Carla Florentino de Souza
Edson Leite Araújo
Jennifer Chuin Lee

Mapeamento de Textura
Uma Introdução

4 de Julho de 2011

Universidade Estadual de Campinas - UNICAMP

Faculdade de Engenharia Elétrica - FEEC
IA725 - Computação Gráfica I
Profa. Wu Shin-Ting

Conteúdo

1. Introdução

Quando os detalhes se tornam refinados e mais intrincados, a modelagem a partir de polígonos ou outras primitivas geométricas torna-se impraticável. Uma alternativa é mapear uma imagem, que contenha os detalhes desejados, produzidos de forma sintética ou digital, e mapeá-la sobre a superfície que se deseja produzir o efeito. Esta técnica primeiro pensada e desenvolvida por Catmull [3] e em seguida refinada por Blinn e Newell[2]. Ficou conhecida como *mapeamento de textura (texture mapping)*. A imagem utilizada é chamada de *mapa de textura (texture map)* e seus elementos individuais são freqüentemente chamados de *texels*.

Texture Mapping tornou-se uma poderosa técnica na adição de realismo à cenas geradas por computador e tem sido usada com muita freqüência, na indústria de jogos, produzindo efeitos muito bons a um custo relativamente barato, tendo em vista que mapas de textura são amplamente suportados em hardwares gráficos.

Com o passar do tempo, percebeu-se que os mapas de textura podem carregar quaisquer tipos de informações e atualmente estão sendo utilizadas de três formas básicas:

- O mapa de textura pode conter cores, que são aplicadas para a superfície, agindo de maneira semelhante a um adesivo sobre a mesma. Atuando desta maneira, os cálculos de iluminação são desnecessários pois a cor de um determinado pixel será sobreposta pela cor do pixel correspondente a ele no mapa de textura.

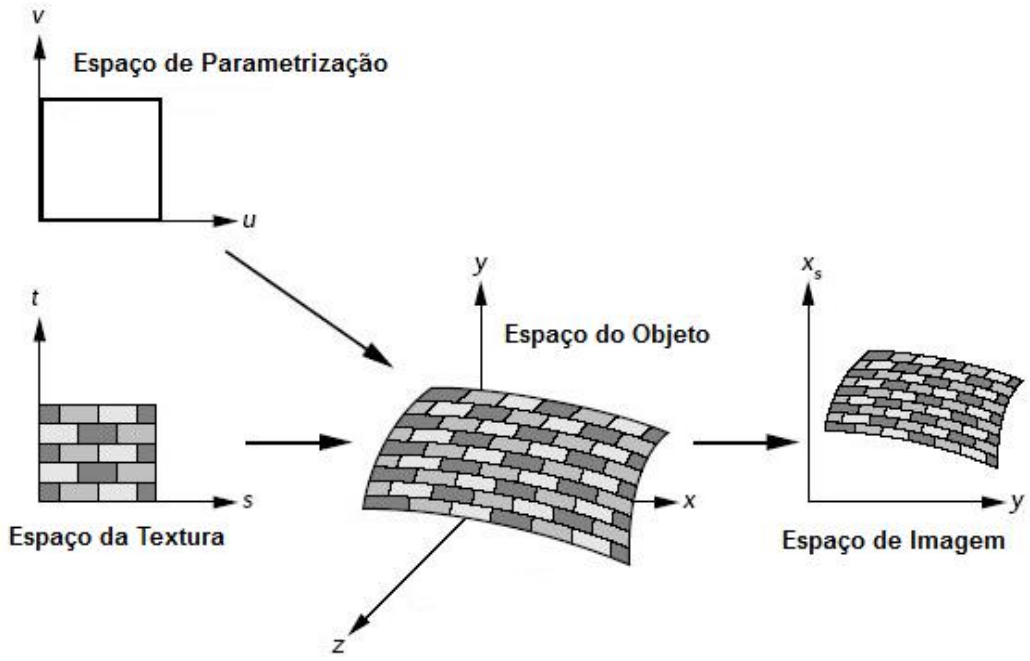
- Pode conter atributos tais como cor, brilho ou transparência que afetam a aparência da superfície após a aplicação do modelo de iluminação. Neste caso, os atributos contidos no mapa de textura serão combinados com os resultados obtidos após a iluminação.

- Em último caso, o mapa de textura pode também conter coeficientes de refletividade, deslocamentos das normais ou outros parâmetros que podem ser utilizados no modelo de iluminação. Neste caso, os valores do mapa de textura modificam as propriedades da superfície, uma vez que são usados como entrada para o modelo de iluminação. Um exemplo disto é a técnica conhecida como *Bump Mapping*.

2. Procedimento Básico

Embora existam várias abordagens para o mapeamento de texturas, todas requerem uma seqüência de passos que envolvem o mapeamento entre três ou quatro referenciais diferentes:

- Referencial do objeto, onde estão os objetos sobre os quais a textura será aplicada;
- Referencial de textura, usado para localizar posições na textura;
- Referencial paramétrico, usado na parametrização da superfícies dos objetos.



Na maioria das aplicações, texturas surgem como imagens bi-dimensionais e estas podem ser geradas por programas ou obtida através de imagens fotográficas ou através de scanners. mas, sem levar em consideração a sua origem, todas são abordadas como vetores bi-dimensionais na memória do computador, onde cada elemento pode é acessado como $T(u, v)$, onde as variáveis u e v são conhecidas como *coordenadas de textura*. Sem perda de generalidade, podemos escalar nossa textura de modo que as coordenadas u e v variem dentro do intervalo $[0, 1]$.

Um *mapeamento de textura* associa um texel a cada ponto sobre o objeto que por sua vez é mapeado em coordenadas de imagem para posterior exibição. Se o objeto é representado

em coordenadas homogêneas, ou seja

$$(x, y, z, w)$$

então existem funções, conhecidas como parametrização da superfície, tais que

$$x = x(u, v)$$

$$y = y(u, v)$$

$$z = z(u, v)$$

$$w = w(u, v)$$

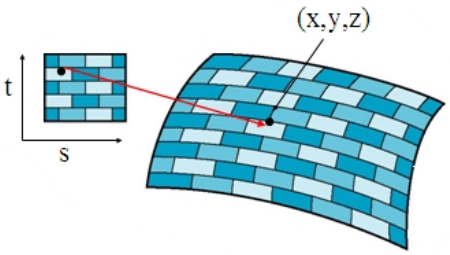
Uma das dificuldades que devemos encontrar, é que, embora estas funções existam conceitualmente, encontrá-las pode não ser possível na prática. Além disto, ainda queremos encontrar estas funções tais que sua inversa também seja permitida, ou seja, dado um ponto (x, y, z, w) sobre a superfície do objeto precisamos encontrar as coordenadas da textura correspondente a este ponto, ou seja

$$u = u(x, y, z, w)$$

$$v = v(x, y, z, w)$$

Embora este mapeamento seja simples para superfícies, tais como esferas, cilindros e planos, precisamos também encontrar a função de mapeamento que relacione um ponto (u, v) no domínio da parametrização da superfície, como um ponto (s, t) do espaço da textura, e em alguns casos, devemos conhecer a função inversa deste mapeamento, ou seja, a função que relaciona um ponto sobre a textura com um ponto sobre o domínio de parametrização da superfície.

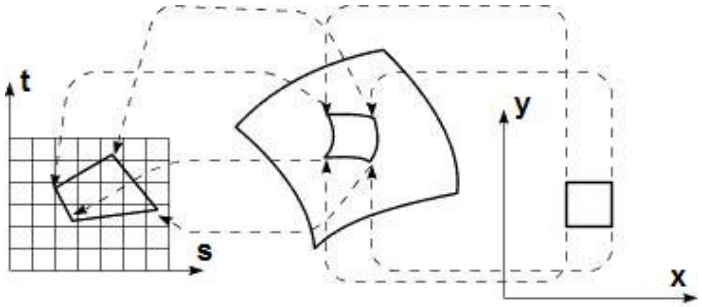
A estratégia prática mais utilizada, consiste em considerar o objeto a ser renderizado formado por uma malha de polígonos planos, em geral triângulos. Durante a fase de modelagem, coordenadas (s, t) do espaço de textura são associadas com os vértices do polígono sobre o qual será aplicada a textura. O problema desta idéia, é que a geometria de um objeto definida a partir de uma malha poligonal, só existe nos vértices de cada polígono. Como solução deste problema, existem dois algoritmos possíveis: o *inverse mapping* e o *forward mapping*.



forward mapping

O mapeamento inverso (*inverse mapping*) é um algoritmo orientado ao espaço de imagem e, para cada pixel encontramos através da função de mapeamento inverso o ponto correspondente no espaço de textura. Uma operação de filtragem integra a informação contida no ponto e

determina a cor para aquele pixel. Este algoritmo é vantajoso se o mapeamento de textura deve ser incorporado ao z-buffer. Um pixel quadrado produz um quadrilátero como pré-imagem.



inverse mapping

No *forward mapping* o algoritmo é orientado ao espaço de textura, onde um texel quadrado no espaço de textura produz um quadrilátero curvilinear no espaço de imagem e possui um problema em potencial que é a formação de "buracos" e sobreposições na imagem texturizada.

Apesar do *forward mapping* ser de fácil compreesão, na prática os algoritmos de mapeamento inverso são os preferidos e, em nosso trabalho, será a única estratégia utilizada para o mapeamento de objetos descritos a partir de malhas poligonais.

2.1 Mapeamento Inverso por Interpolação Bilinear

Para o mapeamento inverso, é conveniente considerar o mapeamento que procuramos, como uma transformação do espaço bidimensional (x, y), da imagem, para o espaço bidimensional (s, t), da textura. Esta é uma operação de *warpind* sobre a imagem e pode ser modelada como uma transformação projetiva racional, com a seguinte forma

$$x = \frac{as + bt + c}{gs + ht + i}$$

$$y = \frac{ds + et + f}{gs + ht + i}$$

Esta é uma transformação não-linear e, podemos representá-la matricialmente através do uso de coordenadas homogênea, assim

$$\begin{bmatrix} x' \\ y' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} s' \\ t' \\ q' \end{bmatrix}$$

onde

$$(x, y) = \left(\frac{x'}{w}, \frac{y'}{w}\right)$$

$$(s, t) = \left(\frac{s'}{q}, \frac{t'}{q}\right)$$

Esta é conhecida como transformação linear racional e sua inversa é dada por

$$\begin{bmatrix} s' \\ t' \\ q' \end{bmatrix} = \begin{bmatrix} A & B & C \\ D & E & F \\ G & H & I \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w \end{bmatrix}$$

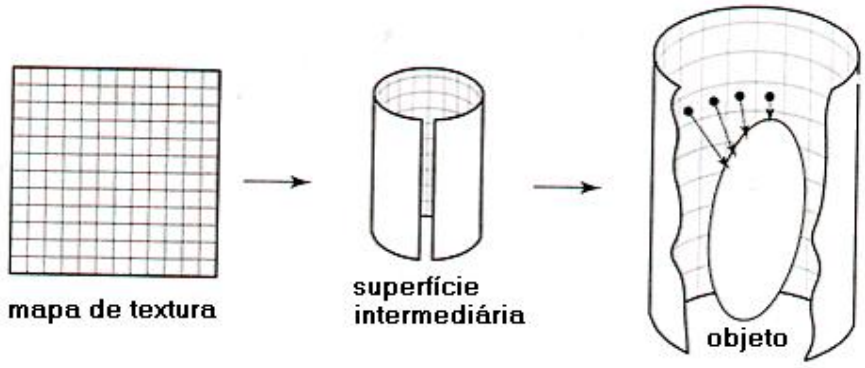
$$= \begin{bmatrix} ei - fh & ch - bi & bf - ce \\ fg - di & ai - cg & cd - af \\ ge - dh & bg - ah & ae - bd \end{bmatrix} \begin{bmatrix} x' \\ y' \\ w \end{bmatrix}$$

Observe agora que, na maioria das aplicações prática, devemos definir durante a fase de modelagem, uma correspondência entre pontos da textura e os vértices da malha poligonal. Portanto, se tivermos a correspondência para os quatro vértices de um quadrilátero, podemos determinar os coeficientes $(a, b, c, d, e, f, g, h, i)$ e com isto teremos a transformação inversa para qualquer ponto dentro daquele polígono.

2.2 Mapeamento Inverso usando Uma Superfície Intermediária

O método anterior de mapeamento inverso é, sem dúvida, a abordagem mais comum. O método que descreveremos agora, pode ser usado em aplicações onde não existe correspondência entre os vértices dos polígonos e pontos da textura. Pode ser usado também como um pré-processamento para determinação desta correspondência e em seguida usar o método anterior para a renderização final.

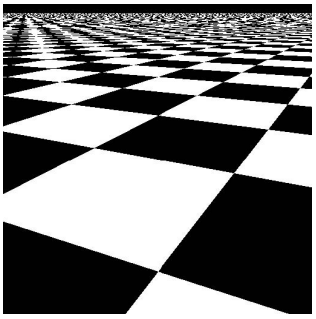
O mapeamento em dois estágios é uma técnica que supera o problema de parametrização da superfície em objetos de malha poligonal usando uma superfície intermediária de "fácil" parametrização, sobre a qual a textura é aplicada inicialmente. Esta técnica foi introduzida por Bier e Sloan[12] e pode ser usada também para implementar uma técnica que veremos mais a frente, o *environment mapping*, e é portanto uma técnica que unifica os processos de aplicação de textura e o *environment mapping*.



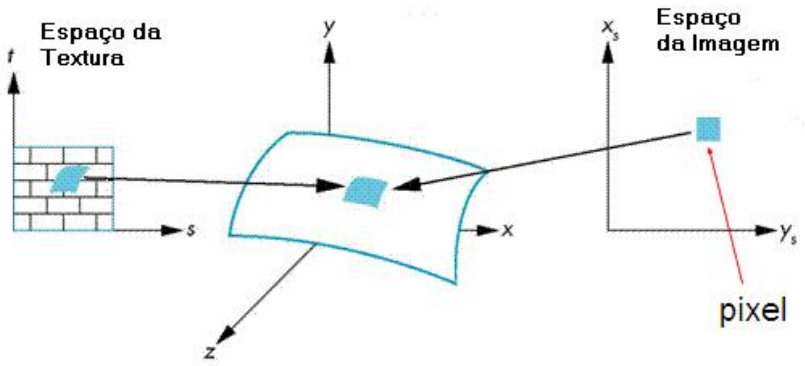
Este processo é conhecido como mapeamento em dois estágios por que a textura é mapeada sobre uma superfície intermediária, antes de ser mapeada sobre o objeto. A superfície intermediária em geral não é plana, mas possui uma função de mapeamento analítica e a textura é aplicada sobre esta superfície sem dificuldades. Por fim, encontrar a correspondência entre um ponto sobre a superfície intermediária e um ponto do objeto, torna-se um problema de encontrar uma transformação entre espaços 3d.

2.3 Aliasing

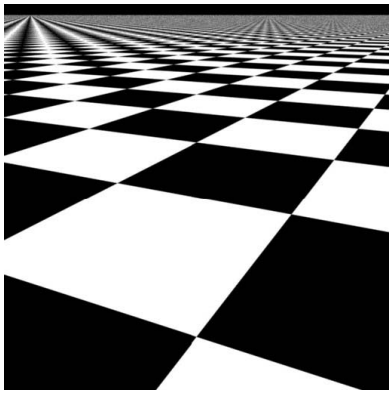
O *aliasing* é um sério problema no mapeamento de textura pois, as bordas e a projeção perspectiva podem causar padrões de alta freqüência no espaço da imagem. Afigura abaixo mostra os resultados de um mapa de textura aplicado sobre um plano. Neste exemplo, ao mapearmos um pixel do espaço de imagem, via mapeamento inverso, no espaço de textura, usamos o pixel mais próximo do resultado obtido, como escolha para renderizar o ponto na tela.



Catmull[3] foi o primeiro a perceber este problema, que ocorreu em seu trabalho sobre subdivisão de superfícies curvas. Como possível solução para o problema ele usou uma média dos vértices de um quadrilátero curvado, obtido como mapeamento inverso dos vértices do pixel (visto como um quadrado no espaço da imagem).



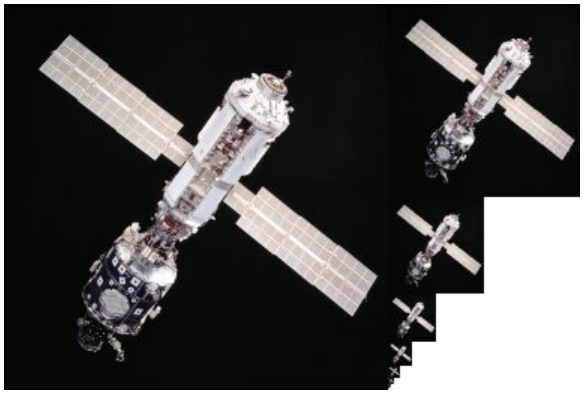
O resultado desta técnica pode visto na imagem abaixo:



Blinn e Nevell[2] usaram uma versão do filtro de Barlet para resolver este problema e Feibush et al.[13] estenderam essa idéia para um filtro mais elaborado. Infelizmente, estas técnicas são expansivas em termos computacionais.

2.3.1 Mipmapping

Williams [14] desenvolveu uma técnica de mapeamento de textura, chamada *mipmapping* (do Latim, **mip** significa *multum in porvo*, que em português quer dizer *muitas coisas num lugar pequeno*). Esta técnica gera muitas imagens de resolução decrescente a partir do mapa de textura original gerando a cor do pixels nas novas imagens como médias dos pixels correspondentes a ele na imagem original. Abaixo temos uma figura que exemplifica este procedimento



A imagem original e as imagens de resolução menor são armazenadas em múltiplas tabelas.

Assumindo que um pixel cobre uma área aproximadamente planar no espaço da imagem, então um pixel quadrado transforma-se num quadrilátero no espaço da textura. O tamanho aproximado da área do quadrilátero é dado em função das derivadas parciais da função de mapeamento inverso. Estas derivadas parciais são aproximadas usando diferenças finitas entre os pontos próximos do pixel de interesse sobre a textura.

Mipmapping padrão filtram apropriadamente apenas regiões quadradas. Assim, o quadrilátero sobre a textura será aproximado por um quadrado. Escolhendo um quadrado muito pequeno, levá-nos ao *aliasing*, muito grande levá-nos ao *blurring*. Heckbert [5] tomou como lado para o quadrado que aproxima o quadrilátero o valor

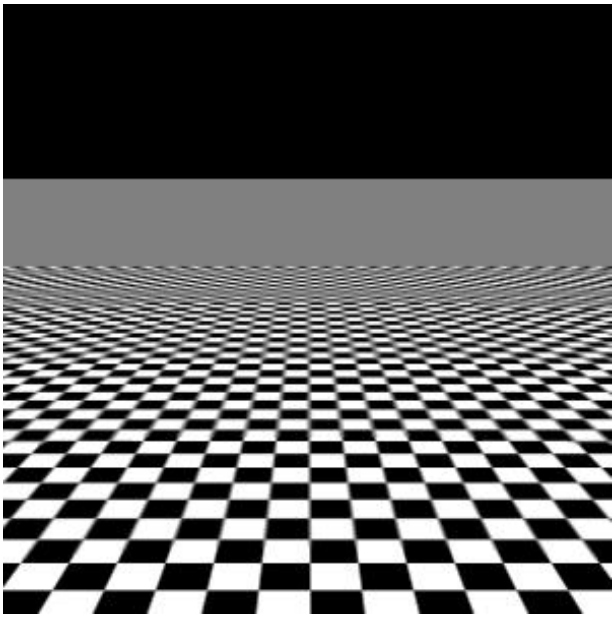
$$d = \max\left(\sqrt{u_x^2 + u_y^2}, \sqrt{v_x^2 + v_y^2}\right)$$

onde u e v são as funções de mapeamento inverso. Ou seja, ele escolheu lado do quadrilátero, de maior comprimento. A partir daí, a interpolação trilinear é usada para determinar a cor naquele pixel. Conseqüentemente, o *mipmapping* é algumas vezes chamado de interpolação trilinear. a memória necessária para produzir o mapa de textura usado no mipmapping é

$$1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + ... = \frac{4}{3}$$

da imagem original.

Esta é provavelmente a técnica mais usada para a filtragem de textura, com intenção de redução do aliasing, porque produz resultados aceitáveis, é rápida e requer pouco memória adicional.



3. Extensões do Conceito

3.1 Bump Mapping

Desenvolvida por Blinn [1], esta é uma forma bastante elegante de fazer com que uma superfície adquira uma aspecto rugoso ou ondulado sem a necessidade de modelar estas ondulações geometricamente. Para alcançar este objetivo, as normais da superfície são angularmente perturbadas de acordo com as informações contidas num mapa de textura, o *bump map*.

Em um ponto sobre a superfície, digamos $X(u, v)$, as derivadas parciais X_u e X_v definem dois vetores que estão no plano tangente à superfície, neste ponto. Assim, o produto vetorial destes vetores nos dão como resultado o vetor normal à superfície neste ponto, ou seja

$$n(u, v) = X_u \times X_v$$

Blinn definiu uma nova superfície, dando a ela uma aparência rugosa, inserindo uma função de perturbação $P(u, v)$, que atua modificando os vetores normais da superfície $X(u, v)$ com isto a nova superfície, criada por Blinn possui a seguinte parametrização

$$Q(u, v) = X(u, v) + P(u, v)\frac{\mathbf{n}}{\|\mathbf{n}\|}$$

Observe que, sendo Q definida desta forma, teremos

$$Q_u = X_u + P_u\frac{\mathbf{n}}{\|\mathbf{n}\|} + P\frac{\mathbf{n}_u}{\|\mathbf{n}\|}$$

$$Q_v = X_v + P_v\frac{\mathbf{n}}{\|\mathbf{n}\|} + P\frac{\mathbf{n}_v}{\|\mathbf{n}\|}$$

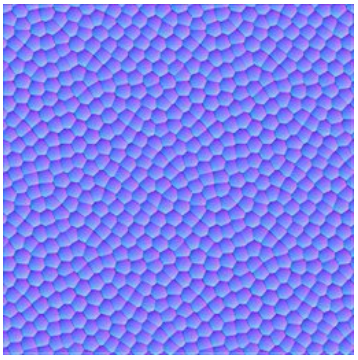
Como P é muito pequeno, os termos $P\dfrac{\mathbf{n}_u}{\|\mathbf{n}\|}$ e $P\dfrac{\mathbf{n}_v}{\|\mathbf{n}\|}$ podem ser negligenciados, de modo que o vetor normal desta nova superfície é

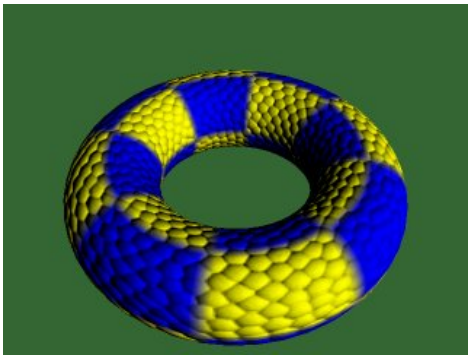
$$\begin{aligned}
\mathbf{n}'(u, v) &= Q_u \times Q_v \\
&= \left(X_u + P_u\frac{\mathbf{n}}{\|\mathbf{n}\|}\right) \times \left(X_v + P_v\frac{\mathbf{n}}{\|\mathbf{n}\|}\right) \\
&= X_u \times X_v + P_v\left(\frac{X_u \times \mathbf{n}}{\|\mathbf{n}\|}\right) + P_u\left(\frac{\mathbf{n} \times X_v}{\|\mathbf{n}\|}\right) + P_uP_v\left(\frac{\mathbf{n}}{\|\mathbf{n}\|} \times \frac{\mathbf{n}}{\|\mathbf{n}\|}\right) \\
&= \mathbf{n}(u, v) + P_v\left(\frac{X_u \times \mathbf{n}}{\|\mathbf{n}\|}\right) + P_u\left(\frac{\mathbf{n} \times X_v}{\|\mathbf{n}\|}\right)
\end{aligned}$$

onde o termo

$$P_v\left(\frac{X_u \times \mathbf{n}}{\|\mathbf{n}\|}\right) + P_u\left(\frac{\mathbf{n} \times X_v}{\|\mathbf{n}\|}\right)$$

representa o efeito da perturbação P sobre a normal da superfície X. O vetor n', após ser normalizado será então utilizado no modelo de iluminação para a renderização da nova superfície.



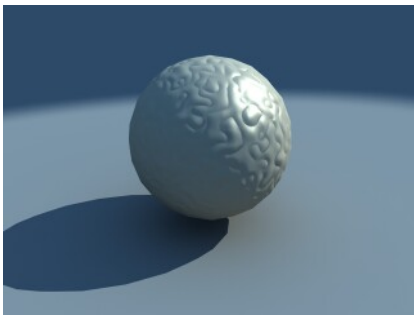


Quase qualquer função para as quais as derivadas parciais estejam definidas podem ser usadas como a função de perturbação P. Blinn usou em seus testes, desde uma função simples definida matematicamente para gerar o *bump map*. Para padrões que não podem ser representados matematicamente, a função de perturbação é representada em uma tabela bi-dimensional indexada pelos parâmetros u, v onde, valores intermediários são obtidos através de interpolação bilinear e as derivadas parciais P_u e P_v são determinadas usando diferenças finitas.

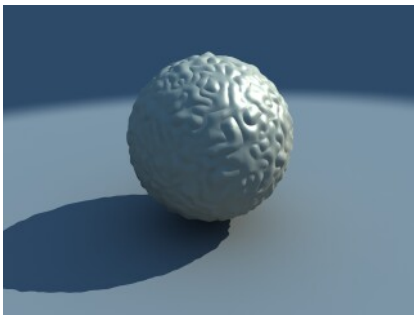
O efeito de rugosidade não é invariante com mudanças na escala do objeto de modo que, se o tamanho do objeto for escalado por um fator 2, então o comprimento do vetor normal será escalado por um fator 4, enquanto que o mapa de perturbação será escalado por um fator 2. Isto resulta na suavização do efeito da textura a medida que o objeto aumenta.

3.2 Displacement Mapping

Um problema com o *bump mapping* é que, o mesmo não lida com sombras ou silhueta do objeto. Estas limitações ocorrem por que, de fato a superfície não sofre alteração nenhuma em sua geometria.



bump mapping



displacement mapping

Se desejarmos mais realismo na cena, uma técnica alternativa é o *displacement mapping,* desenvolvida por Cook et al[6]. O *displacement mapping* altera a geometria do objeto usando o mapa de textura. Uma simplificação comum é usar o deslocamento na direção da normal.

Se $X(u, v)$ é um ponto sobre a superfície cujo vetor normal é dado por $\mathbf{n}(u, v)$, então este método gera uma nova superfície a partir do mapa de textura, da seguinte maneira

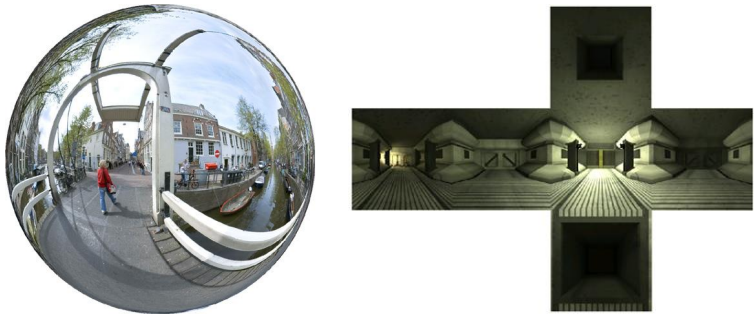
$$X'(u, v) = X(u, v) + f(X(u, v))\mathbf{n}$$

onde $f(X(u, v))$ é o valor no mapa de textura correspondente ao ponto $X(u, v)$.

3.3 Environment Mapping

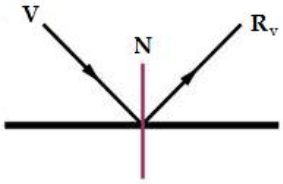
Também conhecida como *"reflection mapping"*, está é uma técnica desenvolvida para renderizar objetos reluzentes exibindo em sua superfície o ambiente ao seu redor. O *"environment mapping"* é uma técnica alternativa ao ray tracing, relativamente barata em termos computacionais e com bons resultados.

A idéia geral do método é: assumimos que temos um objeto reluzente, relativamente pequeno. Um espelho plano ou esférico, um bule feito de material reluzente, são exemplos. Obtemos então, a partir de uma fotografia ou imagem gerada por computador, uma visão do ambiente ao redor, como se o observador estivesse no centro do objeto. Usamos esta *"vista"* como mapa de textura.



Para renderizarmos um ponto sobre o objeto, usamos a posição do observador P_o, o ponto P e a normal $\mathbf{N}$ da superfície para calcular a direção do raio de visão refletido, $\mathbf{R}_v$, ou seja

$$\mathbf{R}_v = 2(\mathbf{N} \cdot \mathbf{V})\mathbf{N} - \mathbf{V}$$



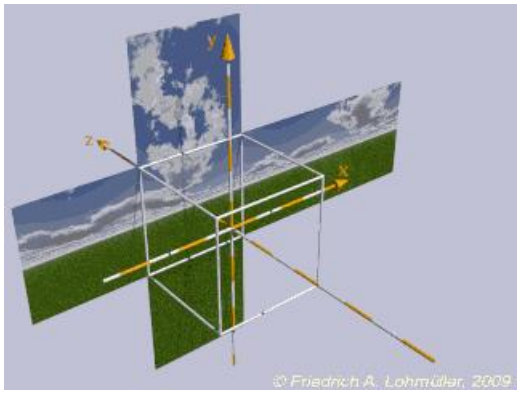
onde

$$\mathbf{V} = P_o - P$$

A partir do raio de visão refletido, $\mathbf{R}_v$, podemos determinar o ponto sobre o mapa de textura correspondente ao ponto P, e conseqüentemente a cor usada em sua renderização.

3.3.1 Cube Mapping

Como vimos anteriormente, o *environment mapping* é um processo em dois estágios, onde um deles é a geração do mapa de textura. O *"cube mapping"* está entre as técnicas mais populares de *environment mapping* por conta da facilidade na construção de seus mapas. O mapa utilizado, neste caso, não verdade são seis sub-mapas que juntos compões as faces de uma cubo. Um exemplo disto pode ser visto na figura abaixo.

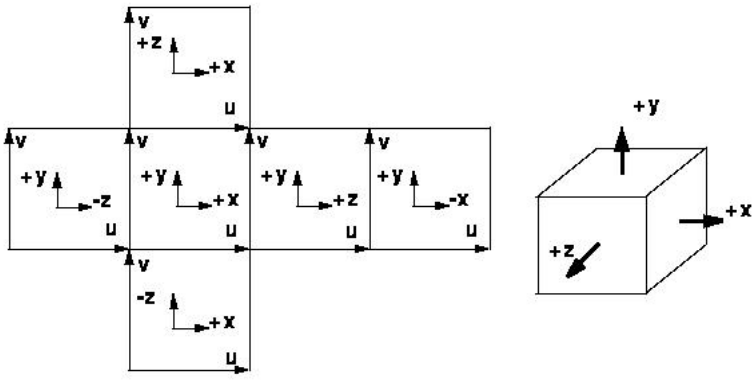


O ponto de vista é fixado no centro do objeto para receber o mapa do ambiente, e seis vistas são renderizadas. Considere, por exemplo um ponto de visão colocado no centro de uma sala vazia, as seis faces seriam as quatro paredes laterais, o solo e o teto da sala.

Um dos problemas do mapeamento cúbico é que, as raios de visão refletidos sobre os vértices ou arestas do cubo, compartilham mais de uma das faces localizadas no mapa de textura e alguma estratégia de decisão deve ser usada para que o mesmo não possua dois mapeamentos diferentes.

O algoritmo para a geração do mapa de textura pode ser enunciado da seguinte forma:

1. **Determinar a face interceptada** - Isto envolve cálculos simples de comparação entre o raio de visão refletido normalizado e as faces do cubo, que assume-se em geral ser unitário e centrado na origem.

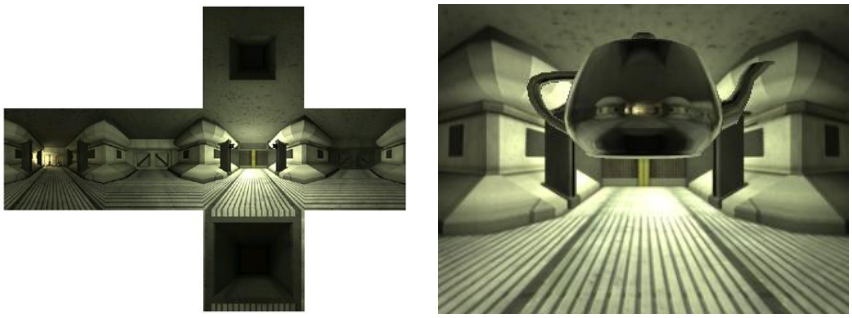


2. **Determinar as coordenadas** (u, v) - Por exemplo, um ponto (x, y, z) que intercepta a face cuja normal é o eixo z, pode ser mapeado em

$$u = x + 0.5$$

$$v = -z + 0.5$$

Abaixo temos um exemplo de uma mapa de textura cúbico e seu mapeamento sobre a superfície de um bule.



3.3.2 Sphere Mapping

O primeiro uso do "environment mapping" foi feito por Blinn e Newell [2] onde uma esfera, em vez de um cubo foi usada como base para seu método. O mapa do ambiente consistia de uma projeção latitude-longitude e o raio de visão refletido, $\mathbf{R}_v$ foi mapeado nas coordenadas (u, v) conforme as expressões abaixo:

$$u = \frac{1}{2}\left(1 + \frac{1}{\pi}\arctan\left(\frac{R_{vy}}{R_{vx}}\right)\right), -\pi < \arctan x < \pi$$

$$v = \frac{R_{vz} + 1}{2}$$

onde R_{vx}, R_{vy} e R_{vz} são as coordenadas do vetor $\mathbf{R}_v$.

O principal problema com esta técnica bastante simples, são as singularidades nos pólos. Na área polar, pequenas variações no raio de visão refletido ocasionam grandes variações nas coordenadas (u, v), pois

$$R_{vz} \to \pm 1 \Rightarrow R_{vx} \to 0 \, e \, R_{vy} \to 0 \Rightarrow \frac{R_{vy}}{R_{vx}} = indefinido$$

Do mesmo modo, quando $v \to 1$ ou $v \to 0$, o comportamento da coordenada u se torna instável, causando distúrbios visuais sobre a superfície. Este efeito pode ser amenizado modulando a resolução horizontal do mapa de textura através do $\operatorname{sen}\theta$, onde θ é o ângulo de elevação em coordenadas polares.

Abaixo temos um exemplo do uso desta técnica.

3.3.3 Multi-texturas

Embora tenhamos restringido a discussão à geometria e assumido que o objeto que é mapeado pelo ambiente possua uma superfície perfeitamente especular e o mapa de textura seja indexado usando-se apenas um raio de reflexão, é possível também admitir que a superfície do objeto possua outras propriedades diferentes da reflexão especular perfeita. Usando o modelo de iluminação de Phong, podemos considerar neste modelo, duas componentes, a componente difusa e a componente especular, e construir dois mapas. O mapa da componente difusa é indexado pelo vetor normal da superfície no ponto de interesse, enquanto que o mapa da componente especular é indexado pelo raio de visão refletido. A contribuição relativa de cada mapa é determinada pelos coeficientes de reflexão difuso e especular do mesmo modo que no modelo de Phong. Isto nos permite renderizar objetos como se eles fossem iluminados segundo o modelo de Phong mas, com a adição da reflexão do ambiente sobre a superfície do objeto, que pode ser borrado para simular uma superfície não suave.

A primeira menção a esta técnica foi feita por Miller e Hoffman[7]. Em sua abordagem, os mapas difuso e especular foram gerados processando o mapa do ambiente, de forma que este método pode ser visto como um algoritmo de duas etapas

- Gera-se a iluminação em um ponto em relação a cena, sem o objeto presente.
- Filtra-se o mapa para obter informações sobre a superfície do objeto.

Miller e Hoffman, geraram o mapa difuso a partir da seguinte definição

$$D(\mathbf{N}) = \frac{\sum_{\mathbf{L}} I(\mathbf{L}) \times Area(\mathbf{L}) \times f_d(\mathbf{N} \cdot \mathbf{L})}{4\pi}$$

onde

- **N** é o vetor normal à superfície no ponto de interesse;
- **L** direção incidente;
- $I(\mathbf{L})$ é o mapa do ambiente tomado em função de **L;**
- $Area(\mathbf{L})$ é a área sobre a superfície da esfera unitária associada a **L**;

- f_d é uma função de convolução difusa, dada por

$$f_d(x) = \begin{cases} k_d x, \, x > 0 \\ \\ 0, \quad x \leq 0 \end{cases}$$

- k_d é o coeficiente de reflexão difusa.

E, o mapa especular é definido como

$$S(\mathbf{R}_v) = \frac{\sum_{\mathbf{L}} I(\mathbf{L}) \times Area(\mathbf{L}) \times f_s(\mathbf{R}_v \cdot \mathbf{L})}{4\pi}$$

onde

- f_s é uma função de convolução especular, dada por

$$f_s(x) = \begin{cases} k_s x^n, \, x > 0 \\ \\ 0, \quad x \leq 0 \end{cases}$$

- k_s é o coeficiente de reflexão especular;

E, por fim, a intensidade de luz refletida num ponto da superfície é dada por

$$D(\mathbf{N}) + S(\mathbf{R}_v)$$

3.4 Shadow Mapping

A observação básica a ser feita sobre o mapa de sombras é que, se renderizarmos uma cena usando a localização da fonte de luz como posição do observador, as superfícies visíveis apareceriam iluminadas e, as superfícies não visíveis apareceriam escurecidas, ou seja, na sombra. Está idéia pode ser utilizada para determinar quando um ponto está ou não na sombra, [8].

Inicialmente, rasterizamos a cena do ponto de vista da fonte de luz. Neste caso, a matriz de projeção utilizada na cena será a mesma que normalmente se usa, alterando-se a *posição do observador*, que agora é a *localização da fonte de luz*, e a *normal* do plano de visão que agora será a *direção da fonte de luz*.

Assim, suponha que um dado ponto P, no referencial absoluto tenha coordenadas (x, y, z) e, ao ser projetado na imagem, sob a perspectiva da fonte de luz tenha coordenadas $(i, j, depth)$. Ao renderizarmos o mesmo ponto P, agora usando a maneira correta, ou seja, tomando a posição do observador como referência, suponha que o resultado obtido seja

$$(x', y', z')$$

Podemos comparar os valores das coordenadas z, nas duas projeções e inferir que: se os valores são iguais, então o ponto está iluminado, caso contrário está na sombra. Por conta dos erros

computacionais cometidos no processo, a maneira de comparar a igualdade deve ser tomada com certa margem de tolerância.

Outra observação importante a ser feita, é que, para que a comparação tenha sentido, as coordenadas $z's$ devem ser colocadas no mesmo referencial.

4. Textura 3D

Vimos anteriormente que existem muitas dificuldades associadas com o mapeamento de texturas bi-dimensionais em superfícies de objetos tri-dimensionais. As razões para isto são:

(1) O mapeamento de textura bi-dimensional baseado no sistema de coordenadas da superfície podem produzir grandes variações na compressão da textura que refletem uma variação correspondente na curvatura da superfície.

(2) A tentativa de se realizar um mapeamento contínuo sobre a superfície do objeto que possui uma topologia não trivial pode rapidamente se tornar muito estranho.

O mapeamento de texturas tri-dimensionais, circunscreve perfeitamente estes problemas pois a única informação necessária para determinar um ponto sobre uma textura é sua posição no espaço. Fazer a correspondência entre um objeto e um mapa de textura 3d, consiste apenas em determinar uma função em $\mathbb{R}^3$ que relacione os pontos do objeto com os pontos da textura. Um necessidade óbvia nesta idéia técnica é que, os mapas de texturas 3d devem ser gerados proceduralmente. Caso contrário a necessidade de memória se torna exorbitante. Também é inerente ao método, sua ineficiência, ao se construir, por exemplo, um mapa de textura cúbico por inteiro, quando necessitamos destes valores apenas na superfície do objeto em questão.

Dado um ponto (x, y, z) sobre a superfície de um objeto, sua cor é definida como sendo $T(x, y, z)$ onde T é o valor do mapa de textura, ou seja, simplesmente usamos a função identidade (possivelmente em conjunção com uma escala):

$$\begin{aligned} u &= x \\ v &= y \\ w &= z \end{aligned}$$

onde (u, v, w) são as coordenadas no mapa de textura.

Isto pode ser considerado análogo a atividade esculpir ou talhar um objeto a partir de um bloco de certo material. A cor do objeto é determinada pela interseção do objeto com o mapa de textura. Esta idéia foi simultaneamente desenvolvida por Perlin[9] e Peachey[10] onde o termo *"textura sólida"* foi cunhado.

A desvantagem desta técnica é que, embora ela elimine problemas de mapeamento, os padrões são limitados à qualquer coisa que você possa imaginar. Isto contrasta com os mapas de textura 2d. Nesta técnica, qualquer textura pode ser usada.

4.1 Ruído 3D

Uma classe popular de técnicas de texturização procedural, tendo em comum o fato de que todas usam uma função de ruído como primitiva básica de modelagem. Entre estas, a mais notável é uma que produz a simulação de turbulência, podendo produzir uma surpreendente variedade de texturas com efeitos realísticos e de aparência natural. Nesta seção, nos concentraremos nos detalhes envolvidos no algoritmo de geração da primitiva básica: **o ruído 3d**.

Perlin [9] foi o primeiro a sugerir esta aplicação para o ruído, definindo uma função chamada *noise()* que toma uma posição tri-dimensional como entrada e retorna um valor escalar. Isto é chamado de síntese de modelo dirigido - calculamos a função de ruído apenas no ponto de interesse. Idealmente a função deve possuir as três propriedades que seguem:

(i). Invariância estatística sob rotações;

(ii). Invariância estatística sob translações;

(iii). Um limite estreito na freqüência, na faixa passante.

As duas primeiras condições asseguram que a função de ruído é controlável pois, não importa como possamos mover ou girar a função de ruído no espaço, sua aparência geral é garantida ser a mesma. a terceira condição no permite fazer a amostragem na função de ruído sem *aliasing*. Apesar de que uma função de ruído insuficientemente amostrada possa não produzir defeitos notáveis em imagens estáticas, se usado em animações, o ruído amostrado incorretamente produzirá um efeito trêmulo ou de borbulha.

O método de Perlin para geração do ruído, consiste na geração de um gride de inteiros, ou seja, um conjunto no espaço, situado em posições (i, j, k) onde i, j e k são inteiros. Cada ponto do gride possui um valor randômico associado a ele. Isto pode ser feito simplesmente usando uma tabela de referência, ou como Perlin sugere, através de uma função de hash para economizar espaço. O valor da função de ruído num ponto do gride será então este valor randômico associado a ele. Para pontos que não estão sobre o gride a função de ruído pode ser obtida através de interpolação bilinear a partir dos pontos do gride em torno do ponto de interesse. Se usarmos este método para gerarmos uma função de ruído sólida $T(u, v, w)$, está irá apresentar uma tendência direcional em relação aos eixos coordenados. Isto pode ser amenizado usando interpolação bicúbica mas é uma opção de longe mais expansiva em relação ao custo computacional e as tendências direcionais ainda continuarão visíveis. Um alternativa para os métodos de geração de ruído, que eliminam este problema pode ser encontrada em Lewis[11], entretanto é importante ter em mente que a função de ruído é amostrada apenas sobre a superfície do objeto e isto em si representa uma transformação sobre o ruído que pode ser o suficiente para eliminar as tendências direcionais do ruído.

4.1.1 Simulação de Turbulência

Um parte simples do ruído pode ser colocada em uso para simular um número considerável de efeitos. De longe, a mais versátil destas aplicações é uso da *função de turbulência*, como foi definida por Perlin, que toma uma posição no espaço (x, y, z) e retorna um valor escalar que representa a turbulência neste ponto. Isto escrito na forma de um progressão geométrica e numa versão unidimensional, poderia ser definido da seguinte maneira:

$$\text{turbulence}(\mathbf{x}) = \sum_{i=0}^{k} \left| \frac{\text{noise}(2^i \mathbf{x})}{2^i} \right|$$

este somatório é truncando em k que é o menor inteiro tal que

$$\frac{1}{2^{k+1}} < \text{tamanho do pixel}$$

Este truncamento limita a banda passante da função assegurando o correto anti-aliasing.

A função de turbulência sozinha, representa apenas a metade do processo. Renderizar a função de turbulência diretamente resulta em um padrão homogêneo que não pode ser descrito como naturalístico. Isto ocorre por que, na maioria das texturas que ocorrem naturalmente, contém alguma propriedade estrutural não homogênea e assim, não podem ser simuladas como turbulência sozinhas. O mármore por exemplo, que possui veios de cores percorrendo através dele, facilmente distinguíveis, que sofreu turbulência durante a era geológica, antes de se solidificar. Diante deste fato, podemos identificar dois estágios distintos no processo de simular turbulência:

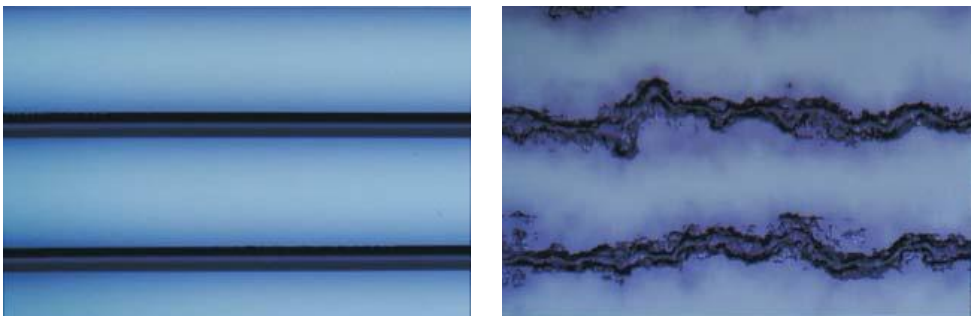
(1) Representação das propriedades estruturais básicas da textura, através de alguma função. Tipicamente esta função é contínua e contém variações significantes em suas derivadas de primeira ordem.

(2) Adição de detalhes de segunda e terceira ordem, usando uma turbulência para perturbar os parâmetros da função obtida no estágio anterior.

Um exemplo clássico, que foi primeiro descrito por Perlin, é a turbulência aplicada a uma onda senoidal para se obter a aparência do mármore. Sem a perturbação, os veios de cores percorrendo o mármore são dados por uma onda senoidal percorrendo um mapa de cores. Para uma senóide percorrendo o eixo x escrevemos:

$$\text{marble}(x) = \text{marble\_color}(\sin(x))$$

O mapa de cores marble_color() mapeia um escalar numa intensidade. A figura abaixo exibe uma fatia de mármore e sua respectiva renderização usando a expressão acima:



A figura seguinte, mostra o exemplo de um bule renderizado usando-se esta técnica.

O uso de funções de turbulência não se restringe à modulação de cores de um objeto. Qualquer parâmetro que afete a aparência de um objeto pode ser alterado através de uma função de turbulência. Turbulência pode direcionar a transparência de objetos como nuvens, por exemplo.

4.2 Animação

A função de turbulência pode ser definida tendo o tempo como um parâmetro, além dos parâmetros espaciais. Colocando desta forma, os pontos do gride, agora deverão ser especificados a partir de quatro índices (i, j, k, l), permitindo que possamos nos referir ao ruído na forma $(\mathbf{x}, t)$ onde $\mathbf{x}$ representa a posição no espaço e t o tempo.

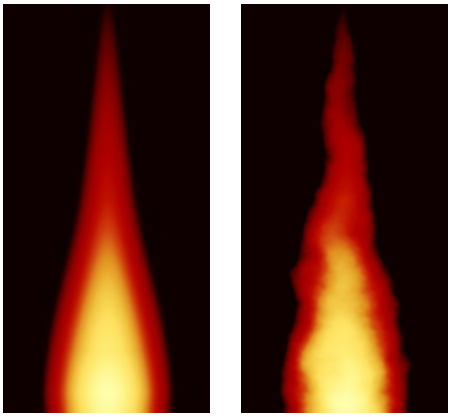
Assim, por exemplo, se quisermos simular o fogo, a primeira coisa que devemos fazer, é tentar representar sua funcionalidade básica, ou seja a "forma da chama". A modelagem adota neste texto, foi simplesmente aquela, segundo o [4], que após a experimentação, gerou os melhores resultados.

A região de uma chama é definida no plano xy pelo retângulo $-b \leq x \leq b$, $0 \leq y \leq h$. Dentro desta região a cor é dada por

$$\text{flame}(x, y) = \left(1 - \frac{y}{h}\right) \text{flame\_colour}\left(\left|\frac{x}{b}\right|\right)$$

A função flame_colour (x) consiste em três splines de cores separadas que mapeiam um valor escalar x num vetor de cor. Cada uma destas splines são responsáveis pelas componentes R, G e B, respectivamente, do vetor de cor obtido.

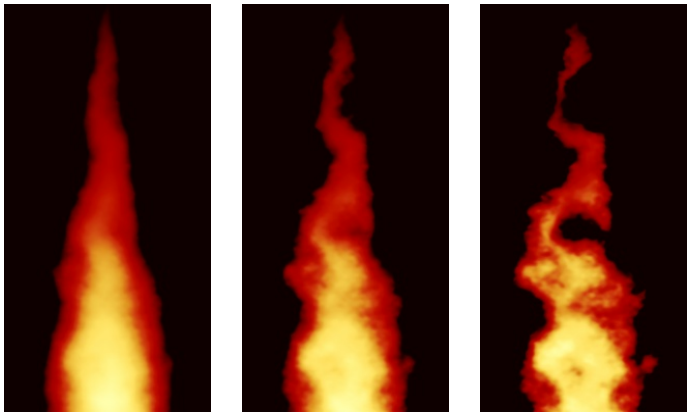
A figura abaixo mostra a chama segunda a modelagem escolhida e, a mesma chama após a aplicação de uma função de turbulência



Para gerar uma animação à chama, introduzimos na função anterior a função de turbulência, ou seja

$$\text{flame}(x,y) = \left(1 - \frac{y}{h}\right) \text{flame\_colour}\left(\left|\frac{x}{b}\right|\right) + \text{turbulence}(x,t)$$

e geramos várias imagens sucessivas variando o parâmetro t da função. Abaixo segue o resultado obtido.



4.3 Mapas de Iluminação 3D

Em princípio, não há nenhuma razão pela qual não se possa criar mapas de iluminação 3d. A restrição prática é a vasta necessidade de memória que isto pode demandar. Ao supormos que isto seja possível, devemos ter um método de guardar a luz refletida em cada ponto da cena. Para isto usamos qualquer método de renderização independente da posição do observador e determinamos a intensidade de luz num (x, y, z) do objeto e guardamos esta informação em $T(x, y, z)$ no nosso mapa de textura. é interessante agora, comparar esta abordagem, com aquela feita anteriormente, via *"environment mapping"*, usando mapas de textura 2d.

Com o *"environment mapping"* salvamos todas a iluminação vinda de um ponto do objeto, num mapa bi-dimensional que foi indexado segunda a direção do raio de luz naquele ponto. Um vetor de vista refletido é então utilizado para recuperar a direção da luz refletida para observador. Isto é feito, normalmente para superfícies perfeitamente especulares e resultam em efeitos rápidos e independentes da posição do observador.

Com mapas de luz bi-dimensionais podemos guardar a luz refletida para cada superfície na cena em um conjunto de mapas bi-dimensionais. A indexação destes mapas de luz, durante a fase de rendering, depende do método que foi usado para fazer a amostragem dos objetos no espaço tri-dimensional. Desta forma, conseguimos obter uma iluminação não-dinâmica independente da posição do observador.

Por fim, com mapas de luz 3d, armazenamos a luz refletida em um ponto numa estrutura 3d que representa o espaço do objeto.

5. Conclusão

Neste texto procuramos realizar uma pequena introdução sobre a área da computação gráfica conhecida como *texture mapping*. Vimos que o tema além de muito interessante é muito vasto e versátil. O *texture mapping* nasceu da simples intenção de simular a aplicação de adesivo sobre a superfície de um objeto tridimensional e a partir daí várias extensões da idéia foram sendo criadas: o *bump mapping* para a simulação de rugosidade na superfície de um objeto, o *environment mapping* para simular objetos com superfícies altamente especulares, o *shadow mapping*, para a criação de sombras na cena, e etc. .

Em muitos casos, o *texture mapping* se apresenta como uma boa alternativa, apresentando bons resultados e com custo computacional relativamente barato. Em quase todos eles, existe a possibilidade de execução em tempo real, como é o caso do *shadow mapping* para a criação de jogos.

Esta técnica apresenta como problema, o *aliasing* que surge devido ao mapeamento inverso. Vimos que o *mipmapping* pode ser uma boa solução para o problema. Além do mipmapping existem outras técnicas que não pudemos abordar aqui.

Em sua extensão 3d, o texture mapping, passa a exigir uma quantidade de memória muito grande para se guardar as texturas, fazendo com que esta idéia seja realmente útil para o caso de texturas procedurais, como é o caso das funções de ruído. Com este tipo funções, é possível simular a turbulência e com esta criar efeitos como o do mármore e animação de chamas.

Existem muitas outras áreas, como o Volume Rendering por exemplo, que podem se beneficiar do texture mappping.

Referências

[1] **Blinn, J. F.**, *Simulation of Wrinkled Surfaces*, Computer Graphics, (Proc. Siggraph), Vol. 12, No. 3, August 1978, pp. 286-292.

[2] **Blinn, J. F. and Newell, M. E.**, *Texture and Reflection in Computer Generated Images*. Comm. ACM, 19 (10), 1976, pp. 362-367.

[3] **Catmull, E**, *Subdivision Algorithm for the Display of Curved Surfaces*, PhD Thesis, University of Utah, 1974.

[4] **Watt and Watt**, *Advanced Animation and Rendering Techniques*, Addison-Wesley, 1992, pp. 199-201;

[5] **P. S. Heckbert**, *Survey of Texture Mapping*, IEEE Computer Graphics and Applications, Nov. pp. 56-67. 1986

[6] **Cook, R. L., Carpenter, L., and Catmull**, **E.,** *The Reyes Image Rendering Architecture*, Comput Graph, Vol. 21, pp. 95-102, 1987 (SIGGRAPH 87).

[7] **Miller G. S. and Hoffman, C. R.**, *Illumination and Reflection Maps: Simulated Objects in Simulaed and Real Environments*. SIGGRAPH'84, *Course Notes*, July, 1984.

[8] **Williams L.**, *Casting Curved Shadows on Curved Surfaces*. Computer Graphics, 12(3), pp. 270-274, 1978.

[9] **Perlin, K.**, *An Image Synthesizer*. Computer Graphics, 19(3), pp. 287-296, 1985.

[10] **Peachey, D. R.**, *Solid Texturing of Complex Surfaces*. Computer Graphics, 19(3), pp. 279-286, 1985.

[11] **Lewis, J. P.**, *Algorithms for Solid Noise Synthesis*. Computer Graphics, 23(3), pp. 263-270. 1989.

[12] **Bier, E. A. and Sloan, K. R.**, *Two-part Texture Mapping*, IEEE Computer Graphics and Applications, 6(9), 40-53, 1986.

[13] Feibush, E. A., Levoy, M. and Cock, R. L., Synthetic Texturing Using Digital Filters. Comput. Graph., SIGGRAPH'80 ,Vol. 14, pp. 294-301, 1980.

[14] **Williams, L.**, *Pyramidal Parametrics*. Comput. Graph, SIGGRAPH'83, Vol. 17, pp. 1-11, 1983.