# Aplicação de Textura e Rugosidade Procedurais para Modelagem de Sujeira e Desgaste em Superfícies

Bianca F. D. Fragoso
Departamento de Informática
PUC-Rio

Waldemar Celes (Orientador)
Departamento de Informática
PUC-Rio

***Resumo*—Neste trabalho, são discutidas algumas técnicas de criação de imperfeições de superfícies, com o objetivo de deixá-las mais realistas. Para criação de imperfeições, serão utilizadas técnicas de geração de texturas e mapas de rugosidades (bumps) procedurais, usando o ruído de Perlin como principal ferramenta. O ruído é usado para modelar sujeiras e corrosões. Experimentos computacionais demonstram a efetividade da técnica.**

***Abstract*—This work discusses some techniques for creating surface imperfections in order to make them more realistic. To create imperfections, techniques for generating procedural textures and procedural bumps will be used, using Perlin noise as the main tool. Noise is used to model dirt and corrosion. Computational experiments demonstrate the effectiveness of the technique.**

## I. Introdução

A busca por metodologias para aperfeiçoar o realismo das imagens sintéticas, no sentido de melhorar a aparência de superfícies, tem sido um dos grandes desafios da Computação Gráfica [1]. Podemos dizer que a pesquisa nesse contexto pode ser classificada em duas vertentes: sombreamento (shading) e texturização (texturing). De maneira bem simplificada, sombreamento é o processo de calcular a cor de um pixel e texturização é o método de adicionar de maneira eficiente detalhes visuais ricos a imagens sintéticas.

Neste trabalho, estaremos interessados em questões relacionadas à aplicação de textura. Os métodos utilizados para criação e mapeamento de texturas podem ser classificados em procedurais e não-procedurais. Uma textura não-procedural é criada através de dados armazenados (imagens), enquanto uma textura procedural é implementada usando equações e descrições matemáticas implementadas por código. Uma das características mais importante das técnicas procedurais é a abstração, no sentido de que ao invés de especificar explicitamente e armazenar todos os detalhes complexos de uma cena (ou sequência), nós os abstraímos em um algoritmo. Como ganhos da abordagem abstrata das técnicas procedurais podemos citar, por exemplo: (1) controle paramétrico: permite atribuir a um parâmetro significado conceitual, no sentido de que através de um parâmetro é possível, por exemplo, controlar o nível de rugosidade de uma montanha; (2) economia de armazenagem: isso acontece devido ao fato de não haver necessidade dos detalhes serem especificados à priori, ficando implícito no algoritmo; (3) multirresolução: permite criar modelos de multiresolução que podem ser avaliados dinamicamente durante a renderização. Este trabalho tem como foco uma breve exploração de algoritmos procedurais, utilizando em especial o ruído desenvolvido por Perlin [2] . Serão aplicados proceduralmente tanto textura quanto rugosidade em superfícies e uma análise do processo de desenvolvimento será feita.

O trabalho está estruturado da seguinte forma: na segunda seção, um pouco da história e da teoria de construção do ruído são apresentados. Na terceira seção, alguns testes iniciais usando o ruído na construção de textura de sujeira são apresentados. Na quarta seção, é apresentado o método de construção do mapa de rugosidade para simulação de imperfeições como corrosões e ferrugens. Na quinta seção, são apresentados os principais resultados comparando-os com imagens reais. Finalmente, na sexta seção, é apresentada uma conclusão de todo o estudo e implementação.

## II. Ruído de Perlin

### A. História

Trabalhos sobre textura procedural remontam aos anos 70. No entanto, é em 1985 que surge um dos trabalhos mais importantes sobre o assunto, onde K. Perlin [2] estabeleceu as bases para uma das classes mais populares de textura procedural em uso atualmente, baseada essencialmente em ruído (uma abordagem estocástica). Essa técnica é também conhecida na literatura como ruído procedural (ver, [2] e [3]). Esse ruído é uma das ferramentas mais bem sucedidas usadas para gerar detalhes com aleatoriedade. Desde a primeira imagem do vaso de mármore, apresentada por K. Perlin, o ruído de Perlin tem sido amplamente utilizado tanto na pesquisa quanto na indústria. Uma das vantagens do ruído é ser capaz de aplicar texturas em objetos sem projetar esses objetos no 2D (texturas tradicionais). A ideia de Perlin consistia em usar as próprias coordenadas do objeto como coordenadas de textura dentro de um volume com textura. Antes de K. Perlin, vários outros autores propuseram o uso de textura procedural. No entanto, todos eles usavam funções que variavam apenas no espaço 2D. Perlin estendeu para funções do espaço 3D chamadas "funções espaciais". Todas essas "funções espaciais" podem

ser entendidas como representando um material sólido. Se avaliarmos essa função nos pontos da própria superfície do objeto, obteremos a textura da superfície, como se estivéssemos esculpindo o objeto com esse material sólido [2].

### B. Construção do ruído

Existem duas partes principais para se obter o ruído de Perlin. A primeira é a geração de um número aleatório, e a segunda é a interpolação entre números aleatórios. Na geração do número aleatório há uma forte restrição como apontado por Tatarinov [4]. Diferentemente de uma função de geração de números totalmente aleatórios, a função utilizada na construção do ruído deve retornar sempre o mesmo valor para a mesma coordenada. A segunda parte é a interpolação de números aleatórios, que é importante pois resulta em valores contínuos e faz com que o ruído final fique com transições mais suaves do que se não houvesse interpolação. Neste trabalho, uma função que gera o ruído de Perlin [5] foi utilizada. No entanto, apesar de já parecer mais natural do que um número aleatório qualquer, o ruído de Perlin não apresenta algumas das irregularidades que possa se esperar da natureza. Por exemplo, um terreno possui características grandes e amplas, como montanhas, características menores, como montes e depressões, ainda menores, como pedras e rochas grandes, e muito pequenas, como seixos e pequenas diferenças no terreno. A solução para lidar com todos esses detalhes é simples. Várias funções de ruído com frequências e amplitudes variadas são criadas. Essas funções são chamadas de oitavas. Oitavas são muito exploradas quando se usa o ruído. Por isso, alguns parâmetros delas são definidos formalmente e podem ser modificados, alterando o resultado final da textura gerada:

- Frequência: Refere-se ao período em que os dados são amostrados. Aplicamos a frequência na hora de passar o parâmetro na função de ruído (multiplicamos ela pelas coordenadas).
- Amplitude: Refere-se ao intervalo no qual o resultado pode estar. Aplicamos a amplitude, multiplicando-a pelo resultado retornado pela função de ruído. Quanto maior é a amplitude, maior é a influência que alguma oitava terá no resultado final.
- Persistência: É o parâmetro que define o quanto cada oitava contribuirá para o ruído final, pois ajusta a amplitude. Uma configuração mais padrão seria uma persistência menor que 1.0 que diminui o efeito de oitavas de maior frequência.
- Lacunaridade: Número que determina quantos detalhes são adicionados ou removidos em cada oitava (ajusta a frequência).

A seguir a equação de oitavas, com *noise* sendo a função de ruído de Perlin, $f_0$ a frequencia inicial, $a_0$ a amplitude inicial, $l$ a lacunaridade, $p$ a persistência e *pos* a posição do vértice em questão.

$$Octave_i(f_0, a_0, l, p, pos) = noise(f_0 \cdot l^{i-1} \cdot pos) \cdot a_0 \cdot p^{i-1}$$

## III. Testes iniciais de criação de texturas para simulação de sujeiras

Antes dos testes de geração de rugosidade, foram feitos testes de aplicação de texturas em malhas. A ideia foi tentar gerar uma aparência de sujo nessas malhas. Para isso, as funções de soma de oitavas e turbulência foram utilizadas. A função de turbulência é bem conhecida na literatura e está presente no artigo de 1985 de Perlin [8]. Ela é usada como base de muitas outras funções que usam o ruído.

A função de soma de oitavas gera um aspecto de sujeira menos pesada, enquanto a de turbulência gera um aspecto de sujeira pesada como pode ser observado na Figura 1. Essas funções usam a função de ruído e retornam um número real $a$ para cada coordenada do vértice da malha. Esse valor $a$ é usado como interpolador das cores do objeto e da sujeira, como mostrado na equação abaixo, gerando o efeito desejado.

$$finalColor = objectColor * (1 - a) + dirtColor * a$$

Figura 1. (a) Brinco limpo (b) Brinco com sujeira (turbulência) (c) Brinco com sujeira (soma de oitavas).

## IV. Aplicação de rugosidade para simulação de imperfeições

O objetivo da aplicação de mapa de rugosidades procedural em superfícies é gerar um mapa de normal que se assemelhasse a corrosões, desgastes, oxidações, arranhões etc.

### A. Geração de um mapa de normal procedural

O cálculo de geração do mapa de normal foi inteiramente feito no fragment shader. Para gerar esse mapa, primeiramente foram utilizadas as coordenadas de tela da malha, ou seja, as coordenadas já multiplicadas pela matriz de modelagem, visualização e projeção (mvp), vindas do vertex shader. Os pontos vizinhos de cada vértice da malha, foram calculados usando a função built-in de derivadas do GLSL. Essas coordenadas estão no plano de projeção. Para que se tenha como resultado um mapa de normal com normais perturbadas, a coordenada z que foi atribuída a todas essas componentes foi o resultado de uma função que retorna um número real, calculado por uma função

que utiliza o ruído de Perlin. Com a coordenada z de cada vizinho, temos como resultado um campo de elevação, como mostrado na Figura 2 (a) (ver [6]). Dessa forma, a normal do mapa de normal pode ser calculada. Criam-se dois vetores utilizando as coordenadas dos pontos vizinhos. Com o produto vetorial entre eles, é calculado um vetor que equivale à normal naquele ponto. Essa normal calculada está no espaço tangente e precisamos passá-la para o espaço do olho que é onde os cálculos estão sendo feitos, e para isso usamos a matriz TBN (tangente, binomial e normal). Essa matriz leva uma coordenada que está em um espaço (nesse caso no espaço do olho) para o espaço tangente. Por isso, para levar a normal calculada do espaço tangente para o espaço do olho, é preciso inverter a matriz TBN. Como trata-se de uma matriz ortonormal, basta acessarmos a transposta. Transformando a normal do espaço tangente pela inversa da matriz TBN temos como resultado uma normal no espaço do olho. Essa normal é utilizada nos cálculos seguintes de iluminação e é ela a responsável pelo efeito de rugosidade dos resultados.

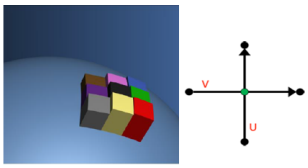


Figura 2. (a) Campo de elevações (b) Vetores compostos por texels vizinhos.

### *B. Funções para cálculo de coordenada z e seus resultados*

Como já dito anteriormente, uma função é utilizada para retornar uma coordenada z para cada vizinho do ponto, e assim construir um mapa de normais. Essa função recebe os pontos em coordenadas de mundo. A função de ruído de Perlin é chamada passando como parâmetros as coordenadas 3D do ponto e devolve um número real. No código, foram criadas várias funções desse tipo para testar os diversos efeitos de utilização desse valor de ruído. Em um primeiro momento, foram testadas funções clássicas, como a função de turbulência. Nesse caso, o ruído era calculado, as oitavas eram construídas e o módulo delas eram somados. No entanto, diferentemente da utilização da função de turbulência para a cor, o resultado não ficava tão natural, pelo contrário, ficava um resultado muito poluído e com detalhes demais . Isso acontece porque em todos os pontos da malha a normal é perturbada causando uma impressão de diferentes elevações em todos os pontos. Por isso, uma alternativa seria filtrar determinados valores da função de turbulência. Para que isso fosse feito, a função de *min*, que retorna o mínimo entre dois números, foi utilizada. Então, a coordenada z foi calculada utilizando a seguinte fórmula:

$$z = min(num, turbulence)$$

Esse filtro fazia com que houvesse uma menor variação entre as coordenadas z dos vizinhos. Isso acontecia porque z só assumia o valor da função de turbulência se esse estivesse abaixo de um determinado valor. Isso resultou em campos de elevações com alturas não muito variadas com relação ao anterior que usava diretamente a função de turbulência.

Foram feitos testes de números para serem utilizados juntamente com o resultado da função de turbulência na função de *min*, até que o resultado desejado fosse atingido. Em seguida foram utilizados valores menores como forma de filtrar ainda mais a função de turbulência. Quanto mais a função era restringida, ou seja, quanto mais os valores usados na função *min* junto com os valores de turbulência eram diminuídos, mais delicadas ficavam as imperfeições e em menor número. Alguns resultados usando a função de turbulência para cálculo da coordenada z são mostrados na Figura 3 .

$$turbulence = \sum_{i=1}^{n} |Octave_i|$$

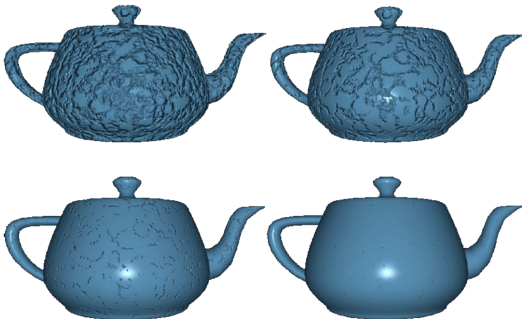

Figura 3. Usando função de turbulência na função de *min* para cálculo de z. Da esquerda para direita valor usado na função de *min* vai diminuindo.

A mesma estratégia de antes foi usada, dessa vez com a função de *min* sendo usada como forma de restringir a função de soma de oitavas. Nesse caso, as imperfeições se assemelhavam a corrosões, como pode-se perceber na Figura 4.

$$sumOctaves = \sum_{i=1}^{n} Octave_i$$

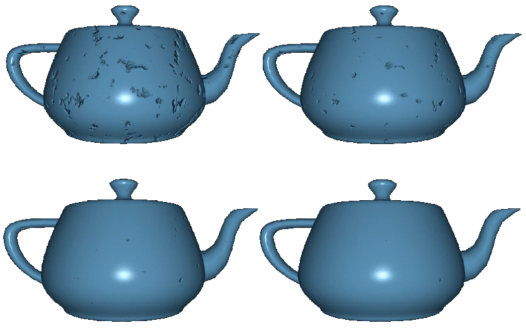

Figura 4. Usando função de soma de oitavas na função de *min* para cálculo de z. Da esquerda para direita valor usado na função de *min* vai diminuindo.

Uma função comumente usada na criação de veios de mármore também foi testada para geração de um campo de elevações.

Essa função gerou imperfeições com uma aparência de arranhões ou trincas. No entanto, novamente foi utilizada a função de mínimo pois se a função de mármore fosse diretamente aplicada o resultado gerado ficava com muitas imperfeições. A função de min deixou os resultados mais refinados como mostrado na Figura 5.

$$marble = \left|\sin\left(\frac{x}{distance}\right)\right| + sumOctaves\left(\frac{pos}{distance}\right)$$

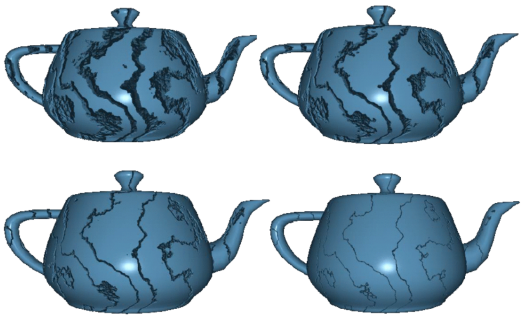

Figura 5. Usando função de mármore na função de *min* para cálculo de z. Da esquerda para direita valor usado na função de *min* vai diminuindo.

## V. Aplicação em modelos CAD

Nesta seção, apresentamos a aplicação da técnica para modelar desgastes de superfícies em modelos CAD de partes de uma plataforma de petróleo. Alguns dos resultados são apresentados nas figuras 6, 7 e 8.

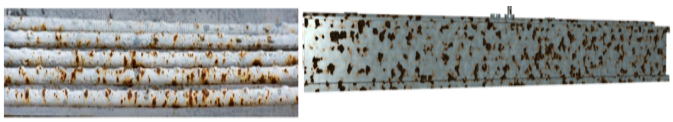

Figura 6. (a) Canos sujos e enferrujados de plataforma. (b) Sujeira e rugosidade gerados pelo algoritmo.

Figura 7. Comparação entre escada real de uma plataforma (esquerda) e malha de escada com rugosidade e com cor (direita).

Figura 8. Comparação entre viga de plataforma (esquerda) e malha com rugosidade e cor (direita).

Modelar desgaste em estruturas de modelos CAD é importante para realçar o realismo da visualização, permitindo seu uso em treinamento de análise de desgaste e serviços de manutenção.

## VI. Conclusão

Neste trabalho, pesquisas e testes foram feitos para se entender um pouco melhor os parâmetros do ruído de Perlin. O foco principal do trabalho foi a criação de imperfeições como sujeira, corrosões, oxidações etc utilizando os conceitos citados acima. Nessa parte, texturas e rugosidades foram criadas proceduralmente e testadas com diferentes funções utilizando diversos parâmetros diferentes. Ficou evidente que a utilização do ruído por funções ainda é algo bem artesanal, no sentido de que para se atingir um resultado desejado, muitas vezes é preciso fazer ajustes finos, como multiplicações e divisões por números. A aleatoriedade do ruído não é algo fácil de ser controlado. Apesar disso, alguns padrões de comportamento são percebidos e podem ser utilizados.

## Referências


[1] A. Lagae, S. Lefebvre, R. Cook, T. DeRose, G. Drettakis, D. S. Ebert, J. P. Lewis, K. Perlin, and M. Zwicker, "A survey of procedural noise functions," in *Computer Graphics Forum*, vol. 29, no. 8. Wiley Online Library, 2010, pp. 2579–2600.

[2] K. Perlin, "An image synthesizer," *ACM Siggraph Computer Graphics*, vol. 19, no. 3, pp. 287–296, 1985.

[3] D. S. Ebert, F. K. Musgrave, D. Peachey, K. Perlin, and S. Worley, *Texturing & modeling: a procedural approach*. Morgan Kaufmann, 2003.

[4] A. Tatarinov, "Perlin noise in real-time computer graphics," in *GraphiCon*, 2008, pp. 177–183.

[5] S. Gustavson, *GLSL textureless classic 3D cnoise*, June 01, 2019. [Online]. Available: https://github.com/ashima/webgl-noise/blob/master/src/classicnoise3D.glsl

[6] W. Celes, *Textura Procedural*, June 26, 2019. [Online]. Available: https://webserver2.tecgraf.puc-rio.br/~celes/docs/inf2610/textura_procedural.pdf