a geração de imagens com diferentes qualidades. A Tabela 4.1 mostra a relação dos efeitos de silhuetas, *motion parallax*, auto-oclusão e auto-sombreamento que podem ser geradas por cada uma destas técnicas.

As técnicas que apresentaram os melhores desempenhos no comparativo da Tabela 4.1 são : *Displacement Mapping*, Busca Linear, *Relief Mapping*, *Parallax Occlusion Mapping*, *Cone Step Mapping*, *Displacement Mapping with Distance Functions*, *Relaxed Cone Stepping for Relief Mapping*, *Relief Mapping Of Non-Height-Field Surface Details* e a Tesselação. Contudo, o *Relief Mapping Of Non-Height-Field Surface Details* foi a única técnica referenciada neste trabalho a simular todos os efeitos de detalhamento da meso-estrutura.

Tabela 4.1: Suporte de efeitos nas técnicas de Detalhamento de Superfícies.

	Motion parallax	Auto-occlusão	Auto-sombreamento	Silhuetas
<i>Displacement Mapping (DM)</i>	Sim	Sim	Não	Sim
<i>Bump Mapping</i>	Não	Não	Não	Não
<i>Normal Mapping</i>	Não	Não	Não	Não
<i>Parallaxe Mapping (PM)</i>	Sim	Não	Não	Não
<i>PM with Offset Limmiting</i>	Sim	Não	Não	Não
<i>PM with Slope Information</i>	Sim	Não	Não	Não
<i>Iterative PM</i>	Sim	Não	Não	Não
<i>Busca Linear</i>	Sim	Sim	Sim	Não
<i>Busca Binária</i>	Sim	Sim	Não	Não
<i>Relief Mapping (RM)</i>	Sim	Sim	Sim	Não
<i>Parallax Occlusion Mapping</i>	Sim	Sim	Sim	Não
<i>Cone Step Mapping</i>	Sim	Sim	Sim	Não
<i>DM with Distance Functions</i>	Sim	Sim	Sim	Não
<i>Relaxed Cone Stepping for RM</i>	Sim	Sim	Sim	Não
<i>RM Of Non-Height-Field Surface Details</i>	Sim	Sim	Sim	Sim
<i>Tesselação</i>	Sim	Sim	Não	Sim

4.2 Comparação Prática

A Comparação Prática visa confrontar a qualidade das imagens geradas pelas técnicas de Detalhamento de Superfícies e o desempenho requerido, mesurado em número de imagens geradas por segundo. Como cada uma das técnicas possuem suas especificidades quanto ao número de iterações ou manipulação de parâmetros que permitem a obtenção de melhores qualidades visuais, cada um dos algoritmos foi executado sob configurações que permitiam a obtenção de seus melhores níveis gráficos. Tal decisão evita que uma determinada técnica obtenha qualidades visuais superiores ao realizar um determinado número de iterações ou possuir uma manipulação de parâmetros que seja favorável em uma dada circunstância. Sendo assim, a qualidade visual das imagens é comparada baseada em duas características:

- A presença de artefatos ou distorções na imagem.
- A qualidade do nível de detalhamento da meso-estrutura.

A comparação foi realizada utilizando o software *Render Monkey* que permite a construção e visualização de efeitos criados com *Shaders*. Nele foram aplicados os métodos de Detalhamento de Superfícies:

- (a) *Normal Mapping*.
- (b) *Parallax Mapping*.
- (c) *Parallax Mapping with Offset Limiting*.

- (d) *Iterative Parallax Mapping*.
- (e) *Busca Linear*.
- (f) *Relief Mapping*.
- (g) *Parallax Occlusion Mapping*.
- (h) *Cone Step Mapping*.

Vale ressaltar que as técnicas de *Displacement Mapping*, *Bump Mapping*, *Parallax Mapping with Slope Information* e Busca Binária foram excluídos da comparação. O *Displacement Mapping* e *Bump Mapping* foram abordados neste trabalho apenas como métodos históricos. O *Parallax Mapping with Slope Information* não foi incluído pois nas comparações foi utilizado sua versão iterativa, o *Iterative Parallax Mapping*. A Busca Binária por sua vez não se mostrou eficiente quando aplicada diretamente, gerando artefatos e distorções na maioria dos pontos de observação, sendo assim, foi desconsiderada por apresentar baixo qualidade visual.

Os métodos comparados foram testados sobre três tipos de meso-estrutura em uma superfície com 4 vértices, como mostrada na Figura 4.1. Para cada uma das meso-estruturas foram capturadas imagens em ângulos grandes e médios entre o raio de visão e a superfície. Nos Testes 1, 3 e 5 a captura de imagens foi realizada sobre ângulos grandes entre o raio de visão e a superfície. Nos Testes 2, 4 e 6 foram realizados sobre ângulos médios entre o raio de visão e a superfície. Para que houvesse uma igualdade de condições, cada uma das técnicas foi executado em mesmo computador, com processador de 2.4 GHz, 2 GB de memória RAM e placa de vídeo GeForce 275 GTX com 898 MB de memória

Os Testes 1, 2 e 8 utilizaram uma textura difusa com dimensão de 512x512 *texels* e Mapas de Níveis e de Normais com 256x256 *texels*. O Teste 1 mostrado na Figura 4.2 têm o ponto de observação posicionada sobre ângulos grandes com a superfície. Com isso os métodos comparados foram calibrados de maneira a apresentar o menor número de artefatos ou distorções, exibindo as melhores qualidades de detalhamento da meso-estrutura possíveis. A Busca Linear foi configurada para executar até 80 passos, o *Iterative Parallax Mapping* foi aplicado com 4 iterações, o *Relief Mapping* foi manipulado para executar até 40 passos de Busca Linear e 10 de Busca Binária, o *Parallax Occlusion Mapping* executou

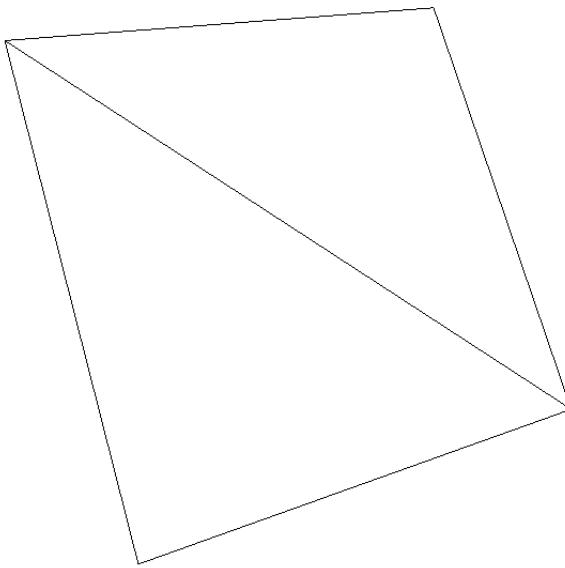


Figura 4.1: A superfície utilizada em todos Testes de comparação abordados neste capítulo.

um mínimo de 30 e máximo de 60 passos na Busca Linear e o *Cone Step Mapping* realizou 20 iterações. O Teste 2 mostrado na Figura 4.3 têm o ponto de observação posicionada sobre ângulos médios com a superfície. A Busca Linear foi configurada para executar até 90 passos, o *Iterative Parallax Mapping* foi aplicado com 4 iterações, o *Relief Mapping* foi manipulado para executar até 20 passos de Busca Linear e 6 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 30 e máximo de 60 passos na Busca Linear e o *Cone Step Mapping* realizou 20 iterações. O Teste 8 mostrado nas Figuras Figuras 4.12, 4.13, 4.14 e 4.15 têm o ponto de observação posicionada sobre ângulos pequenos com a superfície. O *Iterative Parallax Mapping* foi aplicado com 6 iterações, o *Relief Mapping* foi manipulado para executar até 40 passos de Busca Linear e 6 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 50 e máximo de 200 passos na Busca Linear e o *Cone Step Mapping* realizou 20 iterações.

Os Testes 3 e 4 utilizaram uma textura difusa com dimensão de 512x512 *texels* e Mapas de Níveis e de Normais com 256x256 *texels*. O Teste 3 mostrado na Figura 4.4 têm o ponto de observação posicionada sobre ângulos grandes com a superfície. A Busca Linear foi configurada para executar até 70 passos, o *Iterative Parallax Mapping* foi aplicado com 6 iterações, o *Relief Mapping* foi manipulado para executar até 30 passos de Busca Linear e 5 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 30 e máximo de 60 passos na Busca Linear e o *Cone Step Mapping* realizou 12 iterações. O Teste 4 mostrado na Figura 4.5 têm o ponto de observação posicionada sobre ângulos

Tabela 4.2: Taxa de geração de imagens nos Testes.

Métodos	Teste 1	Teste 2	Teste 3	Teste 4	Teste 5	Teste 6
<i>Normal Mapping</i>	2890	2893	2890	2850	2540	2550
<i>Parallax Mapping (PM)</i>	2888	2680	2800	2525	2510	2543
<i>PM with Offset Limmiting</i>	2884	2680	2804	2538	2500	2530
<i>Iterative PM</i>	2900	2600	2790	2522	2130	1163
<i>Busca Linear</i>	1220	700	1558	1031	812	477
<i>Relief Mapping</i>	2000	2238	2516	1850	1839	800
<i>Parallax Occlusion Mapping</i>	2600	2200	2306	1378	1700	1000
<i>Cone Step Mapping</i>	2700	2350	2580	2300	2517	1200

médios com a superfície. A Busca Linear foi configurada para executar até 70 passos, o *Iterative Parallax Mapping* foi aplicado com 6 iterações, o *Relief Mapping* foi manipulado para executar até 30 passos de Busca Linear e 5 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 30 e máximo de 80 passos na Busca Linear e o *Cone Step Mapping* realizou 12 iterações.

Os Testes 5, 6 e 7 utilizaram textura difusa, Mapa de Níves e Mapa de Normais com dimensão de 1024x1024 *texels*. O Teste 5 mostrado na Figura 4.6 têm o ponto de observação posicionada sobre ângulos grandes com a superfície. A Busca Linear foi configurada para executar até 60 passos, o *Iterative Parallax Mapping* foi aplicado com 6 iterações, o *Relief Mapping* foi manipulado para executar até 15 passos de Busca Linear e 6 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 30 e máximo de 40 passos na Busca Linear e o *Cone Step Mapping* realizou 10 iterações. O Teste 6 mostrado na Figura 4.7 têm o ponto de observação posicionada sobre ângulos médios com a superfície. A Busca Linear foi configurada para executar até 64 passos, o *Iterative Parallax Mapping* foi aplicado com 6 iterações, o *Relief Mapping* foi manipulado para executar até 26 passos de Busca Linear e 6 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 10 e máximo de 60 passos na Busca Linear e o *Cone Step Mapping* realizou 20 iterações. O Teste 7 mostrado nas Figuras 4.8, 4.9, 4.10 e 4.11 têm o ponto de observação posicionada sobre ângulos pequenos com a superfície. O *Iterative Parallax Mapping* foi aplicado com 6 iterações, o *Relief Mapping* foi manipulado para executar até 40 passos de Busca Linear e 6 de Busca Binária, o *Parallax Occlusion Mapping* executou um mínimo de 50 e máximo de 200 passos na Busca Linear e o *Cone Step Mapping* realizou 20 iterações.

A eficiência nos Testes 1, 2, 3, 4, 5 e 6 mensuradas em número de imagens geradas por segundo, podem ser vistas na Tabela 4.2.

O *Normal Mapping* obteve os piores níveis de detalhamento em todos os Testes. Isto ocorreu porque ele simula os detalhamentos na superfície somente utilizando o cálculo de iluminação. Como não faz qualquer deslocamento na textura ou traçamento de raios, ele não consegue simular qualquer efeito de detalhamento. Contudo, sua eficiência foi a melhor dentre todas as técnicas, apresentando a maior taxa de geração de imagens, como pode ser observado na Tabela 4.2.

O *Parallax Mapping* e o *Parallax Mapping with Offset Limiting* geraram imagens com qualidade e eficiência muito similares em todas as comparações. Nos Testes 2, 4 e 6 suas imagens se aproximaram bastante daquelas obtidas pelo *Normal Mapping*, por estarem sob pontos de observação onde não são evidenciados grandes números de sobreposições. Contudo, os Testes 1, 3 e 5 mostram o *Parallax Mapping* e o *Parallax Mapping with Offset Limiting* obtendo melhores qualidades de detalhamento que o *Normal Mapping*. Isto ocorreu porque ambas as técnicas simulam o efeito de *motion parallax*, que é mais bem evidenciado à medida que o observador visualiza a superfície sob ângulos menores. A eficiência de ambas as técnicas foi aproximada e muito alta, somente perdendo para aquelas obtidas pelo *Normal Mapping*, como pode ser observado na Tabela 4.2. Deve-se levar em conta como abordado na seção 3.2.2, que o *Parallax Mapping with Offset Limiting* elimina as distorções que frequentemente ocorrem sob ângulos pequenos entre o raio de visão e a superfície.

O *Iterative Parallax Mapping* gerou imagens com extrusão do relevo superior às técnicas de *Normal Mapping*, *Parallax Mapping* e *Parallax Mapping with Offset Limiting* em todos os Testes. Isto ocorreu porque ele leva em consideração o Mapa de Normais para simular o efeito de *motion parallax*. Sobre tudo, os Testes 2, 4 e 5 mostram que a técnica gera algumas distorções que podem ser evidenciadas nas Figuras 4.3, 4.5 e 4.6. Sua eficiência na geração de imagens foi inferior aos métodos previamente analisados, como mostra a Tabela 4.2.

A *Busca Linear*, o *Relief Mapping*, o *Parallax Occlusion Mapping* e o *Cone Step Mapping* geraram imagens com qualidade muito semelhantes e superiores às técnicas de *Normal Mapping*, *Parallax Mapping* e *Parallax Mapping with Offset Limiting* em todos os Testes. Isto ocorreu porque estas técnicas simulam além do efeito de *motion parallax* o efeito de auto-oclusão. Contudo, a requisição de um número elevado de iterações para a obtenção de suas melhores qualidades visuais fez com que estes métodos obtivessem as piores taxas de geração de imagens, como pode ser observado na Tabela 4.2. A *Busca*

Linear obteve dentre todos os métodos, a pior taxa de geração de imagens, pois sua marcha exige um número alto de iterações para o desaparecimento das distorções e artefatos. O *Parallax Occlusion Mapping* obteve geração de imagens superior a Busca Linear em todos os Testes. O *Relief Mapping* por sua vez obteve melhores taxas de geração de imagens que o *Parallax Occlusion Mapping* nos Testes 2, 3, 4 e 5 por ter exigido em média um número menor de iterações. O *Cone Step Mapping* obteve as melhores taxas de geração de imagens dentre as técnicas de *Busca Linear*, *Relief Mapping* e *Parallax Occlusion Mapping*. Isto ocorreu, pois a utilização de uma textura que mapeia os espaços vazios por meio de cones, permite que a interseção entre o raio de visão e o relevo do Mapa de Níveis seja encontrado rapidamente.

A visualização das imagens geradas em cada um dos Testes possibilitou observar que as técnicas de *Iterative Parallax Mapping*, *Busca Linear*, *Relief Mapping*, *Parallax Occlusion Mapping* e *Cone Step Mapping* apresentaram as melhores qualidades gráficas. As técnicas de *Normal Mapping*, *Parallax Mapping* e *Parallax Mapping with Offset Limiting* conseguem gerar alguns detalhes da meso-estrutura, mas o resultado obtido é muito inferior as demais técnicas. Em todos os Testes, as técnicas foram levadas ao limite quanto ao número de iterações, a fim de gerar imagens sob as melhores qualidades possíveis. Isto fez com que boa parte das distorções e artefatos evidenciados durante abordagem dos métodos desaparecessem, dificultando a seleção do melhor método. Sendo assim, foram selecionadas as técnicas que obtiveram os melhores resultados entre qualidade de imagem gerada e eficiência para a execução dos Testes 7 e 8, sob ângulos pequenos entre o raio de visão e a superfície. Tais técnicas são o *Iterative Parallax Mapping*, *Relief Mapping*, *Parallax Occlusion Mapping* e *Cone Step Mapping* sendo excluída a Busca Linear devido ao seu baixíssimo desempenho.

O Teste 7 mostrado nas Figuras 4.8, 4.9, 4.10 e 4.11, confronta respectivamente as técnicas de *Cone Step Mapping*, *Parallax Occlusion Mapping*, *Relief Mapping* e *Iterative Parallax Mapping* sobre ângulos pequenos entre o raio de visão e a superfície. Todos as técnicas apresentaram algum tipo de distorção e presença de artefatos, mas possibilitaram diferentes níveis de extrusão do relevo sem que a imagem gerada fosse totalmente distorcida. O *Iterative Parallax Mapping* na Figura 4.11 foi o método que permitiu a menor extrusão neste ponto de observação, gerando os maiores níveis de distorção na imagem. O *Relief Mapping* na Figura 4.10 obteve níveis de extrusão aproximados aos do *Iterative Parallax Mapping*, sem contudo gerar tantas distorções e artefatos. Por sua vez,

Tabela 4.3: Taxa de geração de imagens nos Testes.

Métodos	Teste 7	Teste 8
<i>Iterative Parallax Mapping</i>	2134	2305
<i>Relief Mapping</i>	775	688
<i>Parallax Occlusion Mapping</i>	355	507
<i>Cone Step Mapping</i>	1120	1400

os métodos de *Cone Step Mapping* e *Parallax Occlusion Mapping* permitiram extrusões bem semelhantes e maiores que as outras técnicas do Teste 7. O *Cone Step Mapping*, apresentou um menor nível de distorções em relação ao *Parallax Occlusion Mapping*.

O Teste 8 mostrado nas Figuras 4.12, 4.13, 4.14 e 4.15 confronta, respectivamente as técnicas *Cone Step Mapping*, *Parallax Occlusion Mapping*, *Relief Mapping* e *Iterative Parallax Mapping* sobre ângulos pequenos entre o raio de visão e a superfície. Todos as técnicas apresentaram pequenas distorções e possibilidade de extrusão do relevo semelhantes, excetuando-se o *Iterative Parallax Mapping* na Figura 4.15. Este obteve os piores níveis de qualidade dentre as técnicas, permitindo que pouca extrusão do relevo pudesse ser observada. Contudo, as demais técnicas mantiveram qualidades de imagens bem próximas.

A Tabela 4.2 mostra a taxa de geração de imagens obtidas para os Testes 7 e 8. Neste caso, percebe-se que o *Iterative Parallax Mapping* foi o mais eficiente, mantendo taxas de gerações altas. Mas suas qualidades visuais obtidas nestes Testes acabaram sendo as piores. O *Relief Mapping* e o *Parallax Occlusion Mapping* tiveram as menores taxas de geração de imagens. O *Relief Mapping* obteve bons resultados no Teste 8, mantendo qualidades gráficas semelhantes ao *Cone Step Mapping* e *Parallax Occlusion Mapping*, porém no Teste 7 não permitiu que fossem geradas grandes extrusões no relevo, sem que a imagem se distorcesse por completo. Ele apresentou ainda grande taxa de geração de imagens, como pode ser observado na Tabela 4.2.

A realização dos oito Testes e as taxas de geração de imagens presentes nas Tabelas 4.2 e 4.2, permitiu a seleção dos melhores métodos na Tabela 4.4. Para isso, levou-se em conta que a qualidade das imagens geradas possuem maior peso que a taxa de gerações de imagens. O *Cone Step Mapping* foi o melhor método implementado neste trabalho, obtendo bons níveis de qualidade gráfica em conjunto com altas taxas de geração

Tabela 4.4: *Ranking* das técnicas de Detalhamento de Superfícies.

1. <i>Cone Step Mapping</i>
2. <i>Parallax Occlusion Mapping e Relief Mapping</i>
3. <i>Iterative Parallax Mapping</i>
4. <i>Busca Linear</i>
5. <i>Parallax Mapping with Offset Limiting</i>
6. <i>Parallax Mapping</i>
7. <i>Normal Mapping</i>

de imagens. Em segundo lugar, os métodos de *Relief Mapping* e *Parallax Occlusion Mapping* empataram por produzirem imagens aproximadas, a taxas de geração de imagens antangônicos em alguns Testes, o que dificultou estimar qual é propriamente o melhor. Em seguida o *Iterative Parallax Mapping* obteve altas taxas de imagens, contudo, a qualidade gráfica não foi tão boa nos Testes os Testes 2, 4 ,5, 7 e 8. A Busca Linear, mesmo exibindo altos níveis de qualidade, obteve as piores taxas de geração de imagens entre todos os Testes, com isso foi posicionada no quarto lugar. O *Parallax Mapping with Offset Limiting* e o *Parallax Mapping* obtiveram níveis de qualidade gráficos bem semelhantes em todos os Testes, porém, o *Parallax Mapping with Offset Limiting* não apresenta distorções quando observados sobre pequenos ângulos entre o raio de visão e a superfície. O *Normal Mapping* obteve as piores qualidades de imagem, com geração de pequenas sensações de extrusão do relevo. Em contraposição, sua taxa de geração de imagens foi a maior dentre todos os Testes.

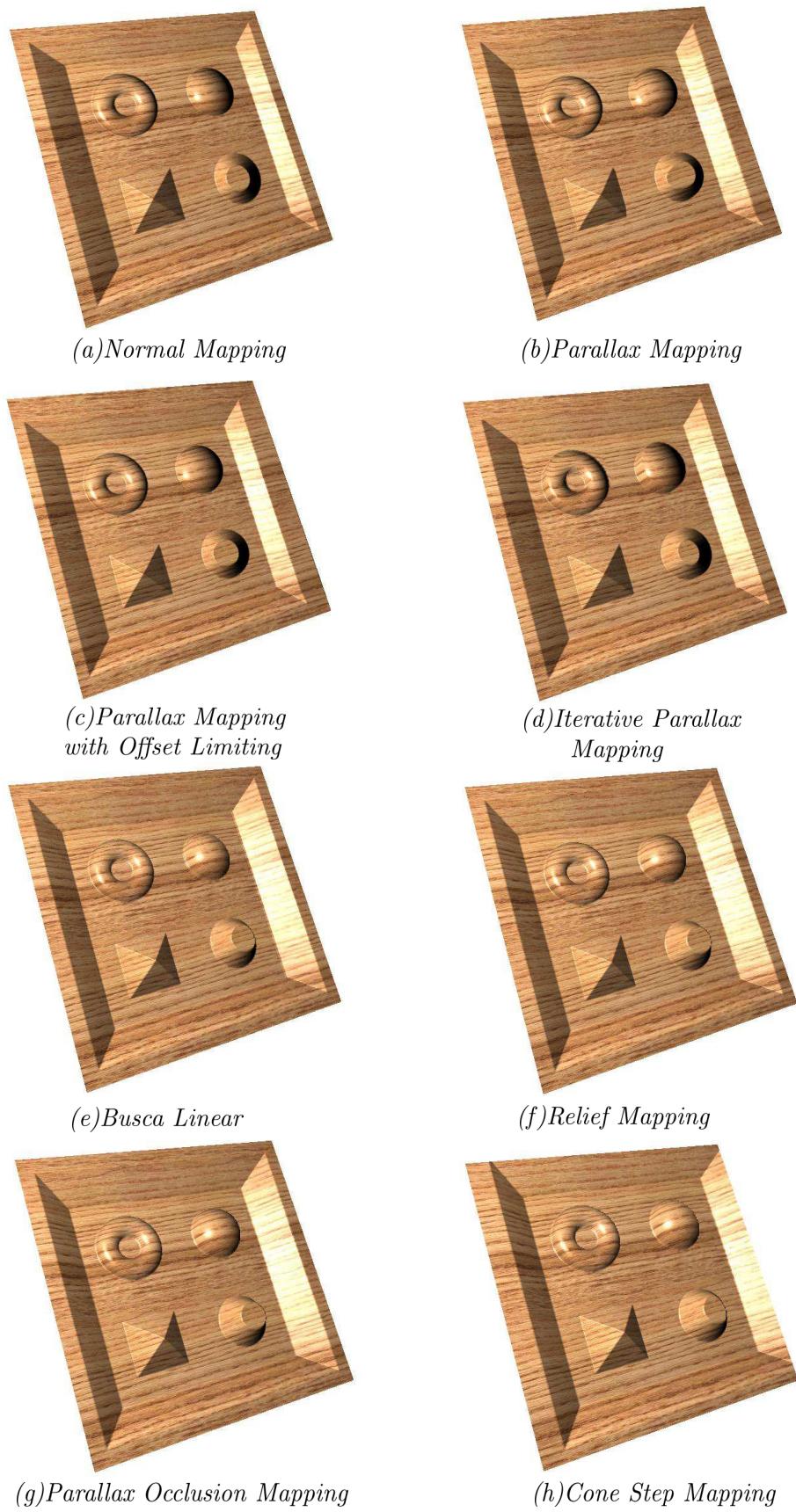


Figura 4.2: Teste 1, Comparação entre as técnicas de Detalhamento de Superfícies .
(a)Normal Mapping, (b)Parallax Mapping, (c)Parallax Mapping with Offset Limmiting,
(d)Iterative Parallax Mapping, (e)Busca Linear, (f)Relief Mapping, (g)Parallax Occlusion
Mapping e (h)Cone Step Mapping.

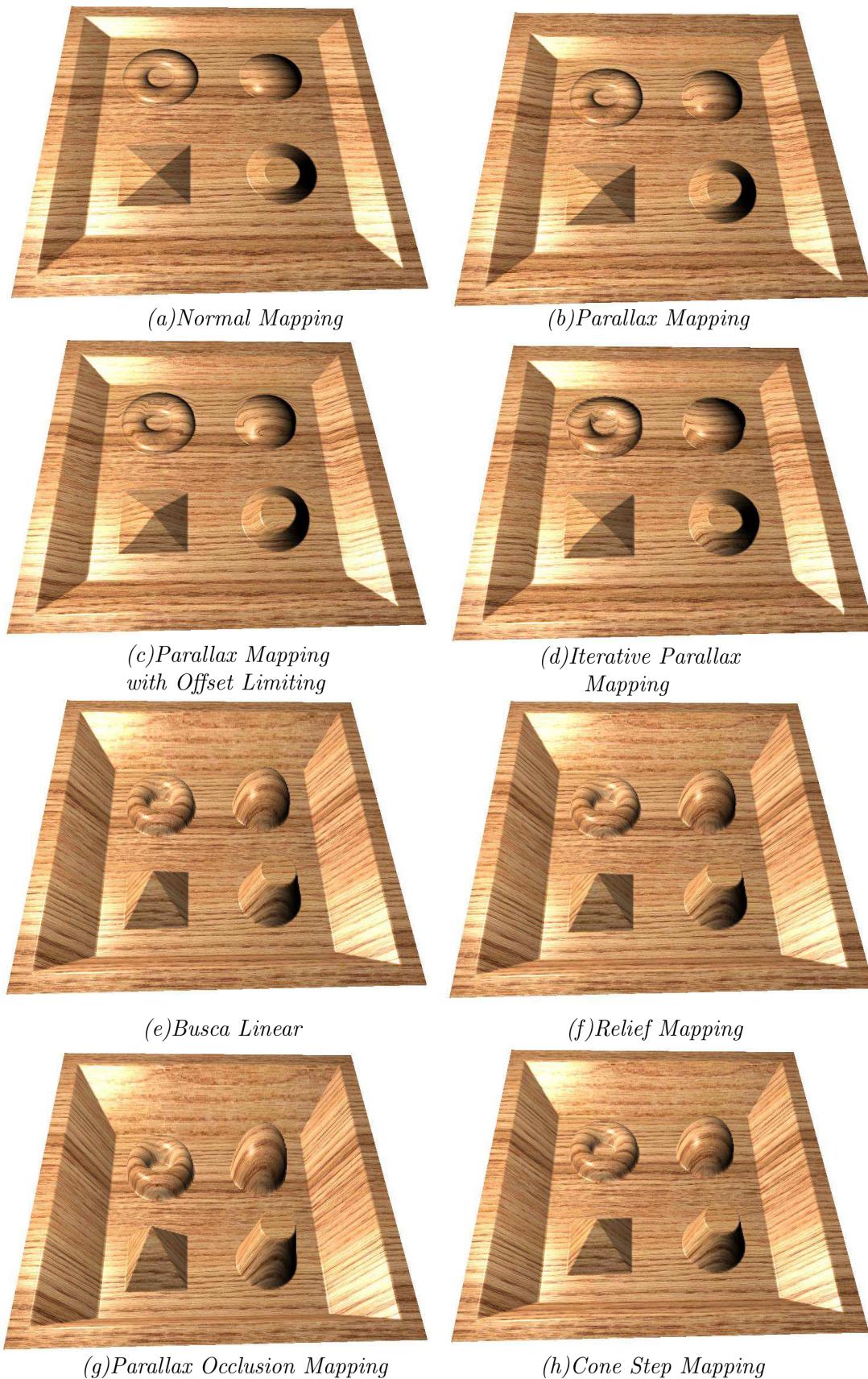


Figura 4.3: *Teste 2*, Comparação entre as técnicas de Detalhamento de Superfícies .
 (a) *Normal Mapping*, (b) *Parallax Mapping*, (c) *Parallax Mapping with Offset Limmiting*,
 (d) *Iterative Parallax Mapping*, (e) *Busca Linear*, (f) *Relief Mapping*, (g) *Parallax Occlusion
 Mapping* e (h) *Cone Step Mapping*.

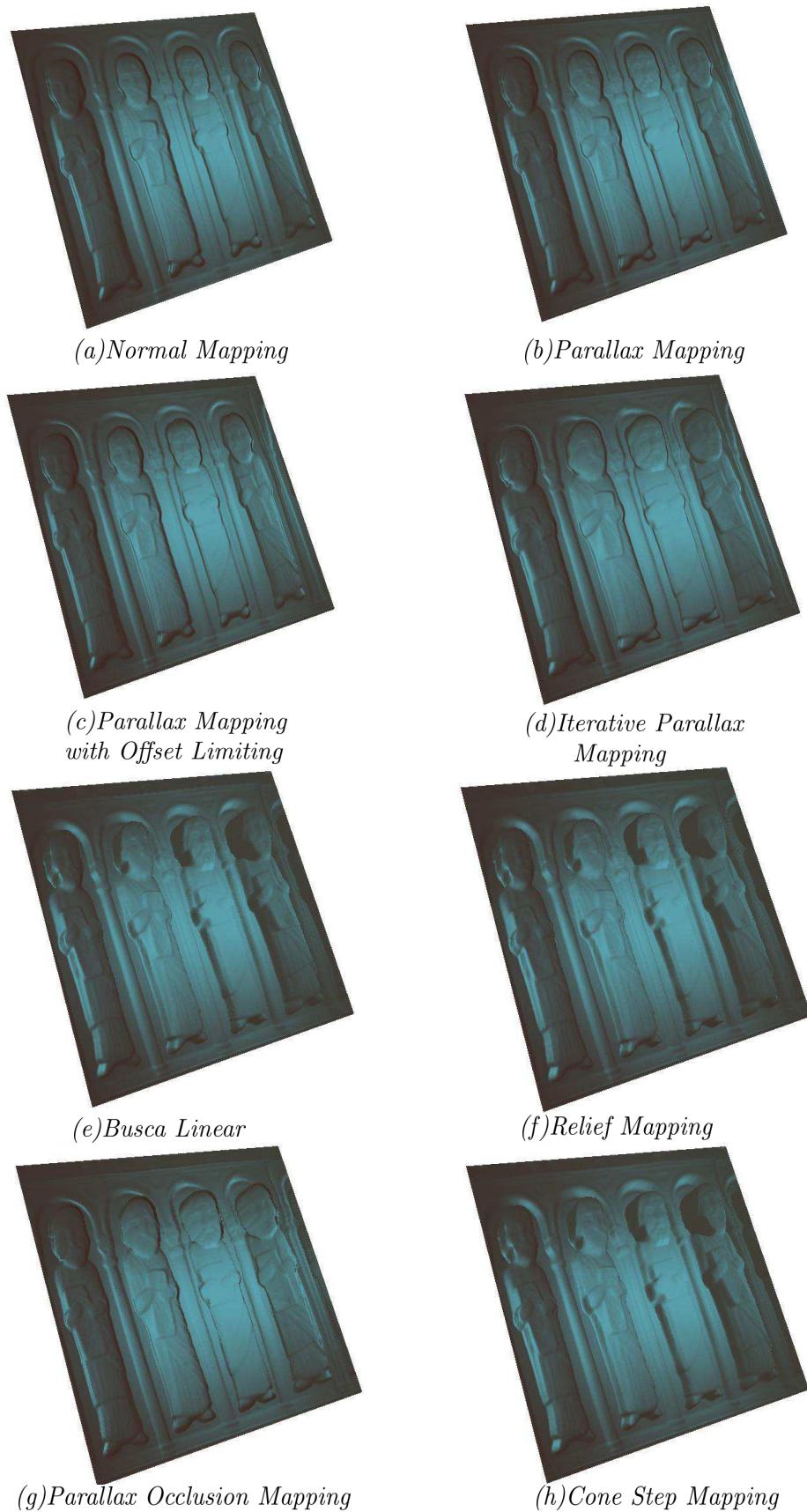


Figura 4.4: Teste 3, comparação entre as técnicas de Detalhamento de Superfícies. (a)Normal Mapping, (b)Parallax Mapping, (c)Parallax Mapping with Offset Limmiting, (d)Iterative Parallax Mapping, (e)Busca Linear, (f)Relief Mapping, (g)Parallax Occlusion Mapping e (h)Cone Step Mapping.

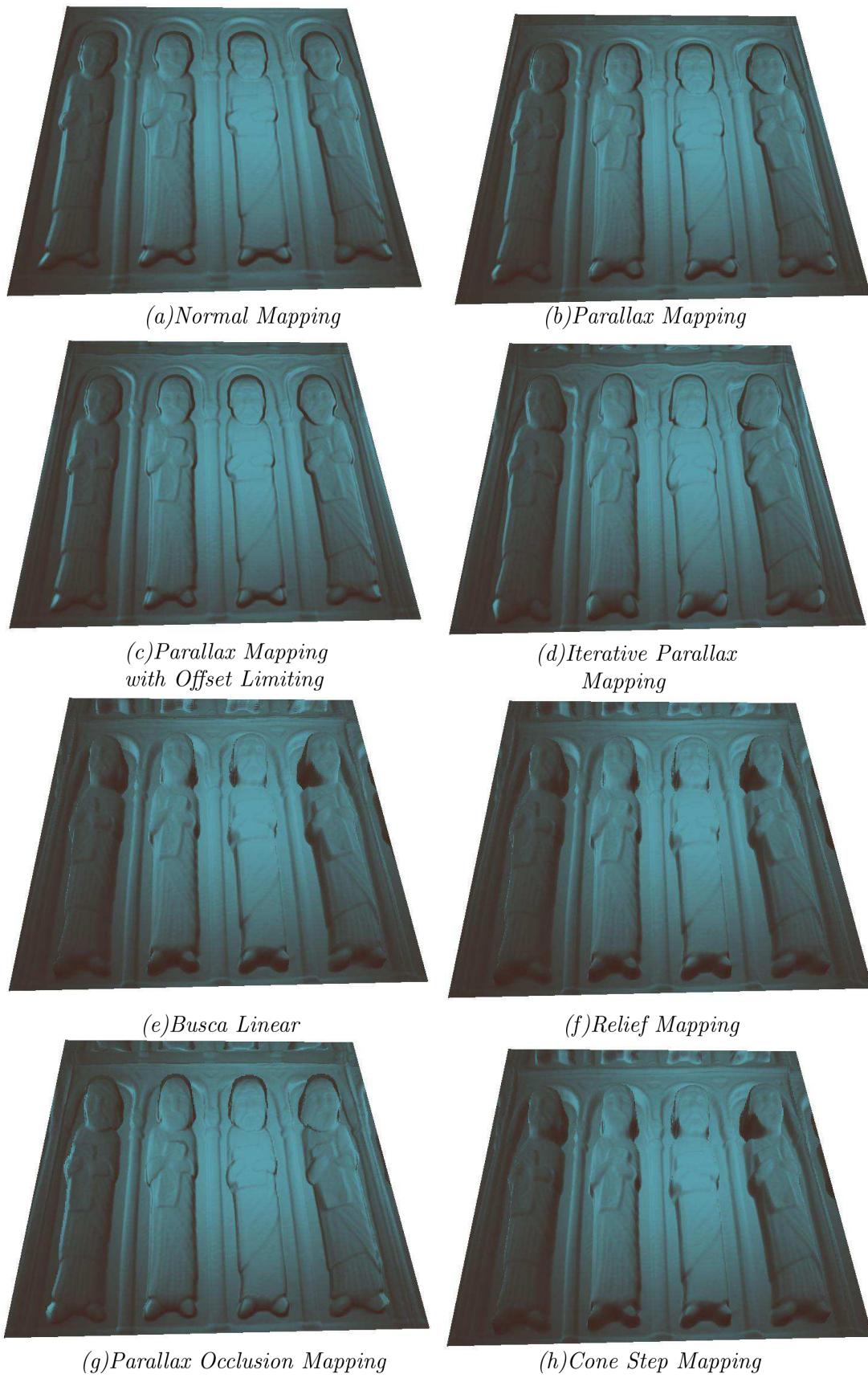


Figura 4.5: *Teste 4*: Comparação entre as técnicas de Detalhamento de Superfícies.
 (a)Normal Mapping, (b)Parallax Mapping, (c)Parallax Mapping with Offset Limmiting,
 (d)Iterative Parallax Mapping, (e)Busca Linear, (f)Relief Mapping, (g)Parallax Occlusion
 Mapping e (h)Cone Step Mapping.



Figura 4.6: Teste 5, comparação entre as técnicas de Detalhamento de Superfícies. (a)Normal Mapping, (b)Parallax Mapping, (c)Parallax Mapping with Offset Limmiting, (d)Iterative Parallax Mapping, (e)Busca Linear, (f)Relief Mapping, (g)Parallax Occlusion Mapping e (h)Cone Step Mapping.

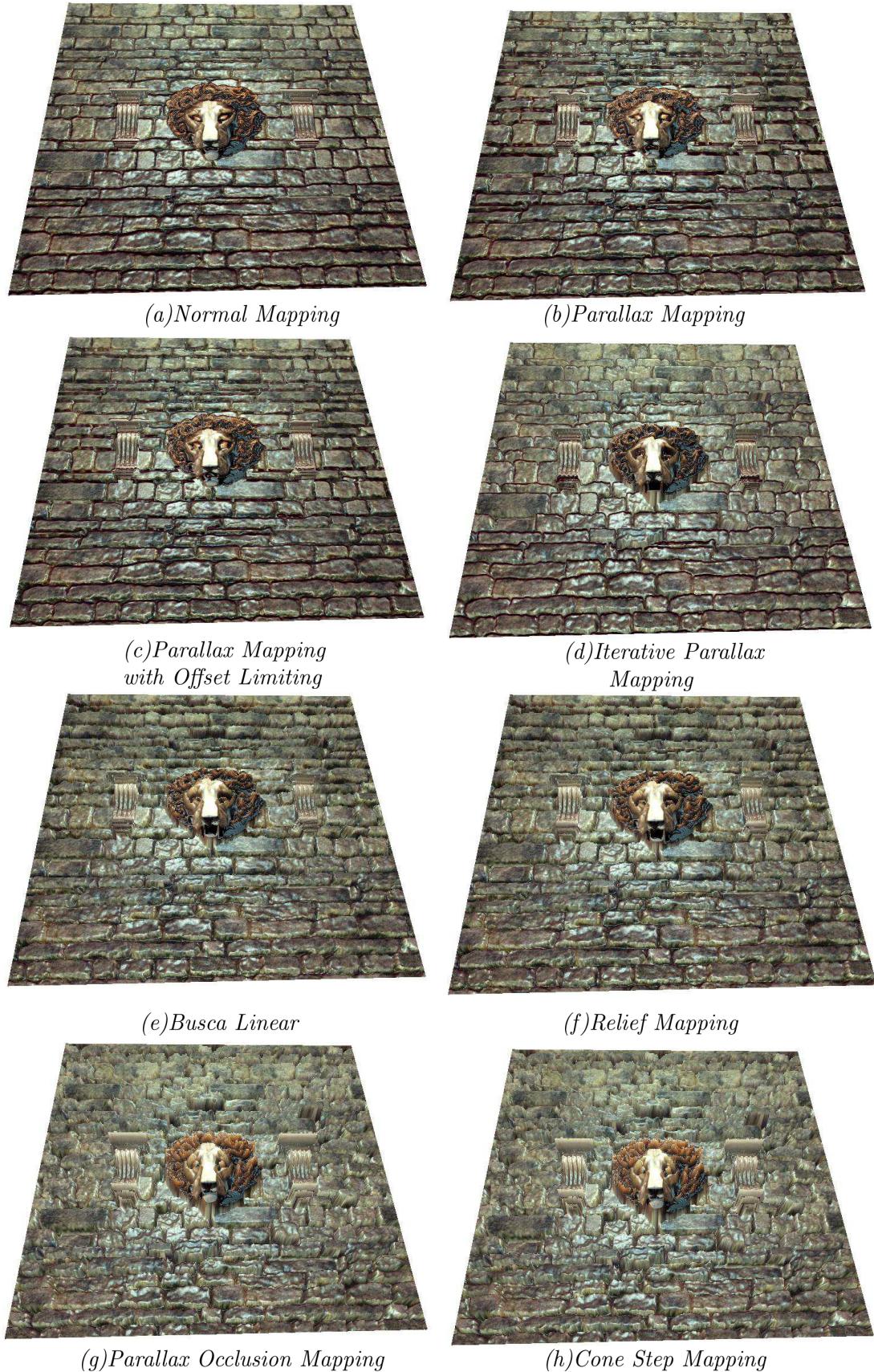


Figura 4.7: Teste 6, comparação entre as técnicas de Detalhamento de Superfícies.
(a)Normal Mapping, (b)Parallax Mapping, (c)ParallaxMapping with Offset Limmiting,
(d)Iterative Parallax Mapping, (e)Busca Linear, (f)Relief Mapping, (g)Parallax Occlusion
Mapping e (h)Cone Step Mapping.

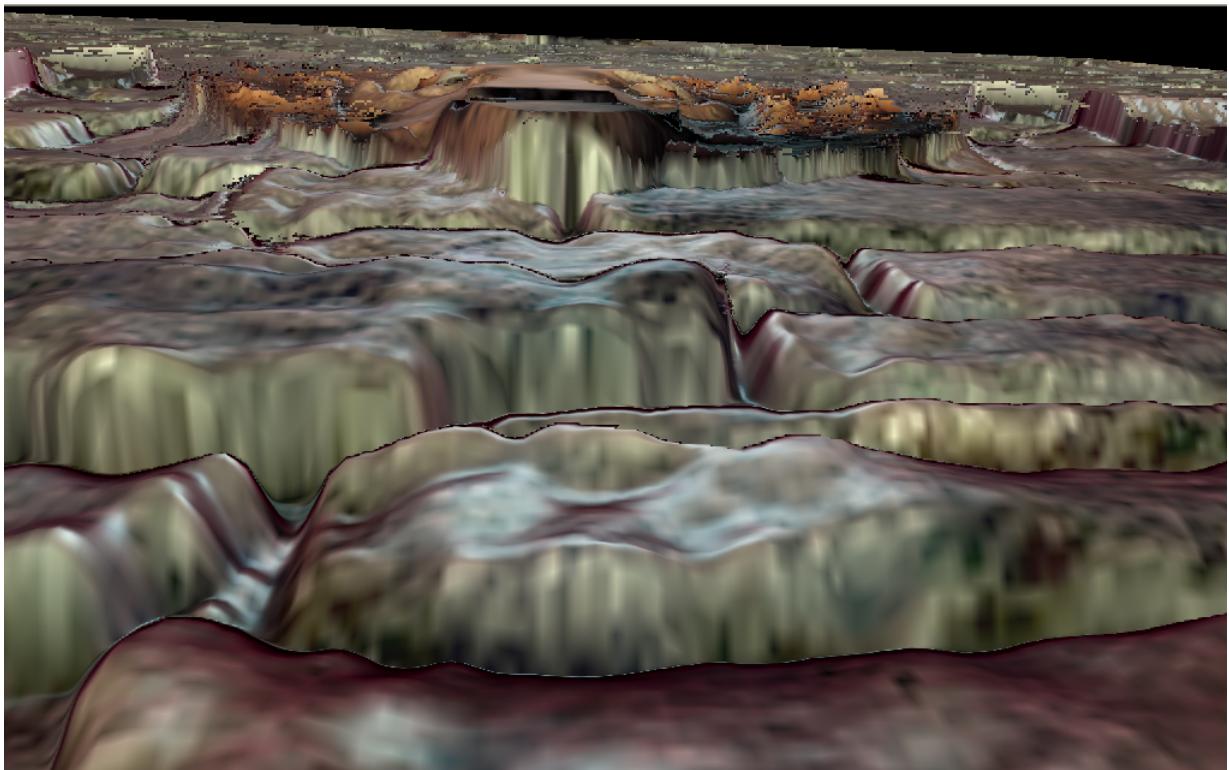


Figura 4.8: *Teste 7*, visualização do *Cone Step Mapping* a pequenos ângulos entre o raio de visão e a superfície.

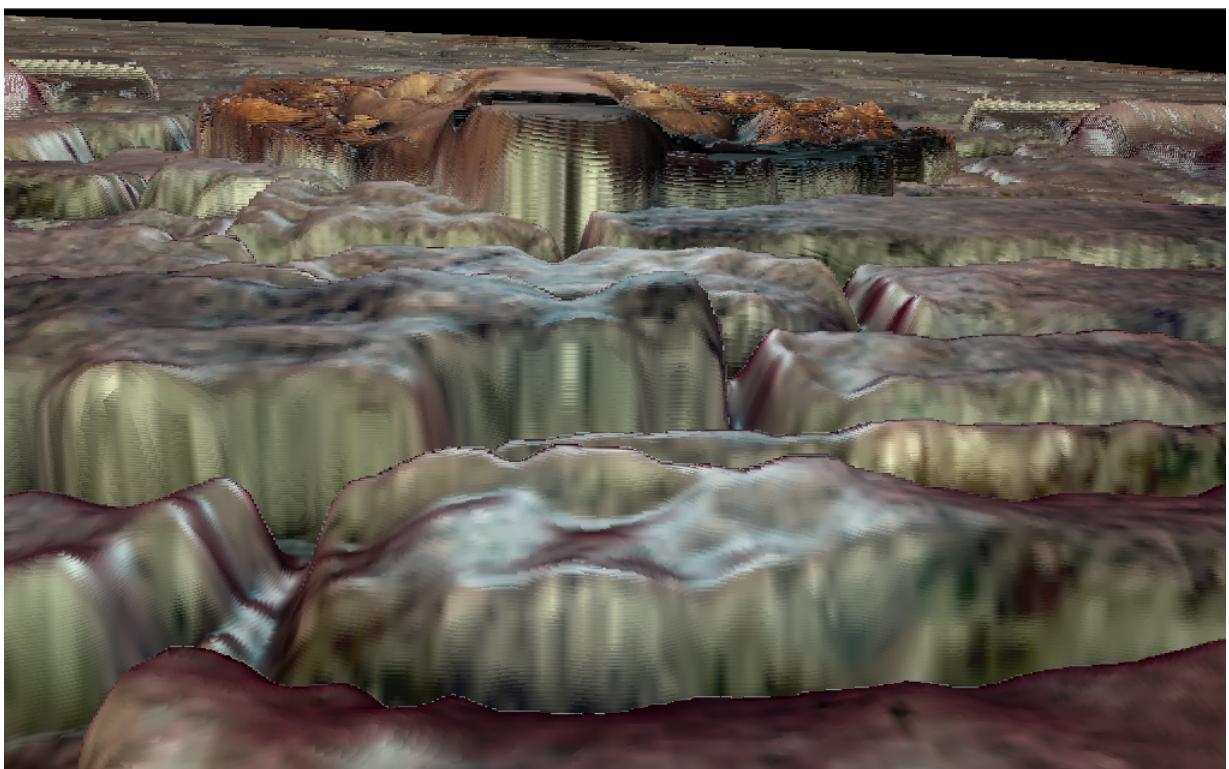


Figura 4.9: *Teste 7*, visualização do *Parallax Occlusion Mapping* a pequenos ângulos entre o raio de visão e a superfície.

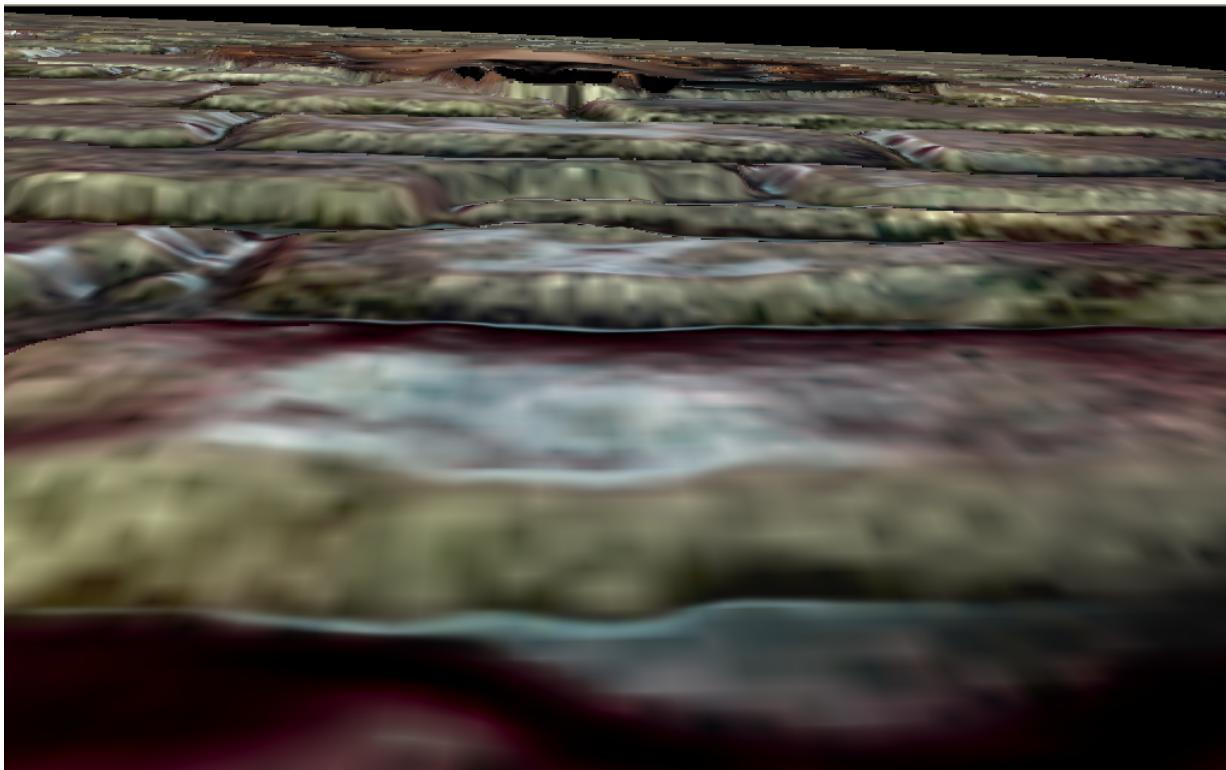


Figura 4.10: *Teste 7*, visualização do *Relief Mapping* a pequenos ângulos entre o raio de visão e a superfície.

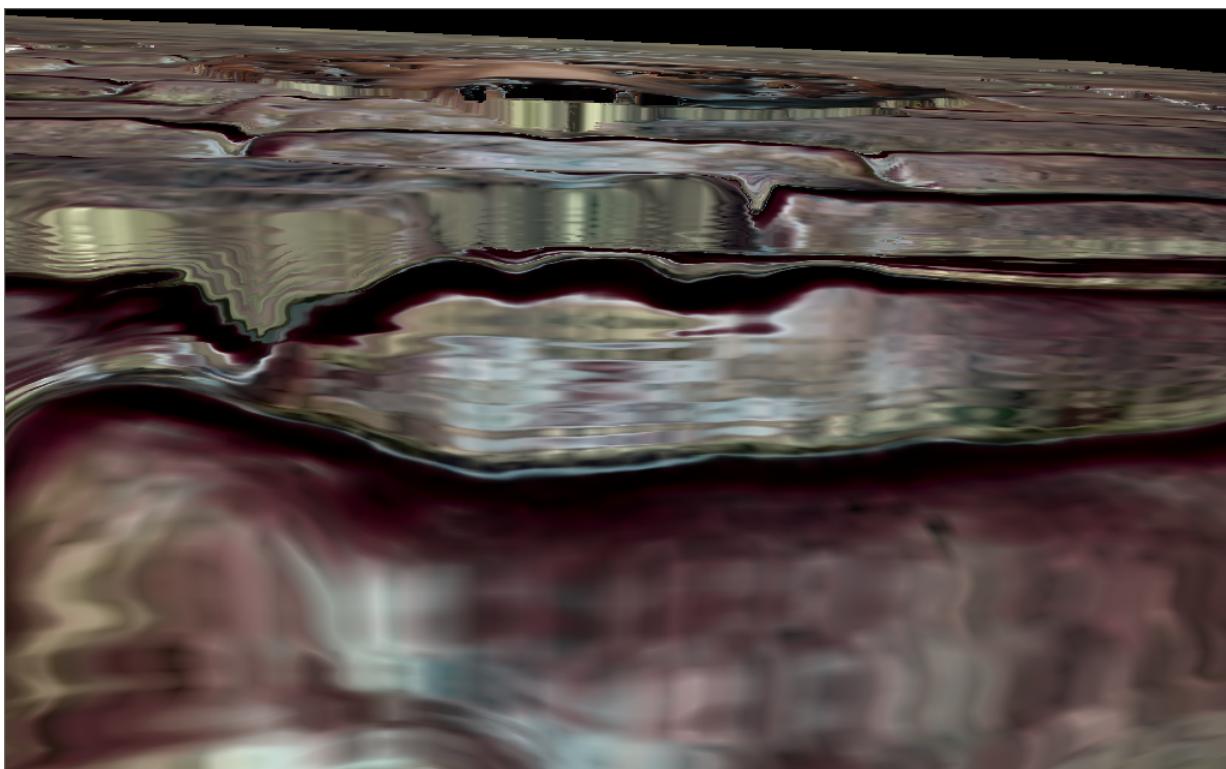


Figura 4.11: *Teste 7*, visualização do *Iterative Parallax Mapping* a pequenos ângulos entre o raio de visão e a superfície.

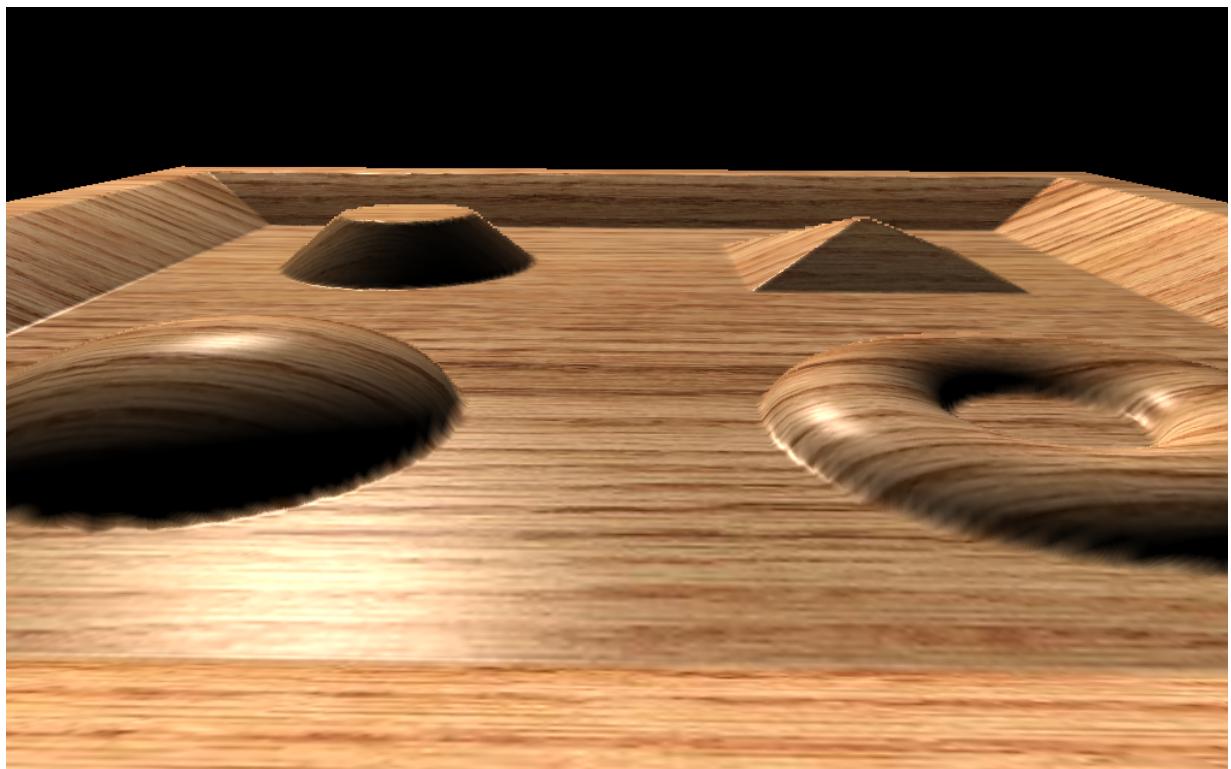


Figura 4.12: *Teste 8*, visualização do *Cone Step Mapping* a pequenos ângulos entre o raio de visão e a superfície.

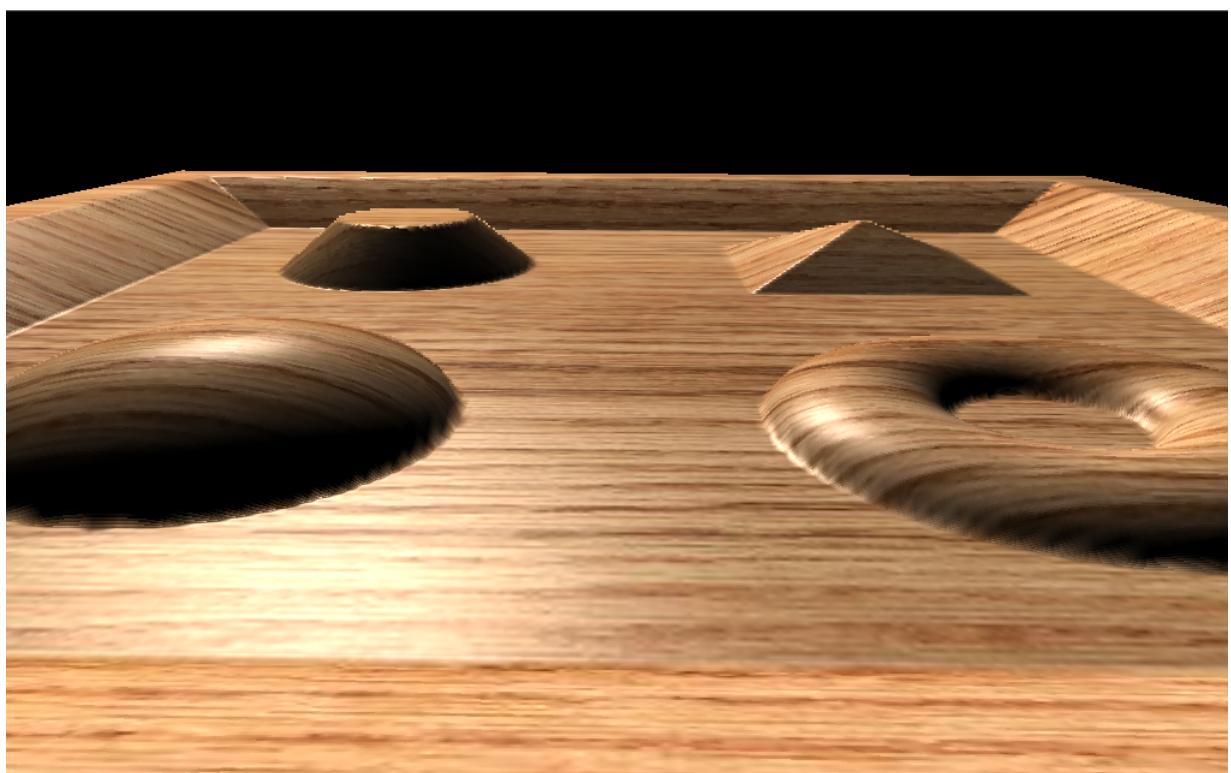


Figura 4.13: *Teste 8*, visualização do *Parallax Occlusion Mapping* a pequenos ângulos entre o raio de visão e a superfície.

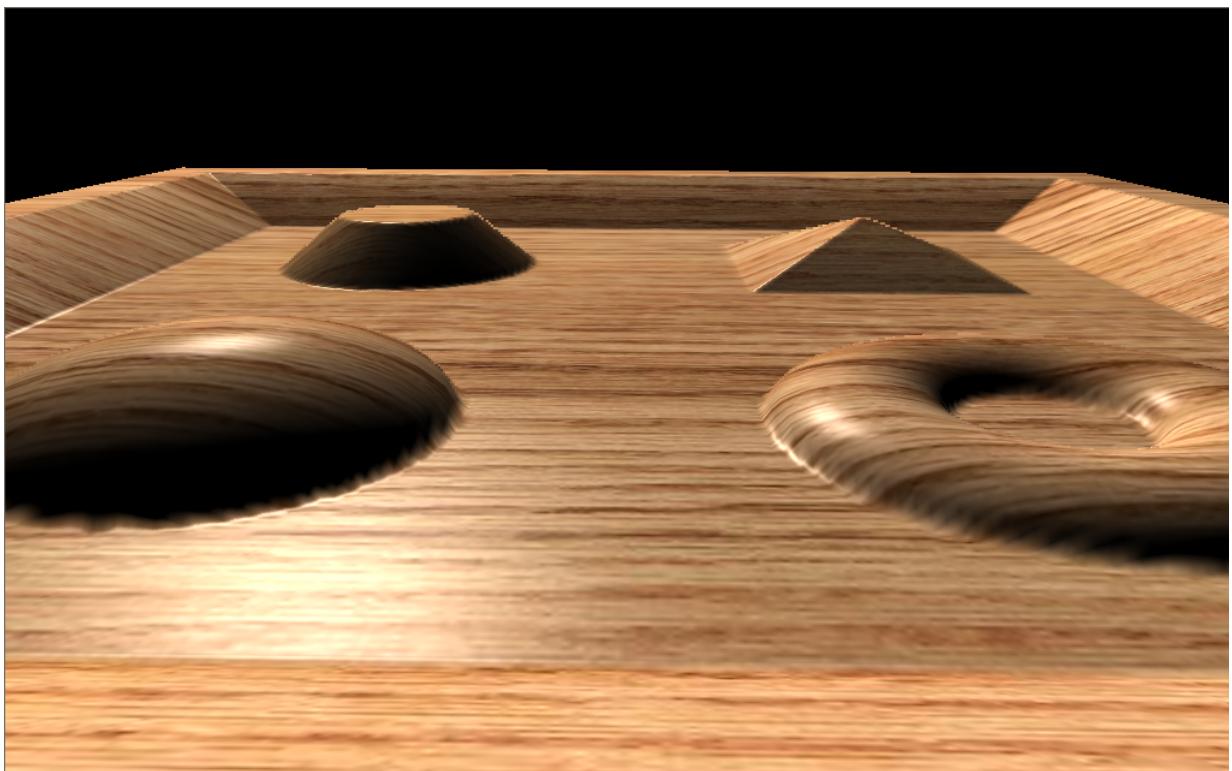


Figura 4.14: *Teste 8*, visualização do *Relief Mapping* a pequenos ângulos entre o raio de visão e a superfície.

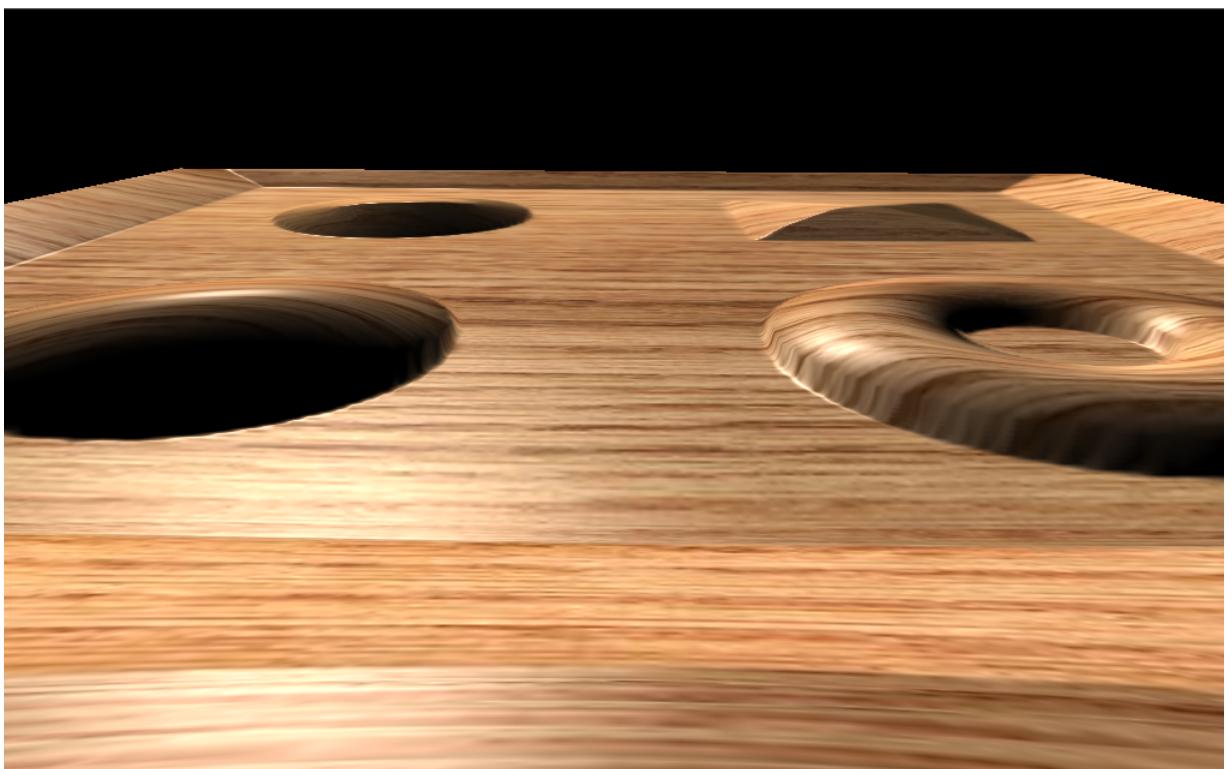


Figura 4.15: *Teste 8*, visualização do *Iterative Parallax Mapping* a pequenos ângulos entre o raio de visão e a superfície.

5 Considerações Finais

Este trabalho analisou e comparou as principais técnicas de Detalhamento de Superfícies utilizadas por aplicações de tempo real, como jogos e simuladores da atualidade. As técnicas foram abordadas e comparadas a medida que eram tratadas, fazendo com que os leitores pudessem identificar seus efeitos visuais e suas vantagens e desvantagens de utilização.

As comparações mostraram que os métodos Iterativos simulam de forma satisfatória o efeito de oclusão. Em particular, o *Cone Step Mapping*, *Parallax Occlusion Mapping* e o *Relief Mapping* obtiveram os melhores resultados garantido o encontro da interseção entre raio de visão e o relevo do Mapa de Níveis com boa precisão e a custos computacionais viáveis. Os métodos de busca são inviáveis quando utilizados separadamente. A Busca Linear obtém bons níveis de qualidade, porém requisita alto custo computacional. A Busca Binária é rápida, mas gera distorções nos casos onde são evidenciados pequenos ângulos entre o raio de visão e o plano de textura. Os métodos Não Iterativos obtiveram resultados inferiores, a pequenos custos computacionais. Dentre eles destacamos o *Iterative Parallax Mapping*, que apresentou bons níveis de qualidade a um número reduzido de iterações. O *Parallax Mapping with Slope Information* por sua vez obteve resultados satisfatórios na maioria dos casos, contudo inferiores ao *Iterative Parallax Mapping*. Por fim, o *Parallax Mapping*, *Parallax Mapping with Offset Limiting* e o *Normal Mapping* obtiveram os piores resultados, sobre tudo evidenciaram os menores custos computacionais.

A abordagem deste trabalho permitiu ainda a identificação dos vários tipos de ferramentas utilizadas pelas técnicas de Detalhamento de Superfícies. Tais como, o modelo de iluminação *Phong*, o deslocamento de texturas, a Busca Linear, a Busca Binária, a interseção por Falsa Posição e o Mapeamento de Espaços vazios. A Figura 5.1 mostra como cada uma destas ferramentas são utilizadas pelas técnicas referenciadas neste trabalho.

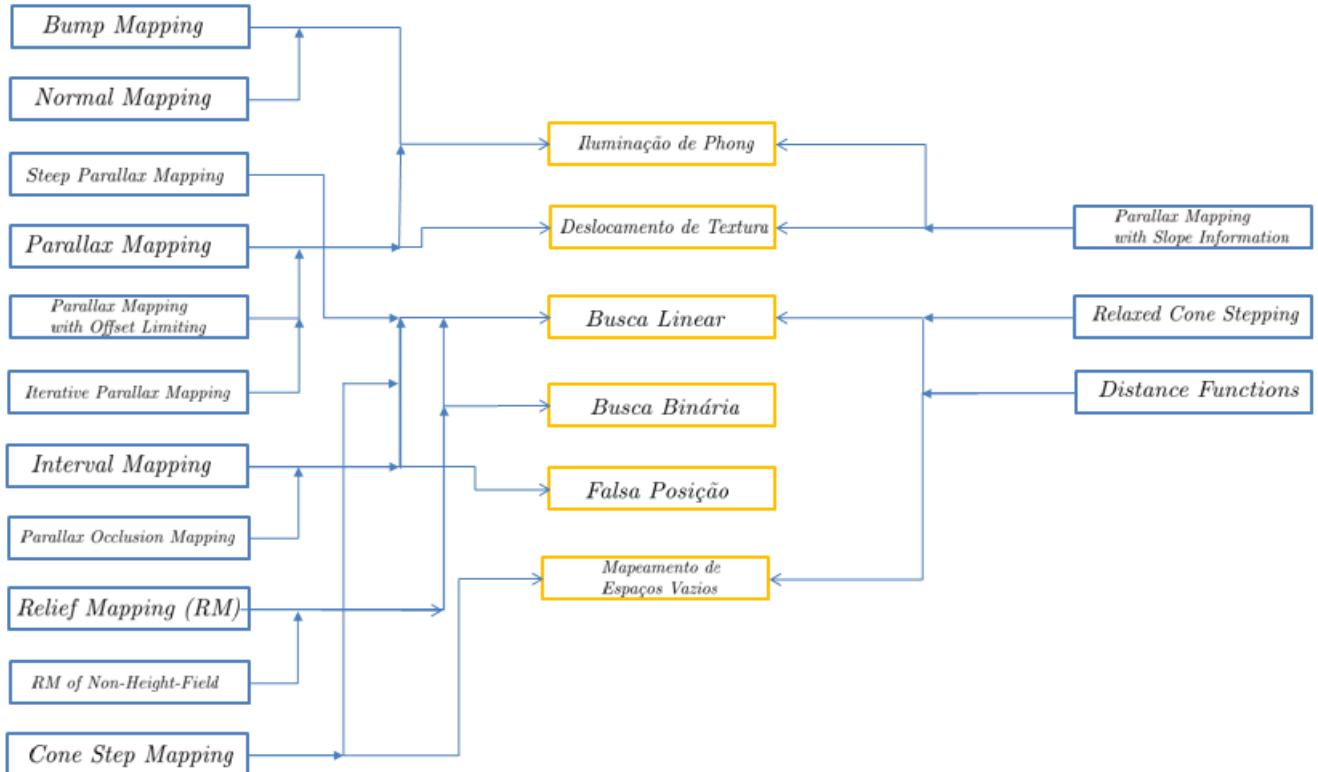


Figura 5.1: As ferramentas para criação de detalhes da meso-estrutura utilizadas pelas técnicas: *Bump Mapping*, *Normal Mapping*, *Steep Parallax Mapping*, *Parallax Mapping*, *Parallax Mapping with Offset Limiting*, *Iterative Parallax Mapping*, *Interval Mapping*, *Parallax Occlusion Mapping*, *Relief Mapping*, *Cone Step Mapping*, *Parallax Occlusion Mapping*, *Parallax Mapping with Slope Information*, *Relaxed Cone Stepping* e *Distance Functions*.

A Detalhamento de Superfícies através do Mapeamento de Texturas

A.1 Mapeamento de Texturas

```

1 varying vec2 Texcoord;
2 void main( void )
3 {
4     gl_Position = ftransform();
5     Texcoord    = gl_MultiTexCoord0.xy;
6
7 }
```

Listing A.1: Normal Mapping Vertex Shader

```

1 uniform sampler2D baseMap;
2
3 varying vec2 Texcoord;
4
5 void main( void )
6 {
7     gl_FragColor = texture2D( baseMap, Texcoord );
8
9 }
```

Listing A.2: Normal Mapping Pixel Shader

A.2 Normal Mapping

Normal Mapping Vertex Shader:

```

1 uniform vec3 fvLightPosition;
2 varying vec2 Texcoord;
3 varying vec3 ViewDirection;
4 varying vec3 LightDirection;
5 attribute vec3 rm_Binormal;
```

```

6 attribute vec3 rm_Tangent;
7
8 void main( void )
9 {
10    gl_Position = ftransform();
11    Texcoord    = gl_MultiTexCoord0.xy;
12    vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
13    vec3 fvViewDirection = - fvObjectPosition.xyz;
14    vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
15    vec3 fvNormal       = gl_NormalMatrix * gl_Normal;
16    vec3 fvBinormal     = gl_NormalMatrix * rm_Binormal;
17    vec3 fvTangent      = gl_NormalMatrix * rm_Tangent;
18    ViewDirection.x = dot( fvTangent, fvViewDirection );
19    ViewDirection.y = dot( fvBinormal, fvViewDirection );
20    ViewDirection.z = dot( fvNormal, fvViewDirection );
21    LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
22    LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
23    LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
24
25 }
```

Listing A.3: Normal Mapping Pixel Shader

Normal Mapping Pixel Shader:

```

1 uniform vec4 fvAmbient;
2 uniform vec4 fvSpecular;
3 uniform vec4 fvDiffuse;
4 uniform float fSpecularPower;
5 uniform sampler2D baseMap;
6 uniform sampler2D bumpMap;
7 varying vec2 Texcoord;
8 varying vec3 ViewDirection;
9 varying vec3 LightDirection;
10
11 void main( void )
12 {
13    vec3 fvLightDirection = normalize( LightDirection );
14    vec3 fvNormal        = normalize( ( texture2D( bumpMap, Texcoord ).xyz
15                                     * 2.0 ) - 1.0 );
16    float fNDotL         = dot( fvNormal, fvLightDirection );
```

```

16    vec3 fvReflection      = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
17        fvLightDirection );
18    vec3 fvViewDirection   = normalize( ViewDirection );
19    float fRDotV           = max( 0.0, dot( fvReflection, fvViewDirection ) );
20    vec4 fvBaseColor        = texture2D( baseMap, Texcoord );
21    vec4 fvTotalAmbient     = fvAmbient * fvBaseColor;
22    vec4 fvTotalDiffuse      = fvDiffuse * fNDotL * fvBaseColor;
23    vec4 fvTotalSpecular     = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
24    gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
25 }
```

Listing A.4: Normal Mapping Pixel Shader

A.3 Parallax Mapping

Parallax Mapping Vertex Shader

```

1 uniform vec3 fvLightPosition;
2 uniform vec3 fvEyePosition;
3 varying vec2 Texcoord;
4 varying vec3 ViewDirection;
5 varying vec3 LightDirection;
6 attribute vec3 rm_Binormal;
7 attribute vec3 rm_Tangent;
8
9 void main( void )
10 {
11     gl_Position = ftransform();
12     Texcoord    = gl_MultiTexCoord0.xy;
13     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
14     vec3 fvViewDirection = - fvObjectPosition.xyz;
15     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
16     vec3 fvNormal       = gl_NormalMatrix * gl_Normal;
17     vec3 fvBinormal     = gl_NormalMatrix * rm_Binormal;
18     vec3 fvTangent       = gl_NormalMatrix * rm_Tangent;
19     ViewDirection.x = dot( fvTangent, fvViewDirection );
20     ViewDirection.y = dot( fvBinormal, fvViewDirection );
21     ViewDirection.z = dot( fvNormal, fvViewDirection );
```

```

22     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
23     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
24     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
25
26 }
```

Listing A.5: Normal Mapping Pixel Shader

Parallax Mapping Pixel Shader

```

1 uniform sampler2D baseMap;
2 uniform sampler2D bumpMap;
3 uniform sampler2D heightMap;
4 uniform vec4 fvAmbient;
5 uniform vec4 fvDiffuse;
6 uniform vec4 fvSpecular;
7 uniform float fSpecularPower;
8 uniform float fScale;
9 uniform float fBias;
10 varying vec2 Texcoord;
11 varying vec3 ViewDirection;
12 varying vec3 LightDirection;
13
14 void main( void )
15 {
16     vec3 fvLightDirection = normalize( LightDirection );
17     vec3 fvViewDirection = normalize( ViewDirection );
18     float ho = texture2D( heightMap, Texcoord );
19     float hsb = fScale*ho + fBias;
20     float t = hsb/fvViewDirection.z;
21
22     vec2 newCoord = Texcoord + t * vec2( fvViewDirection.x, fvViewDirection.y
23                                         );
24     vec3 fvNormal = normalize( ( texture2D( bumpMap, newCoord ).xyz
25                                 * 2.0 ) - 1.0 );
26     float fNDotL = dot( fvNormal, fvLightDirection );
27     vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
28                                   fvLightDirection );
29     float fRDotV = max( 0.0, dot( fvReflection, fvViewDirection ) );
30     vec4 fvBaseColor = texture2D( baseMap, newCoord );
```

```

28     vec4 fvTotalAmbient = fvAmbient * fvBaseColor;
29     vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor;
30     vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
31
32
33     gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
34 }
```

Listing A.6: Normal Mapping Pixel Shader

A.4 Parallax Mapping with Offset Limiting

Parallax Mapping with Offset Limiting Vertex Shader

```

1 uniform vec3 fvLightPosition;
2 uniform vec3 fvEyePosition;
3 varying vec2 Texcoord;
4 varying vec3 ViewDirection;
5 varying vec3 LightDirection;
6 attribute vec3 rm_Binormal;
7 attribute vec3 rm_Tangent;
8
9 void main( void )
10 {
11     gl_Position = ftransform();
12     Texcoord    = gl_MultiTexCoord0.xy;
13
14     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
15     vec3 fvViewDirection = - fvObjectPosition.xyz;
16     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
17     vec3 fvNormal        = gl_NormalMatrix * gl_Normal;
18     vec3 fvBinormal      = gl_NormalMatrix * rm_Binormal;
19     vec3 fvTangent        = gl_NormalMatrix * rm_Tangent;
20     ViewDirection.x = dot( fvTangent, fvViewDirection );
21     ViewDirection.y = dot( fvBinormal, fvViewDirection );
22     ViewDirection.z = dot( fvNormal, fvViewDirection );
23     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
24     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
25     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
26 }
```

27 }

Listing A.7: Normal Mapping Pixel Shader

Parallax Mapping with Offset Limiting Pixel Shader

```

1
2 uniform sampler2D baseMap;
3 uniform sampler2D bumpMap;
4 uniform sampler2D heightMap;
5 uniform vec4 fvAmbient;
6 uniform vec4 fvDiffuse;
7 uniform vec4 fvSpecular;
8 uniform float fSpecularPower;
9 uniform float fScale;
10 uniform float fBias;
11 varying vec2 Texcoord;
12 varying vec3 ViewDirection;
13 varying vec3 LightDirection;
14
15
16
17 void main( void )
18 {
19     vec3 fvLightDirection = normalize( LightDirection );
20     vec3 fvViewDirection = normalize( ViewDirection );
21     float ho = texture2D( heightMap, Texcoord );
22     float hsb = fScale*ho + fBias;
23     float t = hsb;
24
25     vec2 newCoord = Texcoord + t * vec2( fvViewDirection.x, fvViewDirection.y
26                                         );
26     vec3 fvNormal = normalize( ( texture2D( bumpMap, newCoord ).xyz
27                                 * 2.0 ) - 1.0 );
28     float fNDotL = dot( fvNormal, fvLightDirection );
29     vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
30                                   fvLightDirection );
31     float fRDotV = max( 0.0, dot( fvReflection, fvViewDirection ) );
32     vec4 fvBaseColor = texture2D( baseMap, newCoord );
33     vec4 fvTotalAmbient = fvAmbient * fvBaseColor;

```

```

32     vec4 fvTotalDiffuse    = fvDiffuse * fNDotL * fvBaseColor;
33     vec4 fvTotalSpecular   = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
34     gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
35
36 }
```

Listing A.8: Normal Mapping Pixel Shader

A.5 Parallax Mapping with Slope Information

Parallax Mapping with Slope Information Vertex Shader

```

1 uniform vec3 fvLightPosition;
2 uniform vec3 fvEyePosition;
3
4
5 varying vec2 Texcoord;
6 varying vec3 ViewDirection;
7 varying vec3 LightDirection;
8
9 attribute vec3 rm_Binormal;
10 attribute vec3 rm_Tangent;
11
12 void main( void )
13 {
14     gl_Position = ftransform();
15     Texcoord    = gl_MultiTexCoord0.xy;
16     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
17     vec3 fvViewDirection = 0 - fvObjectPosition.xyz;
18     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
19     vec3 fvNormal        = gl_NormalMatrix * gl_Normal;
20     vec3 fvBinormal      = gl_NormalMatrix * rm_Binormal;
21     vec3 fvTangent        = gl_NormalMatrix * rm_Tangent;
22     ViewDirection.x = dot( fvTangent, fvViewDirection );
23     ViewDirection.y = dot( fvBinormal, fvViewDirection );
24     ViewDirection.z = dot( fvNormal, fvViewDirection );
25     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
26     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
27     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
28 }
```

29 }

Listing A.9: Normal Mapping Pixel Shader

Parallax Mapping with Slope Information Pixel Shader

```

1
2 uniform sampler2D baseMap;
3 uniform sampler2D bumpMap;
4 uniform sampler2D heightMap;
5 uniform vec4 fvAmbient;
6 uniform vec4 fvDiffuse;
7 uniform vec4 fvSpecular;
8 uniform float fSpecularPower;
9 uniform float fScale;
10 uniform float fBias;
11 varying vec2 Texcoord;
12 varying vec3 ViewDirection;
13 varying vec3 LightDirection;
14
15
16 void main( void )
17 {
18     vec3 fvLightDirection = normalize( LightDirection );
19     vec3 fvViewDirection = normalize( ViewDirection );
20     float ho = texture2D( heightMap, Texcoord );
21     vec3 normalAho = normalize( ( texture2D( bumpMap, Texcoord ).xyz * 2.0
22         ) - 1.0 );
23     float hsb = fScale*ho + fBias;
24     float t = normalAho.z * hsb;
25     vec2 newCoord = Texcoord + t * vec2( fvViewDirection.x, fvViewDirection.y
26         );
27     vec3 fvNormal = normalize( ( texture2D( bumpMap, newCoord ).xyz
28         * 2.0 ) - 1.0 );
29     float fNDotL = dot( fvNormal, fvLightDirection );
30     vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
31         fvLightDirection );
32     float fRDotV = max( 0.0, dot( fvReflection, fvViewDirection ) );
33     vec4 fvBaseColor = texture2D( baseMap, newCoord );
34     vec4 fvTotalAmbient = fvAmbient * fvBaseColor;

```

```

31    vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor;
32    vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
33    gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
34
35
36 }
```

Listing A.10: Normal Mapping Pixel Shader

A.6 Iterative Parallax Mapping

Iterative Parallax Mapping Vertex Shader

```

1 uniform vec3 fvLightPosition ;
2 uniform vec3 fvEyePosition ;
3 varying vec2 Texcoord ;
4 varying vec3 ViewDirection ;
5 varying vec3 LightDirection ;
6 attribute vec3 rm_Binormal ;
7 attribute vec3 rm_Tangent ;
8
9 void main( void )
10 {
11     gl_Position = ftransform();
12     Texcoord    = gl_MultiTexCoord0.xy;
13     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
14     vec3 fvViewDirection = 0 - fvObjectPosition.xyz;
15     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
16     vec3 fvNormal       = gl_NormalMatrix * gl_Normal;
17     vec3 fvBinormal     = gl_NormalMatrix * rm_Binormal;
18     vec3 fvTangent       = gl_NormalMatrix * rm_Tangent;
19     ViewDirection.x = dot( fvTangent, fvViewDirection );
20     ViewDirection.y = dot( fvBinormal, fvViewDirection );
21     ViewDirection.z = dot( fvNormal, fvViewDirection );
22     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
23     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
24     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
25
26 }
```

Listing A.11: Normal Mapping Pixel Shader

Iterative Parallax Mapping Pixel Shader

```

1 uniform vec4 fvSpecular;
2 uniform float fSpecularPower;
3 uniform int numIterations;
4 uniform float fScale;
5 uniform float fBias;
6 varying vec2 Texcoord;
7 varying vec3 ViewDirection;
8 varying vec3 LightDirection;
9
10 void main( void )
11 {
12     vec3 fvLightDirection = normalize( LightDirection );
13     vec3 fvViewDirection = normalize( ViewDirection );
14     vec2 coordLoop = Texcoord;
15     int i = 0;
16     while( i < numIterations ){
17         float ho = texture2D( heightMap , coordLoop );
18         vec3 normalAho = normalize( ( texture2D( bumpMap, coordLoop ) .xyz *
19             2.0 ) - 1.0 );
20         float hsb = fScale * ho + fBias;
21         float t = normalAho.z * hsb;
22         coordLoop = coordLoop + t * vec2( fvViewDirection .x, fvViewDirection
23             .y );
24         i++;
25     }
26     vec3 fvNormal = normalize( ( texture2D( bumpMap, coordLoop ) .
27         xyz * 2.0 ) - 1.0 );
28     float fNDotL = dot( fvNormal , fvLightDirection );
29     vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
30         fvLightDirection );
31     float fRDotV = max( 0.0 , dot( fvReflection , fvViewDirection ) );
32     vec4 fvBaseColor = texture2D( baseMap , coordLoop );
33     vec4 fvTotalAmbient = fvAmbient * fvBaseColor;
34     vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor;
35     vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV, fSpecularPower ) );

```

```

32     gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
33 }
```

Listing A.12: Normal Mapping Pixel Shader

A.7 Busca Linear

Busca Linear Vertex Shader

```

1 uniform vec3 fvLightPosition;
2 uniform vec3 fvEyePosition;
3 varying vec2 Texcoord;
4 varying vec3 ViewDirection;
5 varying vec3 LightDirection;
6 varying vec3 pixelPos;
7 attribute vec3 rm_Binormal;
8 attribute vec3 rm_Tangent;
9
10 void main( void )
11 {
12     gl_Position = ftransform();
13     Texcoord    = gl_MultiTexCoord0.xy;
14     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
15     vec3 fvViewDirection = -fvObjectPosition.xyz;
16     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
17     vec3 fvNormal       = gl_NormalMatrix * gl_Normal;
18     vec3 fvBinormal     = gl_NormalMatrix * rm_Binormal;
19     vec3 fvTangent       = gl_NormalMatrix * rm_Tangent;
20     ViewDirection.x = dot( fvTangent, fvViewDirection );
21     ViewDirection.y = dot( fvBinormal, fvViewDirection );
22     ViewDirection.z = dot( fvNormal, fvViewDirection );
23     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
24     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
25     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
26 }
```

Listing A.13: Normal Mapping Pixel Shader

Busca Linear Pixel Shader

```

1 uniform vec4 fvAmbient;
```

```
2 uniform vec4 fvSpecular;
3 uniform vec4 fvDiffuse;
4 uniform float fSpecularPower;
5 uniform bool wireFrame;
6 uniform sampler2D baseMap;
7 uniform sampler2D bumpMap;
8 uniform sampler2D heightMap;
9 uniform float userDepth;
10 uniform float userTile;
11 uniform float heightScale;
12 uniform int linearSteps;
13 uniform float Bias;
14 uniform float fUser;
15 varying vec2 Texcoord;
16 varying vec3 ViewDirection;
17 varying vec3 LightDirection;
18 varying vec3 pixelPos;
19
20 float linearSearch(in vec2 initialPos, in vec2 offsetDir){
21
22     float size = 1.0/linearSteps;
23     float depth = 1;
24     float best_depth = 0.0;
25     for(int i=0; i<linearSteps; i++){
26         depth -= size;
27         float height = texture2D(heightMap, initialPos + offsetDir * depth).x
28             * heightScale + Bias;
29         if(best_depth < 0.01)
30             if(depth <= height){
31                 best_depth = depth;
32             }
33
34     return best_depth;
35 }
36
37 void main( void )
38 {
39     vec3 fvViewDirection = normalize( ViewDirection );
40     vec2 offsetDir = fvViewDirection.xy * fUser / fvViewDirection.z;
```

```

41 float depth = linearSearch(Texcoord, offsetDir);
42 vec2 newTexCoord = Texcoord + depth*offsetDir;
43 vec3 fvLightDirection = normalize( LightDirection );
44 vec3 fvNormal           = normalize( ( texture2D( bumpMap, newTexCoord ) .
45                                     xyz * 2.0 ) - 1.0 );
46 float fNDotL           = dot( fvNormal, fvLightDirection );
47 vec3 fvReflection       = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
48                                     fvLightDirection );
49 float fRDotV           = max( 0.0, dot( fvReflection, fvViewDirection ) );
50 vec4 fvBaseColor        = texture2D( baseMap, newTexCoord );
51 vec4 fvTotalAmbient     = fvAmbient * fvBaseColor;
52 vec4 fvTotalDiffuse      = fvDiffuse * fNDotL * fvBaseColor;
53 vec4 fvTotalSpecular     = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
54 gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
55
56 }
```

Listing A.14: Normal Mapping Pixel Shader

A.8 Busca Binária

Busca Binária Vertex Shader

```

1 uniform vec3 fvLightPosition;
2 uniform vec3 fvEyePosition;
3 varying vec2 Texcoord;
4 varying vec3 ViewDirection;
5 varying vec3 LightDirection;
6 varying vec3 pixelPos;
7 attribute vec3 rm_Binormal;
8 attribute vec3 rm_Tangent;
9
10 void main( void )
11 {
12     gl_Position = ftransform();
13     Texcoord    = gl_MultiTexCoord0.xy;
14     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
15     vec3 fvViewDirection   = -fvObjectPosition.xyz;
16     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
```

```

17    vec3 fvNormal          = gl_NormalMatrix * gl_Normal;
18    vec3 fvBinormal        = gl_NormalMatrix * rm_Binormal;
19    vec3 fvTangent          = gl_NormalMatrix * rm_Tangent;
20    ViewDirection.x = dot( fvTangent, fvViewDirection );
21    ViewDirection.y = dot( fvBinormal, fvViewDirection );
22    ViewDirection.z = dot( fvNormal, fvViewDirection );
23    LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
24    LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
25    LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
26 }

```

Listing A.15: Normal Mapping Pixel Shader

Busca Binária Pixel Shader

```

1 uniform vec4 fvAmbient;
2 uniform vec4 fvSpecular;
3 uniform vec4 fvDiffuse;
4 uniform float fSpecularPower;
5 uniform sampler2D baseMap;
6 uniform sampler2D bumpMap;
7 uniform sampler2D heightMap;
8 uniform float userDepth;
9 uniform float userTile;
10 uniform float heightScale;
11 uniform int binarySteps;
12 uniform float Bias;
13 varying vec2 Texcoord;
14 varying vec3 ViewDirection;
15 varying vec3 LightDirection;
16 varying vec3 pixelPos;
17
18 float binarySearch(in vec2 initialPos, in vec2 offsetDir){
19     float minDepth = 0;
20     float maxDepth = 1;
21     float depthTry;
22     float height;
23     for(int i=0; i<binarySteps; i++){
24         depthTry = (minDepth + maxDepth) / 2;
25         height = texture2D(heightMap, initialPos + offsetDir*depthTry).x *

```

```

26     if( depthTry < height ){
27         minDepth = depthTry ;
28     } else {
29         maxDepth = depthTry ;
30     }
31 }
32
33 }
34
35 void main( void )
36 {
37     vec3 fvViewDirection = normalize( ViewDirection );
38     vec2 offsetDir = fvViewDirection .xy / fvViewDirection .z;
39     float depth = binarySearch( Texcoord , offsetDir );
40     vec2 newTexCoord = Texcoord + depth*offsetDir ;
41     vec3 fvLightDirection = normalize( LightDirection );
42     vec3 fvNormal = normalize( ( texture2D( bumpMap, newTexCoord )
43                                 .xyz * 2.0 ) - 1.0 );
44     float fNDotL = dot( fvNormal , fvLightDirection );
45     vec3 fvReflection = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
46                                   fvLightDirection );
47     float fRDotV = max( 0.0 , dot( fvReflection , fvViewDirection ) );
48
49     vec4 fvBaseColor = texture2D( baseMap , newTexCoord );
50     vec4 fvTotalAmbient = fvAmbient * fvBaseColor ;
51     vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor ;
52     vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV , fSpecularPower ) );
53     gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
54 }
```

Listing A.16: Normal Mapping Pixel Shader

A.9 Relief Mapping

Relief Mapping Vertex Shader

```

1 uniform vec3 fvLightPosition ;
2 uniform vec3 fvEyePosition ;
3 varying vec2 Texcoord ;
4 varying vec3 ViewDirection ;
```

```

5 varying vec3 LightDirection;
6 varying vec3 pixelPos;
7 attribute vec3 rm_Binormal;
8 attribute vec3 rm_Tangent;
9
10 void main( void )
11 {
12     gl_Position = ftransform();
13     Texcoord    = gl_MultiTexCoord0.xy;
14     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
15     vec3 fvViewDirection = - fvObjectPosition.xyz;
16     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
17     vec3 fvNormal        = gl_NormalMatrix * gl_Normal;
18     vec3 fvBinormal       = gl_NormalMatrix * rm_Binormal;
19     vec3 fvTangent        = gl_NormalMatrix * rm_Tangent;
20     ViewDirection.x = dot( fvTangent, fvViewDirection );
21     ViewDirection.y = dot( fvBinormal, fvViewDirection );
22     ViewDirection.z = dot( fvNormal, fvViewDirection );
23     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
24     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
25     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
26 }
```

Listing A.17: Normal Mapping Pixel Shader

Relief Mapping Pixel Shader

```

1 uniform vec4 fvAmbient;
2 uniform vec4 fvSpecular;
3 uniform vec4 fvDiffuse;
4 uniform float fSpecularPower;
5 uniform sampler2D baseMap;
6 uniform sampler2D bumpMap;
7 uniform sampler2D heightMap;
8 uniform float userDepth;
9 uniform float userTile;
10 varying vec2 Texcoord;
11 varying vec3 ViewDirection;
12 varying vec3 LightDirection;
13 varying vec3 pixelPos;
14
```

```
15
16 float ray_intersect(in vec2 initialPos , in vec2 offsetDir){
17     int linearSteps=10;
18     int binarySteps=5;
19     float depth_step = 1.0/linearSteps ;
20     float size = depth_step ;
21     float depth=0.0;
22     float best_depth = 1.0;
23
24     for(int i=0; i<linearSteps -1;i++){
25         depth += size;
26         float height = texture2D(heightMap , initialPos + offsetDir*depth).x;
27         if(best_depth>0.996)
28             if(depth >= height)
29                 best_depth = depth;
30     }
31
32     depth = best_depth;
33     for(int i=0; i<binarySteps ;i++){
34         size *= 0.5;
35         float height = texture2D(heightMap , initialPos + offsetDir*depth).x;
36         if(depth>= height){
37             best_depth = depth;
38             depth -= 2*size ;
39         }
40         depth += size;
41     }
42     return best_depth;
43 }
44
45 void main( void )
46 {
47     vec3 fvViewDirection = normalize( ViewDirection );
48     vec2 offsetDir = fvViewDirection.xy * userDepth/fvViewDirection.z;
49     vec2 offsetPos = Texcoord * userTile;
50     float depth = ray_intersect(offsetPos , offsetDir);
51     vec2 newTexCoord = offsetPos + depth*offsetDir;
52     vec3 fvLightDirection = normalize( LightDirection );
53     vec3 fvNormal           = normalize( ( texture2D( bumpMap, newTexCoord ) .
xyz * 2.0 ) - 1.0 );
```

```

54 float fNDotL           = dot( fvNormal, fvLightDirection );
55 vec3 fvReflection      = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
56                                     fvLightDirection );
56 float fRDotV           = max( 0.0, dot( fvReflection, fvViewDirection ) );
57 vec4 fvBaseColor        = texture2D( baseMap, newTexCoord );
58 vec4 fvTotalAmbient     = fvAmbient * fvBaseColor;
59 vec4 fvTotalDiffuse      = fvDiffuse * fNDotL * fvBaseColor;
60 vec4 fvTotalSpecular     = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
61 gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
62 }
```

Listing A.18: Normal Mapping Pixel Shader

A.10 Parallax Occlusion Mapping

Parallax Occlusion Mapping Vertex Shader

```

1 uniform vec3 fvLightPosition;
2 uniform vec3 fvEyePosition;
3 varying vec2 Texcoord;
4 varying vec3 ViewDirection;
5 varying vec3 LightDirection;
6 varying vec3 pixelPos;
7 attribute vec3 rm_Binormal;
8 attribute vec3 rm_Tangent;
9
10 void main( void )
11 {
12     gl_Position = ftransform();
13     Texcoord    = gl_MultiTexCoord0.xy;
14     vec4 fvObjectPosition = gl_ModelViewMatrix * gl_Vertex;
15     vec3 fvViewDirection = - fvObjectPosition.xyz;
16     vec3 fvLightDirection = fvLightPosition - fvObjectPosition.xyz;
17     vec3 fvNormal       = gl_NormalMatrix * gl_Normal;
18     vec3 fvBinormal     = gl_NormalMatrix * rm_Binormal;
19     vec3 fvTangent       = gl_NormalMatrix * rm_Tangent;
20     ViewDirection.x = dot( fvTangent, fvViewDirection );
21     ViewDirection.y = dot( fvBinormal, fvViewDirection );
22     ViewDirection.z = dot( fvNormal, fvViewDirection );
```

```

23     LightDirection.x = dot( fvTangent, fvLightDirection.xyz );
24     LightDirection.y = dot( fvBinormal, fvLightDirection.xyz );
25     LightDirection.z = dot( fvNormal, fvLightDirection.xyz );
26 }
```

Listing A.19: Normal Mapping Pixel Shader

Parallax Occlusion Mapping Pixel Shader

```

1
2 uniform vec4 fvAmbient;
3 uniform vec4 fvSpecular;
4 uniform vec4 fvDiffuse;
5 uniform float fSpecularPower;
6
7 uniform sampler2D baseMap;
8 uniform sampler2D bumpMap;
9 uniform sampler2D heightMap;
10 uniform float userDepth;
11 uniform float userTile;
12
13
14 varying vec2 Texcoord;
15
16 varying vec3 ViewDirection;
17
18
19 varying vec3 LightDirection;
20 varying vec3 pixelPos;
21 varying vec3 normal;
22
23
24 uniform int secantSteps;
25
26 float fHeightMapScale      = 0.04;
27 float fTexelsPerSide       = sqrt(256.0 * 256.0 * 1);
28
29 uniform int nMaxSamples;
30 uniform int nMinSamples;;
31
32
```

```
33
34
35 vec2 Intersection(vec2 texCoord, float steps)
36 {
37
38     float fStepSize = 1.0 / (float)steps;
39
40     float fParallaxLimit = length(ViewDirection.xy) / ViewDirection.z;
41     fParallaxLimit *= fHeightMapScale;
42
43     vec2 vOffset = normalize( -ViewDirection.xy );
44     vOffset = vOffset * fParallaxLimit;
45
46
47     vec2 vOffsetStep = fStepSize * vOffset;
48     vec2 vCurrOffset = float2( 0, 0 );
49     vec2 vLastOffset = float2( 0, 0 );
50     vec2 vFinalOffset = float2( 0, 0 );
51
52     vec4 vCurrSample;
53     vec4 vLastSample;
54
55     float stepHeight = 1.0;
56     int nCurrSample = 0;
57
58
59     while ( nCurrSample < steps )
60     {
61         vCurrSample = texture2D( heightMap, texCoord + vCurrOffset );
62         if ( vCurrSample.x > stepHeight )
63         {
64             // calculate the linear intersection point
65             float Ua = (vLastSample.x - (stepHeight+fStepSize)) / ( fStepSize
66             + (vCurrSample.x - vLastSample.x));
67             vFinalOffset = vLastOffset + Ua * vOffsetStep;
68
69             vCurrSample = texture2D( heightMap, texCoord + vFinalOffset );
70             nCurrSample = steps + 1;
71         }
72         else
```

```
72     {
73         nCurrSample++;
74         stepHeight -= fStepSize;
75         vLastOffset = vCurrOffset;
76         vCurrOffset += vOffsetStep;
77         vLastSample = vCurrSample;
78     }
79 }
80
81
82 return texCoord + vCurrOffset;
83 }
84
85
86 void main( void )
87 {
88
89
90     vec3 fvViewDirection = normalize( ViewDirection ) ;
91     vec2 offsetDir = fvViewDirection.xy * userDepth/fvViewDirection.z;
92
93     vec2 offsetPos = Texcoord;
94     vec3 E = normalize( ViewDirection );
95     vec3 N = normalize( normal );
96
97     int nNumSamples = (int)mix( nMaxSamples, nMinSamples, dot( E, N ) );
98     vec2 newTexCoord = Intersection( offsetPos, nNumSamples );
99
100    vec3 fvLightDirection = normalize( LightDirection );
101    vec3 fvNormal           = normalize( ( texture2D( bumpMap, newTexCoord ).xyz * 2.0 ) - 1.0 );
102    float fNDotL            = dot( fvNormal, fvLightDirection );
103
104    vec3 fvReflection        = normalize( ( ( 2.0 * fvNormal ) * fNDotL ) -
105                                            fvLightDirection );
106
107    float fRDotV             = max( 0.0, dot( fvReflection, fvViewDirection ) );
108    vec4 fvBaseColor          = texture2D( baseMap, newTexCoord );
```

```

109
110     vec4 fvTotalAmbient = fvAmbient * fvBaseColor;
111     vec4 fvTotalDiffuse = fvDiffuse * fNDotL * fvBaseColor;
112     vec4 fvTotalSpecular = fvSpecular * ( pow( fRDotV, fSpecularPower ) );
113
114     gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
115
116 }
```

Listing A.20: Normal Mapping Pixel Shader

A.11 Cone Step Mapping

Cone Step Mapping Vertex Shader

```

1
2 uniform float csm_gain;
3 uniform float csm_offset;
4 uniform float texsize;
5 uniform vec3 fvLightPosition;
6
7
8 varying vec2 texCoord;
9 varying vec3 vertex_pos;
10 varying vec3 light_dir;
11 varying float vertex_dist;
12 varying float light_dist;
13 varying vec3 ViewDirection;
14
15 attribute vec3 rm_Binormal;
16 attribute vec3 rm_Tangent;
17
18 void main(void)
19 {
20     vec3 eyeSpaceVert = (gl_ModelViewMatrix * gl_Vertex).xyz;
21     vec3 eyeSpaceLight = fvLightPosition;
22
23
24     vec3 fvViewDirection = -eyeSpaceVert.xyz;
```

```

26
27     vec3 eyeSpaceTangent = normalize(gl_NormalMatrix * rm_Tangent);
28     vec3 eyeSpaceBinormal = normalize(gl_NormalMatrix * rm_Binormal);
29     vec3 eyeSpaceNormal = normalize(gl_NormalMatrix * gl_Normal);
30
31
32
33
34     vertex_pos = vec3 (
35         dot (eyeSpaceTangent , eyeSpaceVert) ,
36         dot (eyeSpaceBinormal , eyeSpaceVert) ,
37         dot (eyeSpaceNormal , eyeSpaceVert));
38
39
40
41     light_dir = vec3 (
42         dot (eyeSpaceTangent , eyeSpaceLight) ,
43         dot (eyeSpaceBinormal , eyeSpaceLight) ,
44         dot (eyeSpaceNormal , eyeSpaceLight))
45         - vertex_pos ;
46
47     ViewDirection.x = dot( eyeSpaceTangent , fvViewDirection );
48     ViewDirection.y = dot( eyeSpaceBinormal , fvViewDirection );
49     ViewDirection.z = dot( eyeSpaceNormal , fvViewDirection );
50
51
52     vertex_dist = length (eyeSpaceVert);
53     light_dist = length (light_dir);
54     texCoord = gl_MultiTexCoord0.xy;
55
56
57
58     gl_Position = ftransform ();
59 }
```

Listing A.21: Normal Mapping Pixel Shader

Cone Step Mapping Pixel Shader

```

1
2 uuniform float depth;
```

```
3 uniform float texsize;
4 uniform vec4 fvDiffuse;
5 uniform vec4 fvAmbient;
6 uniform vec4 fvSpecular;
7 uniform float fSpecularPower;
8 uniform float fAtt;
9
10 varying vec2 texCoord;
11 varying vec3 vertex_pos;
12 varying vec3 light_dir;
13 varying vec3 ViewDirection;
14 varying float light_dist;
15
16 uniform sampler2D stepmap;
17 uniform sampler2D texmap;
18 uniform sampler2D height;
19 uniform sampler2D normalMap;
20
21 uniform int conesteps;
22
23 void intersect_cone_step (inout vec3 dp, in vec3 ds);
24
25
26 void main(void)
27 {
28     vec4 n, c;
29     vec3 l, s;
30     vec3 pt_eye, pt_light;
31     float a;
32
33     vec3 fvViewDirection = normalize(ViewDirection);
34
35     a = -depth / vertex_pos.z;
36     s = vertex_pos * a;
37     s.z = 1.0;
38
39
40     pt_eye = vec3 (texCoord, 0.0);
41     intersect_cone_step (pt_eye, s);
42 }
```

```

43    vec3 fvLightDirection = normalize( light_dir );
44    vec3 fvNormal           = normalize( ( texture2D( normalMap , pt_eye.xy ) .
45                                         xyz * 2.0 ) - 1.0 );
46    float fNDotL            = dot( fvNormal , fvLightDirection );
47
48
49
50    vec3 fvReflection        = normalize( ( 2.0 * fvNormal ) * fNDotL ) -
51                                         fvLightDirection );
52
53
54    float fRDotV             = max( 0.0 , dot( fvReflection , fvViewDirection ) )
55                                         );
56
57    vec4 fvBaseColor          = texture2D( texmap , pt_eye.xy );
58    fvBaseColor.a = 0;
59
60    vec4 fvTotalAmbient       = fvAmbient * fvBaseColor;
61
62    vec4 fvTotalDiffuse        = fvDiffuse * fNDotL * fvBaseColor;
63
64    vec4 fvTotalSpecular       = fvSpecular * ( pow( fRDotV , fSpecularPower ) );
65
66
67
68    gl_FragColor = ( fvTotalAmbient + fvTotalDiffuse + fvTotalSpecular );
69
70
71
72
73 }
```

Listing A.22: Normal Mapping Pixel Shader

Referências Bibliográficas

- [1] J. F. Blinn. Simulation of wrinkled surfaces. *SIGGRAPH*, 12:286292, 1978.
- [2] J. Cohen, M. Olano, and D. Manocha. Appearance-preserving simplification. In *IN PROC. SIGGRAPH 98*, 1998.
- [3] R. Cook. Shade trees. In *IN PROC. SIGGRAPH 84*, 1984.
- [4] D. S.-C. Crespo Dalmau. *Core Techniques and Algorithms in Game Programming*. New Riders Games, 2004.
- [5] M. P. B. Donald D. Hearn. *Computer Graphics with OpenGL*. Prentice Hall, third edition, 2004.
- [6] D. H. Eberly. *3D Game Engine Design: A Practical Approach to Real-Time Computer Graphics*. MORGAN KAUFMANN, 2006.
- [7] W. Engel. *ShaderX3: Advanced Rendering with DirectX and OpenGL (Shaderx Series)*. Charles River Media, Inc, 2004.
- [8] M. M. d. O. N. Fabio Policarpo. Relief mapping of non-height-field surface details. In *SI3D*, 2006.
- [9] R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley Professional, 2003.
- [10] R. Fosner. *Real-Time Shader Programming*. San Francisco: Publishers. 2003. Morgan Kaufmann, 2003.
- [11] T. Kaneko, T. Takahei, M. Inami, N. Kawakami, Y. Yanagida, T. Maeda, and S. Tachi. Detailed shape representation with parallax mapping. In *205–208*, 2001.
- [12] E. Lengyel. *Mathematics for 3D Game Programming & Computer Graphics*. Charles River Media, 2001.
- [13] T. U. László Szirmay-Kalos. Displacement mapping on the gpu state of the art. In *Computer Graphics Forum*, 2008.

- [14] F. P. Manuel M. Oliveira. An efficient representation for surface details, 2005.
- [15] M. McGuire and M. McGuire. Steep parallax mapping. *I3D*, 2005.
- [16] H. Nguyen. *GPU Gems 3*. Addison-Wesley Professional, 2007.
- [17] J. C. B. PEERCY, M.; AIREY. Efficient bump mapping hardware. *Computer Graphics*, 31:303,306, 1997.
- [18] M. Pharr and R. Fernando. *GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation*. Addison-Wesley Professional, 2005.
- [19] F. Policarpo, M. M. Oliveira, and J. a. L. D. Comba. Real-time relief mapping on arbitrary polygonal surfaces. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, 2005.
- [20] M. Premecz. Iterative parallax mapping with slope information. In *In Central European Seminar on Computer Graphics*, 2006.
- [21] E. Risser, M. Shah, and S. Pattanaik. Interval mapping.
- [22] R. J. Rost. *OpenGL® Shading Language*. Addison-Wesley Professional, 2006.
- [23] D. Shreiner. *OpenGL® Programming Guide: The Official Guide to Learning OpenGL®*. Addison-Wesley Professional, 2009.
- [24] S. St-Laurent. *Shaders for Game Programmers and Artists*. Course Technology Ptr, 2004.
- [25] M. K. TAKAHASHI M. Gpu based interactive displacement mapping. Technical report, Institute of Electronics, Information and Communication Engineers, 2005.
- [26] N. Tatarchuk. Dynamic parallax occlusion mapping with approximate soft shadows. In *In SIGGRAPH 06: ACM SIGGRAPH 2006 Courses*, ACM, 2006.
- [27] T. Welsh. Parallax mapping with offset limiting: A perpixel approximation of uneven surfaces. Technical report, 2004.
- [28] D. Wolff. *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing, 2011.