Aplicação de textura e *bump* procedural em superfícies.

Bianca Faria Dutra Fragoso

PROJETO FINAL DE GRADUAÇÃO

Centro Técnico Científico – CTC

Departamento de Informática

Curso de Graduação em Engenharia da Computação

Rio de Janeiro, dezembro de 2019

Bianca Faria Dutra Fragoso

Aplicação de textura e *bump* procedural em superfícies.

Relatório de Projeto Final, apresentado ao programa de Engenharia da Computação da PUC-Rio como requisito parcial para a obtenção do título de Engenheiro de Computação.

Orientador: Waldemar Celes

Rio de Janeiro, dezembro de 2019

“It is the uncertainty that charms one. A mist makes things wonderful.”

(Oscar Wilde)

Agradecimentos

Resumo

FRAGOSO, Bianca. CELES, Waldemar. Aplicação de textura e *bump* procedural em superfícies. Rio de Janeiro, 2019. 34 p. Relatório Final de TCC – Departamento de Informática. Pontifícia Universidade Católica do Rio de Janeiro.

Nesse trabalho são discutidas algumas técnicas de criação de imperfeições em malhas através da utilização de texturas e *bumps* procedurais. Vários recursos são utilizados para que o resultado tenha um grau de aleatoriedade que pareça natural e que deixem o objeto com uma aparência mais realista. Dentre esses recursos, temos o ruído de Perlin que fornece a aleatoriedade de sujeiras, corrosões etc e é uma das ferramentas mais importantes desse trabalho, que contém uma explicação geral de seus parâmetros principais. Outro recurso que também é utilizado e discutido no trabalho é a iluminação de *rendering* baseado em física, que é usada para que os metais fiquem com uma aparência mais realista.

Palavras-chave

Textura procedural, Ruído de Perlin, Turbulência, Oitavas, Rendering baseado em física, *Bump.*

Abstract

FRAGOSO, Bianca. CELES, Waldemar. Application of procedural texture and *bump* on surfaces. Rio de Janeiro, 2019. 34 p. Final Report of Final Project II – Department of Informatics. Pontificial Catholic University of Rio de Janeiro.

This work discusses some techniques for creating mesh imperfections through the use of procedural textures and procedural bumps. A variety of features are used to make the result look more natural and to make the object look more realistic. Among these features, we have Perlin noise that provides the randomness of dirt, corrosion etc. and is one of the most important tools of this work, which contains a general explanation of its main parameters. Another feature that is also used and discussed in the paper is physics-based rendering lighting, which is used to make metals look more realistic.

Keywords

Procedural Texture, Perlin Noise, Turbulence, Octaves, PBR, Physically based rendering, Bump.

Sumário

Introdução 1
1. Ruído de Perlin 2
1.1 História 2
1.2 Exemplos de uso 3
1.3 Construção do Ruído 4
1.4 Testes utilizando o ruído como textura difusa 6
1.5 Testes utilizando o ruído para modificar a estrutura da malha 8
2. Rendering Baseado Em Física (PBR) 9
2.1 Teoria 9
2.1.1 Modelo de microfacetes 10
2.1.2 Conservação de energia 11
2.1.3 Função de Distribuição Reflexiva Bidirecional (BRDF) 12
2.2 Componentes no PBR 13
2.3 Iluminação 14
3. Implementação final de Textura e Bump em superfícies 16
3.1 Aplicação de textura e resultados 17
3.2 Aplicação de *Bump* para simulação de corrosão 19
3.2.2 Geração de um mapa de normal procedural 19
3.2.3 Funções para cálculo de coordenada z e seus resultados 20
3.2.4 Resultados do *bump* junto com o PBR 25
3.1.5 Resultados do *bump* com cor 25
4. Conclusão 26
Referências Bibliográficas 27

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 [6]. Podemos dizer que a pesquisa nesse contexto pode ser classificada em duas vertentes: sombreamento (*shading*) e texturização (*texturing*). De maneira bem simplificada, *shading* 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.

Nesse projeto estaremos interessados em questões relacionadas aos problemas de textura. Em geral, 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, i.e., 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 especificado à priori, ficando implícito no algoritmo; (3) Multiresolução: permite criar modelos de multiresolução e texturas, que podemos avaliar de acordo com a resolução desejada.

Técnicas procedurais têm sido usadas frequentemente para modelar superfícies ou a representação volumétrica de elementos da natureza tais como, madeira, pedras, mármore, granitos, montanhas etc. Motivado, em parte, pela disponibilidade computacional disponível atualmente, nos últimos 30 anos tem havido um desenvolvimento teórico vertiginoso de técnicas procedurais. Nesse cenário, resultados de renderização tem sido obtido através da modelagem numérica de aleatoriedade (fractal, funções de turbulência), com destaque para o denominado ruído de Perlin.

O foco desse trabalho, será explorar algoritmos de textura e bump procedurais, utilizando em especial a classe de ruído procedural desenvolvida por Perlin, chamada de ruído de Perlin. Serão aplicados tanto textura quanto bump em superfícies, fazendo uma análise do processo de desenvolvimento.

O objetivo final é deixar superfícies com um aspecto mais realista, adicionando imperfeições através da utilização de técnicas procedurais. Além disso, para as imagens ficarem com um aspecto mais realista, ainda que não seja o foco desse trabalho, será utilizado um modelo de iluminação baseado na teoria de *rendering* baseado em física (PBR) e por isso será fornecida uma explicação sobre os conceitos mais importantes desse assunto, que é muito amplo, mais focada no que estará sendo utilizado nesse projeto.

Para a elaboração desse trabalho, foi utilizada a linguagem de programação C++, a linguagem de *shading* glsl, a API de computação gráfica openGL e o framework Qt.

Esse trabalho está dividido da seguinte forma: No primeiro capítulo é dada uma visão geral sobre o ruído de Perlin que é parte fundamental desse trabalho. A história, um pouco da teoria de construção e alguns testes usando o ruído são apresentados. No segundo capítulo alguns conceitos e um resumo de como foi implementada a iluminação do *rendering* baseado em física são mostrados. No terceiro e mais importante capítulo, é mostrado como os conhecimentos adquiridos foram utilizados para implementar texturas e *bumps* que podem deixar superfícies com um aspecto mais realista através da adição de imperfeições. Finalmente, no quarto capítulo, é apresentada uma conclusão de todo o estudo e implementação.

1. Ruído de Perlin

1.1 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 [7] estabeleceu as bases para uma das classes mais populares de textura procedural em uso hoje, baseado essencialmente em ruído (uma abordagem estocástica). Essa técnica é também conhecida na literatura como ruído procedural (ver, [7] e [9]).

Como mencionado anteriormente, a adição eficiente de detalhes visuais a imagens sintéticas sempre foi um dos maiores desafios em computação

gráfica. O ruído procedural é uma das ferramentas fundamentais mais bem sucedidas usadas para gerar tais detalhes. 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.

Como apontado no artigo de Andrei Tatarinov [11], uma das vantagens do ruído de Perlin 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 cheio de 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 [7].

Segundo Perlin, essa abordagem traz algumas vantagens. Forma e textura se tornam independentes. A textura não precisa mais “caber” na superfície. Se mudarmos a forma, por exemplo tirando um pedaço dela, a aparência do material sólido irá mudar. Além disso, como todas as texturas procedurais, não é preciso que muitos dados sejam armazenados.

É importante ressaltar que o ruído de Perlin tem invariância estatística com relação a rotação, translação e escala. Ou seja, a textura ou *bump* do objeto calculados utilizando o ruído não são alterados quando realizamos uma dessas operações.

1.2 Exemplos de uso

O ruído tem sido usado para uma ampla e diversificada gama de finalidades em textura procedural, incluindo nuvens, ondas, mármore, tornados, trilhas de foguetes, ondas de calor e assim por diante.

Além disso, se o ruído de Perlin for estendido em uma dimensão adicional e considerar a dimensão extra como sendo o tempo, podemos animá-lo. Por exemplo, o ruído de Perlin 2D pode ser interpretado como terreno, mas o ruído 3D pode ser interpretado como ondas ondulantes em uma cena do oceano.

Atualmente o ruído é amplamente usado tanto na produção de filmes como em videogames, e é implementado em todos os principais softwares de modelagem 3D e animação, como o Autodesk 3ds Max e o Maya, Blender, o RenderMan da Pixar.

1.3 Construção do Ruído

Apesar do foco desse trabalho não ser a implementação do ruído de Perlin em si e sim os múltiplos usos dele aplicado em superfícies, será dada uma visão geral de como ele é estruturado. Isso será importante, visto que, na aplicação dele, o controle e entendimento de seus parâmetros é fundamental para se chegar a um resultado desejado.

É importante entender as etapas principais de construção do ruído. Existem duas partes principais para se obtê-lo. 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 [11]. 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. Isso significa, que não será totalmente aleatório, mas pseudoaleatório.

A segunda parte é a interpolação de números aleatórios. Ela 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.

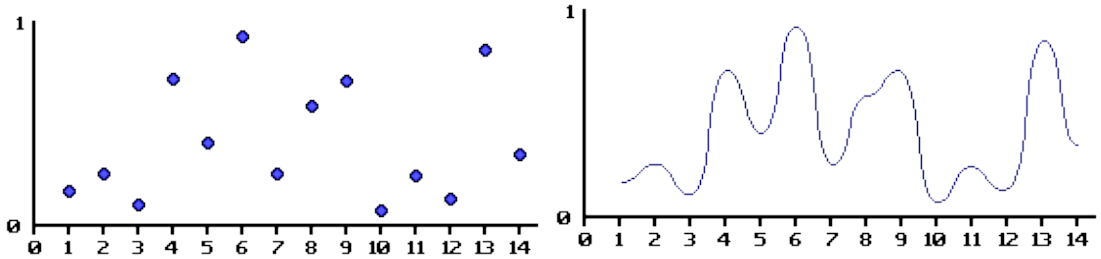

Figura 1: Na esquerda, gráfico mostrando números aleatórios de 0 a 1, um valor pseudoaleatório é atribuído a cada valor no eixo x. Na direita é mostrada a interpolação desses valores gerando uma função contínua.

Dessa forma, o ruído é construído. Nesse trabalho foi utilizada uma função que gera o ruído de Perlin [5]. Além do ruído clássico de Perlin, esse repositório ainda possui o ruído *Simplex* e alguns outros tipos de ruídos interessantes.

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.

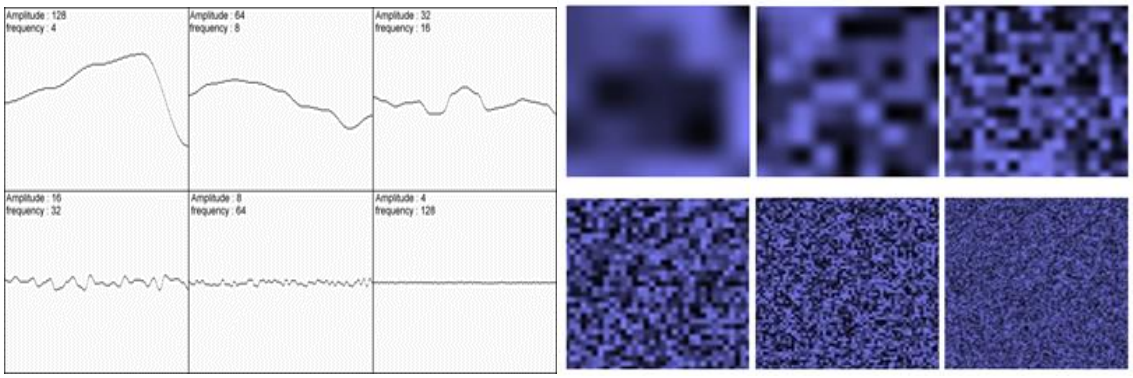

Figura 2: Exemplo de resultados de noise com diferentes frequências e amplitudes (oitavas). À esquerda 6 oitavas em 1D e na direita no 2D.

Essas oitavas são muito exploradas quando se usa o ruído. Por isso, alguns parâmetros delas são definidos formalmente, podem ser modificados e alteram o resultado final da textura gerada.

Parâmetros para construção de oitavas:

- **Frequência:** Refere-se ao período em que os dados são amostrados. Aplicamos a frequência na hora de passar o parâmetro na função de noise (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 noise. 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).

```
//Função de Turbulência
float turbulence(vec3 pos)
{
    float z = 0;
    float frequency = 4;
    float amplitude = 0.5;
    float persistency = 0.5;
    float lacunarity = 2;

    int numOctaves = 4;
    for(int i = 0; i < numOctaves;i++)
    {
        float octave = noise(frequency * pos) * amplitude;
        z += abs(octave);
        frequency *= lacunarity;
        amplitude *= persistency;
    }
    return z;
}
```

Figura 3: Exemplo de função que calcula oitavas com os parâmetros descritos e faz uma soma de seus módulos

1.4 Testes utilizando o ruído como textura difusa

Como testes iniciais da função de ruído, algumas simulações da natureza foram implementadas, tanto em 2D como em 3D.

O primeiro teste foi o de simulação de nuvens, em que os valores das oitavas do ruído são somados. Essa soma gera um número que é usado como interpolador das cores azul e branco. Assim, o efeito de nuvens é gerado.

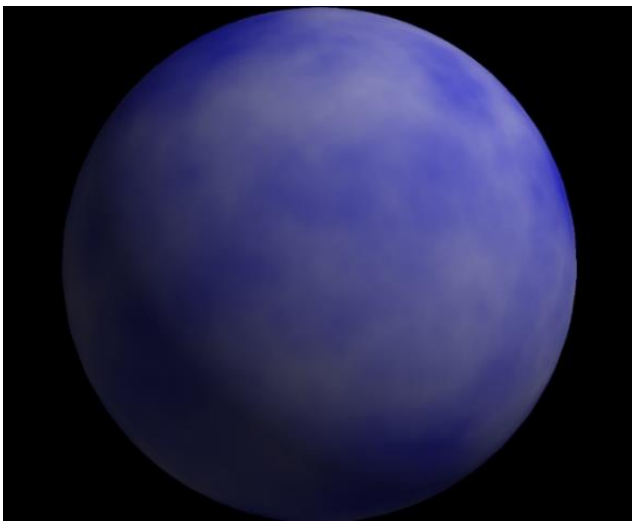

Figura 4: Usando Perlin Noise para produzir textura 3D da nuvem

O segundo teste que foi feito, consistia em simular a superfície do sol, misturando as cores laranja e amarelo. Para gerar o efeito desejado, uma função que soma o módulo das oitavas é utilizada.

Essa função é bem conhecida na literatura com o nome de Turbulência, está presente inclusive no artigo de 1985 de Perlin [7] a Figura 3 contém sua implementação. Essa função é usada como base de muitas outras funções que usam o ruído.

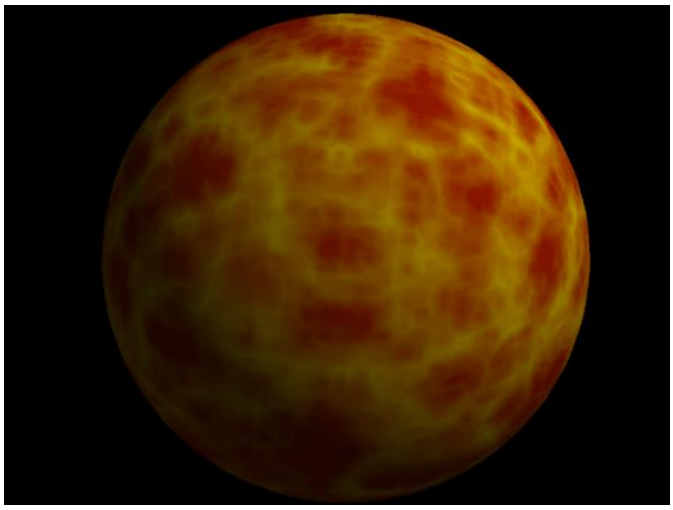

Figura 5: Textura de superfície do sol gerada usando Perlin Noise.

O último teste que foi feito, foi uma tentativa de simular mármore. A função de simulação de mármore utiliza as funções de soma de oitavas, de seno e de valor absoluto. O resultado é o valor absoluto do seno da coordenada x da posição (pois os veios estão na vertical) somado com a função de soma de oitavas.

Figura 6: Vaso de mármore conseguido através da função seno

1.5 Testes utilizando o ruído para modificar a estrutura da malha

O teste seguinte consistiu em modificar a própria estrutura de uma malha. Nesse caso, a malha era um grid de pontos no 2D, em que para cada ponto foi atribuída uma coordenada z gerada pelo ruído. Com isso, cada ponto ficava com uma altura diferente, aparentando ser um terreno.

No entanto, o teste tinha como objetivo simular o movimento da água e por isso, a animação das ondas era importante. Para isso, a cada frame a posição de seus vértices foi modificada, mas de maneira mais natural possível. Para isso, o ruído 3D foi utilizado, usando a terceira coordenada como sendo o tempo.

Para que se tivesse a impressão de que a água estava correndo e as ondas estão sendo propagadas, um incremento de tempo também foi usado na coordenada y.

A soma de oitavas geradas utilizando o ruído calculado para cada ponto, foi utilizado como coordenada z do ponto, que anteriormente tinha valor zero. Com uma nova superfície foi gerada, foi necessário recalcular as normais para cada ponto da superfície para fazer um cálculo correto de iluminação.

O cálculo foi feito aproveitando o fato de que tínhamos as informações dos pontos vizinhos, já que a discretização do grid era conhecida. A normal de segunda ordem, portanto, foi calculada fazendo o produto vetorial do vetor V com o vetor U, representados na Figura 7.

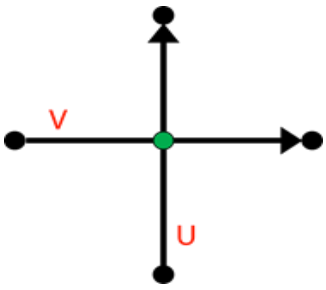

Figura 7: Vetores formados pelos vizinhos do ponto analisado.

Foi necessário testar diferentes parâmetros na geração do ruído. O principal parâmetro testado foi o número de oitavas. No entanto, a lacunarização, persistência e o brilho também foram variados para tentar gerar melhores resultados.

Os melhores resultados conseguidos foram usando 3, 4 e 5 oitavas. Com 3 oitavas, as elevações na água ficam maiores, já com 5 oitavas as elevações ficam bem

pequenas e são mais elevações do que com 3 oitavas. Com 4 oitavas o resultado ficou com um aspecto mais natural.

O resultado desse teste de mudança na estrutura da malha usando o ruído pode ser visto na Figura 8. No entanto, com a animação da água se movendo, o resultado visual fica melhor do que apenas com imagens.

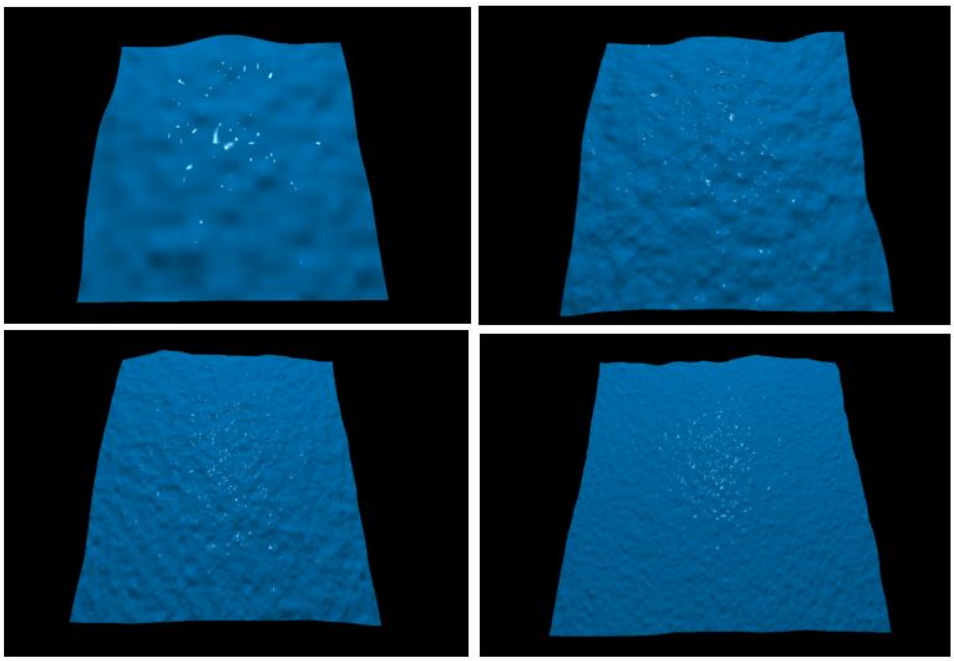

Figura 8: Simulação de água. (a)Usando 3 oitavas. (b) Usando 4 oitavas. (c) Usando 5 oitavas. (d) Usando 6 oitavas.

2. Rendering Baseado Em Física (PBR)

2.1 Teoria

Renderização baseada em física ou PBR, é uma coleção de técnicas de renderização que tentam simular a realidade, isto é, usam princípios da física para modelar a interação da luz com o objeto. Como a renderização com base em física visa imitar a luz de uma maneira fisicamente plausível, geralmente parece mais realista em comparação com outros algoritmos de iluminação bastante usados em computação gráfica, como Phong e Blinn-Phong.

Embora uma abordagem com base física possa parecer a maneira mais óbvia de abordar a renderização, ela só foi amplamente adotada na prática nos últimos 10 anos. A renderização com base física ainda é, no entanto, uma aproximação da realidade (com base nos princípios da física), razão pela qual não é chamada de sombreamento físico, mas sombreamento com base física.

Para que um modelo de iluminação PBR seja considerado baseado em fisica, ele deve atender às três condições:

- Seja baseado no modelo de superfície de microfacet.
- Conserva energia.
- Usa um BRDF com base física.

2.1.1 Modelo de microfacetes

Muitas abordagens baseadas em óptica geométrica para modelar a reflexão e a transmissão de superfície baseiam-se na ideia de que superfícies ásperas podem ser modeladas como uma coleção de pequenos microfacetes que são pequenos espelhos perfeitamente refletivos [8].

Essa ferramenta foi primeiramente desenvolvida pelos pesquisadores da comunidade óptica. Essa teoria foi introduzida na computação gráfica em 1977 por Blinn e em 1981 por Cook e Torrance (ver em [1]).

Quanto mais rugosa for a superfície, mais caótico estará cada microfacete ao longo da superfície. O efeito desses pequenos alinhamentos de espelho é que, quando se fala especificamente sobre iluminação especular, é mais provável que os raios de luz recebidos se espalhem ao longo de direções completamente diferentes em superfícies mais ásperas, resultando em uma reflexão especular mais difundida. Por outro lado, em uma superfície lisa, é mais provável que os raios de luz reflitam mais ou menos na mesma direção, proporcionando reflexos menores e mais nítidos.

Figura 9: À esquerda uma superfície áspera com microfacetes distribuídos de forma caótica com raios de luz refletindo em direções bem diferentes. À direita, uma superfície lisa com raios refletidos de maneira mais uniforme.

Nenhuma superfície é completamente lisa em um nível microscópico, mas, como esses microfacetes são pequenos o suficiente para que não possamos fazer uma distinção entre eles por pixel, aproximamos estatisticamente a rugosidade dos microfacetes da superfície, dada a rugosidade como parâmetro. Baseado na rugosidade de uma superfície, podemos calcular a taxa de microfacetes alinhados a um vetor que fica no meio entre o vetor da fonte de luz e o vetor do observador (*halfway vector*).

Quanto mais os microfacetes estiverem alinhados com o vetor intermediário, mais nítida e mais forte será a reflexão especular.

2.1.2 Conservação de energia

O conceito de conservação de energia afirma que um objeto não pode refletir mais luz do que recebe. O que nos leva a uma conclusão importante: a refração e a reflexão são mutuamente exclusivas. Isso significa que objetos altamente refletivos mostrarão pouca ou nenhuma luz difusa, simplesmente porque pouca ou nenhuma luz penetra na superfície, tendo sido refletida principalmente. O inverso também é verdadeiro: se um objeto refrata muito (difusa clara), terá uma especular pequena, pois refletirá menos.

Figura 10: Esfera com refletividade aumentando da esquerda para direita e com rugosidade, ao mesmo tempo, diminuindo.

Observando a imagem acima, vemos a área de reflexão especular diminuir, porém seu brilho aumenta com a diminuição dos níveis de rugosidade. Se a intensidade especular fosse a mesma em cada pixel, independentemente do tamanho da forma especular, as superfícies mais ásperas emitiriam muito mais energia, violando o princípio de conservação de energia. É por isso que vemos reflexos especulares mais intensamente em superfícies lisas e mais fracamente em superfícies ásperas [3].

É importante entender que a luz refratada tem dois comportamentos em objetos dielétricos (não metálicos). Parte dela é absorvida e a outra parte atravessa o objeto, colide com as partículas e volta para a superfície, contribuindo para a cor do objeto (cor difusa).

Existe uma sutileza quando se trata de reflexão e refração em superfícies metálicas, que são os tipos de superfícies que serão mais utilizadas nesse trabalho. Elas absorvem toda a luz refratada, não há dispersão e a luz é totalmente refletida (componente especular). Por isso, superfícies metálicas não mostram cor difusa [3].

2.1.3 Função de Distribuição Reflexiva Bidirecional (BRDF)

O BRDF é uma função que descreve quanto de luz é refletida quando a luz entra em contato com algum tipo de material. Ou seja, para diferentes tipos de materiais existem BRDFs diferentes.

Matematicamente falando, é uma função que recebe como parâmetros a direção da luz de entrada (ω_i), a direção de saída da reflexão (ω_o) cada um relativo à um pequeno elemento da superfície. O BRDF é definido como o quociente da divisão da quantidade de luz refletida na direção ω_o pela quantidade de luz que atinge a superfície na direção ω_i [12]. Para deixar mais claro, podemos chamar a quantidade de luz refletida na direção ω_o de L_0 e a quantidade de luz incidente na direção ω_i de E_i. Então o BRDF é:

$$\text{BRDF} = \frac{Lo}{E_i}$$

Existem vários BRDFs com base física. No entanto, quase todos os pipelines de renderização em tempo real usam um BRDF conhecido como Cook-Torrance [10].

O BRDF de Cook-Torrance contém uma parte difusa e uma parte especular. Na equação abaixo, k_d é a taxa de energia luminosa que é refratada e k_s é a proporção que é refletida.

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}$$

A parte esquerda da equação é chamada de difuso lambertiano, que é um fator constante. Com c sendo o albedo (cor difusa). O π serve para normalizar a luz difusa.

$$f_{lambert} = \frac{c}{\pi}$$

Existem equações diferentes para a parte difusa do BRDF, que tendem a parecer mais realistas, mas também são mais caras em termos computacionais. Conforme concluído pela Epic Games, no entanto, o difuso lambertiano é suficiente para a maioria dos propósitos de renderização em tempo real. A parte especular do BRDF é um pouco mais avançada e é descrita como:

$$f_{cook-torrance} = \frac{DFG}{4(\omega_0 \cdot n)(\omega_i \cdot n)}$$

Para o cálculo da parte especular do BRDF de Cook-Torrance, são utilizadas 3 funções:

- **Função de distribuição normal:** É a função que aproxima a área em que os microfacetes da superfície estão alinhados com o vetor intermediário (*halfway vector*) influenciado pela rugosidade da superfície. Como já dito anteriormente, isso é importante pois quanto mais microfacetes estão alinhados com o vetor h mais nítida e forte será a reflexão especular. Portanto, essa função recebe como parâmetros, a normal n o vetor h e a rugosidade $\propto$.

- **Função de geometria:** Calcula a área da sua micro-superfície em que os microfacetes se sobrepõem, causando um a oclusão do outro. Superfícies mais ásperas que possuem microfacetes em direções variadas são mais propensas a terem microfacetes se ocludindo. Por isso é importante levar em conta essa equação para simular a realidade mais fielmente, visto que fisicamente superfícies mais rugosas se apresentarão mais escuras com relação às lisas por causa dessas oclusões.

- **Equação de Fresnel:** Ela mede porcentagem de luz que reflete em uma superfície em diferentes ângulos e levando em consideração a conservação de energia, também a porcentagem de luz refratada. Ela depende do ângulo em que o observador está olhando para a superfície. A equação de Fresnel é uma equação bem complexa, por isso é usada geralmente uma aproximação, a equação de Fresnel-Schlick.

2.2 Componentes no PBR

As propriedades físicas do PBR podem ser fornecidas através de texturas. Existem 5 tipos de propriedades que geralmente são usadas na implementação de algum modelo de PBR, contribuindo para que ele tenha um aspecto mais realista.

- **Albedo:** É o equivalente à textura difusa. Uma das maiores diferenças entre um mapa de albedo em um sistema PBR e um mapa difuso tradicional é a falta de luz direcional ou oclusão do ambiente. A luz direcional parecerá incorreta em determinadas condições de iluminação e a oclusão do ambiente deve ser adicionada na textura AO separadamente. O Albedo especifica para cada texel qual é sua cor, ou no caso de texels metálicos, qual é a sua refletividade de base.

- **Normal:** O mapa normal é usado para simular detalhes e irregularidades da superfície. É um mapa RGB em que cada componente corresponde às coordenadas X, Y e Z da superfície normal.

- **Metálico:** mapa metálico especifica se um texel é metálico ou não. Geralmente, valores em escala de cinza ou como preto ou branco (binário) são utilizados para representação se um texel é ou não metálico.

- **Rugosidade:** o mapa de rugosidade especifica a rugosidade de uma superfície por texel. O de rugosidade influencia as orientações dos microfacetes da superfície. Uma superfície mais áspera obtém reflexos mais amplos e borrados, enquanto uma superfície lisa obtém reflexos focados e claros.

- **Oclusão de Ambiente (AO):** a oclusão de ambiente especifica um fator de sombreamento extra da superfície. Se temos uma superfície de tijolo, por exemplo, a textura albedo não deve ter informações de sombra nas fendas do tijolo. O mapa AO, no entanto, especifica essas bordas escuras, pois é mais difícil a luz escapar por ali. A consideração da oclusão de ambiente no final do estágio de iluminação pode aumentar significativamente a qualidade visual da cena.

2.3 Iluminação

Para implementação da iluminação do PBR, foram utilizados todos os conceitos apresentados acima e foi baseado no workflow metálico [3]. Nessa parte é explorada a equação de refletância.

A equação de refletância possui um conceito de radiometria chamado de radiância. No caso de luzes direcionais pontuais, que são as que foram usadas nesse trabalho, a função de radiância L serve para calcular a cor da luz atenuada pela sua distância para um ponto p .

$$L_0(P, \omega_0) = \int_{\Omega} \left(k_d \frac{c}{\pi} + \frac{DFG}{4(\omega_0 \cdot n)(\omega_i \cdot n)} \right) L_i(p, \omega_i) n \cdot \omega_i \, d\omega_i$$

O componente ω_i que é o vetor de incidência da luz, conseguimos facilmente subtraindo a posição da luz pela posição do fragmento que estamos analisando.

Para cada luz presente na cena, calculamos essa radiância da luz multiplicando a cor da luz pela atenuação. A atenuação dela é calculada como sendo $\frac{1}{distance^2}$. Também multiplicamos a radiância pelo produto interno entre a normal e o vetor de incidência da luz. Assim, boa parte da equação de refletância já é calculada.

Depois, é implementada a parte do BRDF, em que é utilizada uma função de cálculo da distribuição normal, duas funções para o cálculo da distribuição geométrica e outra para a função de Fresnel.

Com Fresnel, a porcentagem de luz refletida é calculada. Subtraindo 1 pela porcentagem de luz refletida a fração de luz refratada também é calculada. Com o resultado de todas essas funções, é possível calcular a componente especular da equação já mencionada anteriormente:

$$f_{cook-torrance} = \frac{DFG}{4(\omega_0 \cdot n)(\omega_i \cdot n)}$$

Adicionando algumas texturas, foi possível obter as componentes: Albedo, Ao, rugosidade e metálico. Com o resultado das funções descritas acima e essas outras componentes, foi possível calcular a equação de refletância. O albedo corresponde ao c da equação.

A componente da luz ambiente é calculada multiplicando o albedo pela AO, para considerar a oclusão do ambiente. Como objetos metálicos não têm componente difusa, a cor final é calculada somando a componente ambiente e o resultado da equação de refletância.

No final também é feita uma correção gama para garantir que o resultado esteja no espaço linear de cores.

2.4 PBR x Phong Testes no trabalho

Nas figuras a seguir (Figura 11 e Figura 12) é possível ver uma comparação entre malhas com a iluminação de Phong e com a iluminação de PBR descrita anteriormente. Para esses testes de PBR, foram utilizadas 4 luzes brancas.

Todos os componentes citados anteriormente como Albedo, Rugosidade, Ao e metálico, foram utilizados, usando texturas próprias para implementação de PBR.

Figura 11: À esquerda, malha de brinco com iluminação de Phong e à direita utilizando a iluminação do PBR, usando fluxo metálico

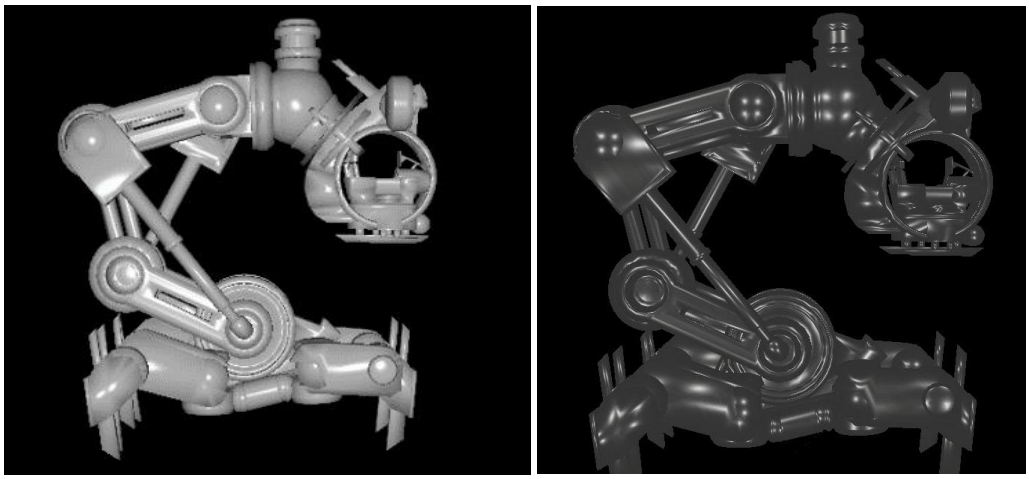

Figura 12: À esquerda, malha de robô com iluminação de Phong e à direita utilizando a iluminação do PBR, usando fluxo metálico

3. Implementação final de Textura e Bump em superfícies

Essa é a parte mais importante do trabalho em que todos os conhecimentos e experiências adquiridos ao longo dos testes foram colocados em prática para geração de superfícies mais realistas. Foram introduzidas imperfeições em superfícies, como: sujeira, arranhões, corrosões e todos eles juntos, tentando simular a realidade.

Para os testes de aplicação das texturas e *bumps* gerados proceduralmente por esse trabalho, foram utilizadas malhas de triângulos, quads ou mistas.

3.1 Aplicação de textura e resultados

Antes dos testes de geração de *bump*, 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 turbulência e de soma de oitavas foram novamente utilizadas. A primeira gerava um aspecto de uma sujeira menos pesada e a segunda de uma sujeira pesada.

Essas funções retornavam um número real da função de ruído para aquela coordenada do vértice da malha. No entanto, queríamos gerar uma cor. Por isso, como feito também nos testes de simulação de nuvem e superfície solar descritos anteriormente, foi usada a função de *mix* do glsl que interpola linearmente dois vetores. O número a obtido pela função de ruído é usado como o interpolador de duas cores, branco e a cor de sujeira (um marrom claro) como mostrado abaixo.

$$objectColor \cdot (1 - a) + dirtColor \cdot a$$

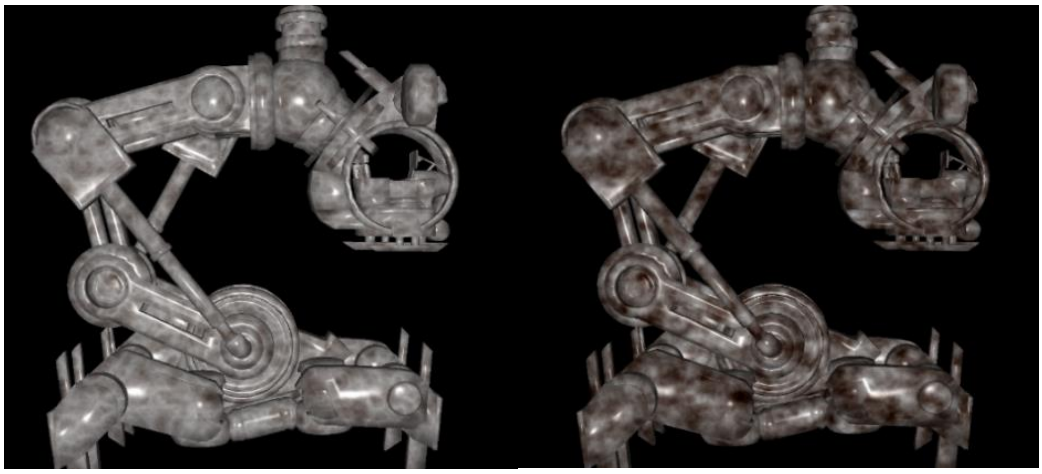

Figura 13: Robô com sujeira leve à esquerda e robô com sujeira pesada à direita

Figura 14: Brinco com sujeira leve à esquerda (turbulência) e brinco com sujeira pesada à direita (soma de oitavas)

O PBR também foi explorado nessa parte. Os melhores resultados gerados nos testes de aplicação de textura, foram da soma de oitavas no lugar da textura de rugosidade e no lugar do albedo.

No caso do ruído no lugar da rugosidade, Figura 14 (a), a malha fica com uma aparência de desgastado, pois algumas partes da malha ficam mais lisas (menos rugosas) que outras. Quando o ruído é utilizado no lugar do albedo, Figura 14 (b), a malha fica com uma aparência de sujo, visto que a textura de albedo é semelhante à textura difusa. No entanto, esse resultado não fica tão interessante, visto que superfícies metálicas não possuem cor difusa, o albedo nesses casos serve para descrever uma cor que será a base da reflexão e não uma cor difusa como em objetos dielétricos.

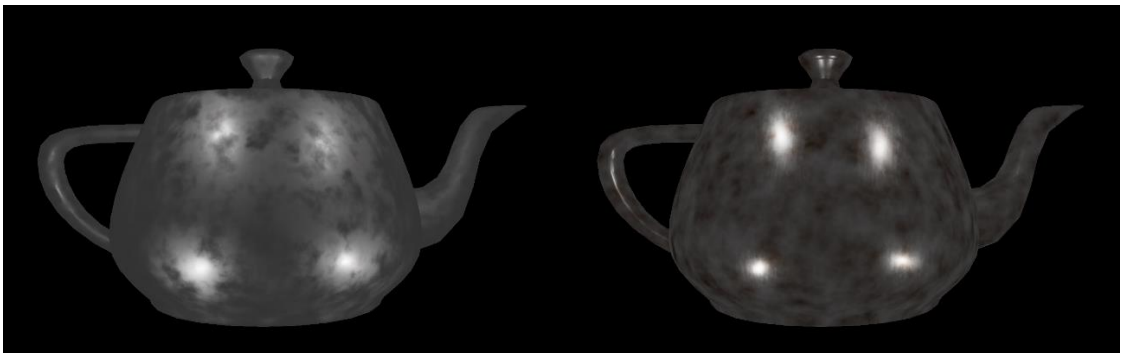

Figura 15: Textura procedural junto com PBR. (a)PBR com ruído no lugar da textura de rugosidade. (b) PBR com ruído no lugar da textura de albedo.

3.2 Aplicação de *Bump* para simulação de corrosão

Na aplicação de *bump* procedural, o objetivo era gerar um mapa de normal que se assemelhasse a corrosões, desgastes, oxidações, arranhões etc.

3.2.2 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 das malhas, ou seja, as coordenadas já multiplicadas pela matriz *mvp* vindas do *vertex shader*.

Os pontos de cima, de baixo, da direita e da esquerda de um vértice, foram calculados usando a função *built-in* de derivadas do glsl, da seguinte maneira:

```
vec3 left = projPos - dFdx(projPos);
vec3 right = projPos + dFdx(projPos);
vec3 up = projPos + dFdy(projPos);
vec3 down = projPos - dFdy(projPos);
```

Essas coordenadas já são projetadas na tela, portanto, suas coordenadas z são zero. 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, calculada por uma função que utiliza o ruído de Perlin.

O exemplo abaixo mostra como são calculadas as coordenadas z de cada vizinho do ponto analisado. A coordenada de mundo desse vizinho é mandada como parâmetro de uma função que utiliza o ruído de Perlin e retornará um número real.

```
left.z = calculateNoise1(worldPos - dFdx(worldPos));
right.z = calculateNoise1(worldPos + dFdx(worldPos));
up.z = calculateNoise1(worldPos + dFdy(worldPos));
down.z = calculateNoise1(worldPos - dFdy(worldPos));
```

Com a coordenada z de cada vizinho alterada, a normal do mapa de normal pode ser calculada. Usando a mesma técnica utilizada no teste de simulação de água (Figura 7), criam – se dois vetores utilizando as coordenadas dos pontos vizinhos e com o produto vetorial entre eles, é calculado um vetor que equivale à normal naquele ponto.

```
vec3 v1 = normalize(right - left);
vec3 v2 = normalize(up - down);
vec3 normal = normalize(cross(v1, v2));
```

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. 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. 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 ela que é a responsável pelo efeito de bump dos resultados.

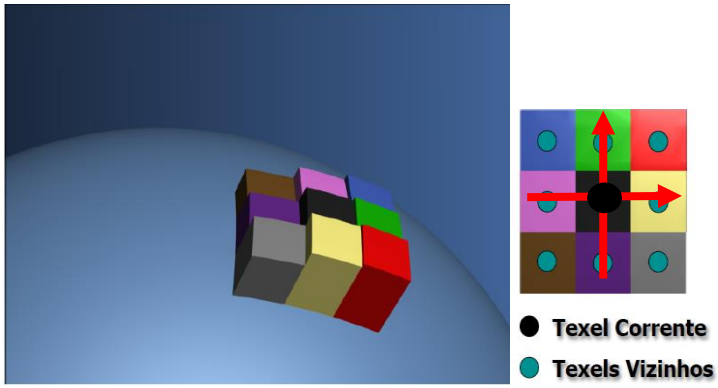

Figura 16: (a) Consultar na superfície do objeto valores dos texels vizinhos [2]

3.2.3 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 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 mais 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 como pode ser percebido na Figura 17 (a). Isso acontece porque em todos os pontos da malha a normal foi 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 foi utilizada. Portanto, os resultados seguintes foram obtidos usando a função de *min (*que retorna o mínimo entre dois números*)* com um número e o resultado da função de turbulência.

Testes de que números poderiam ser utilizados juntamente com o resultado da função de turbulência foram feitos até que fosse atingido um resultado mais desejado. Primeiro, o número 0.3 foi utilizado gerando o resultado obtido na Figura 17 (b) gerando um resultado um pouco menos cheio de detalhes comparado com o anterior em que a função de turbulência foi utilizada diretamente.

```
z = min(0.3,turbulence(pos));
```

Em seguida foram utilizados valores menores como forma de filtrar ainda mais a função de turbulência. Valores como 0.2 (Figura 17(c)), 0.1 (Figura 17 (d)) e 0.05 (Figura 17 (e)) também foram utilizados. 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. No caso da função de cálculo de z que utilizava a função de turbulência, os valores 0.05 e 0.1 na função de *min* foram os que geraram melhores resultados que se assemelhavam à arranhões.

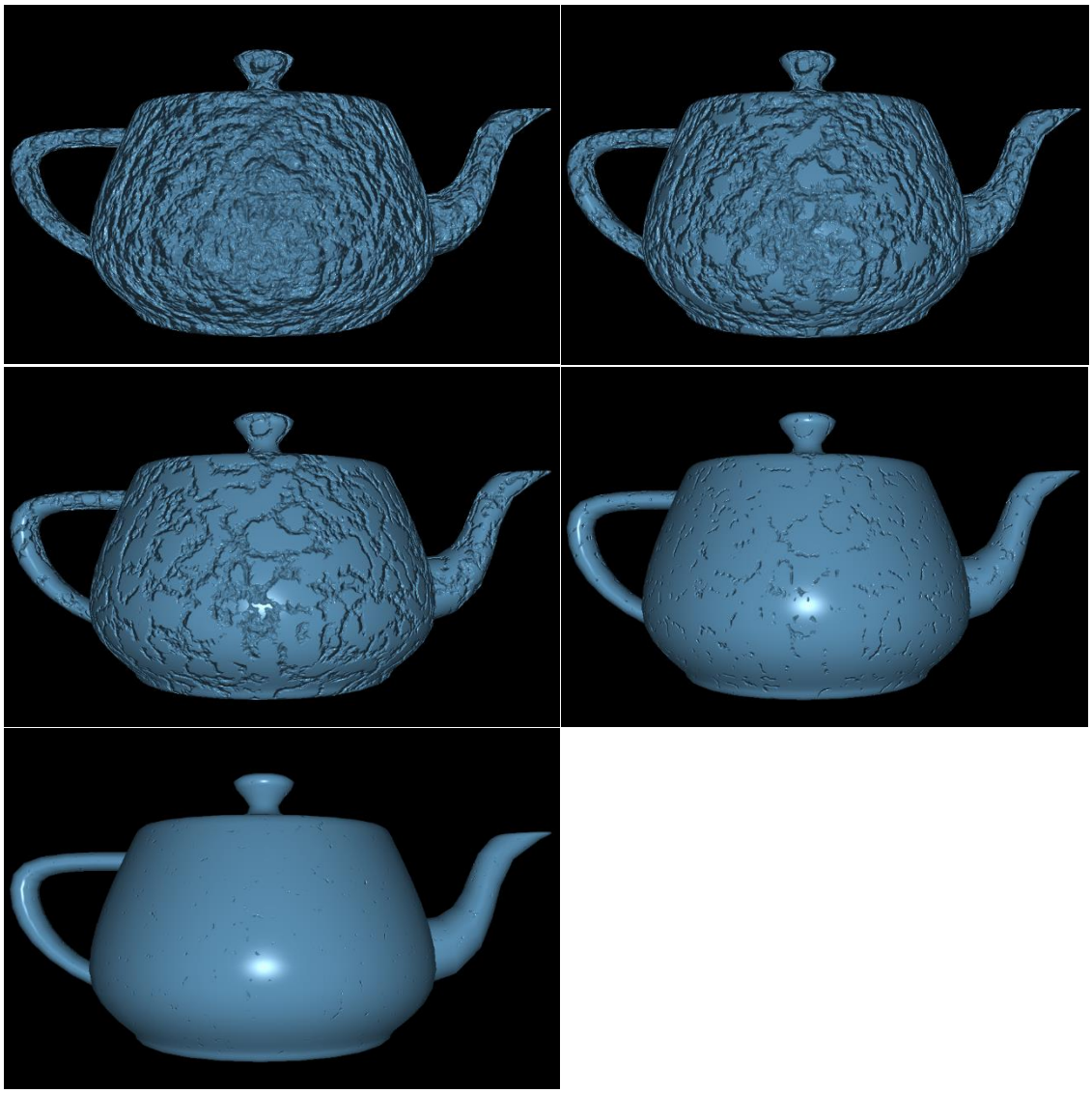

Figura 17: Usando função de turbulência. (a) Função de turbulência aplicada diretamente. (b) Função de min com função de turbulência e 0.3. (c) Função de min com função de turbulência e 0.2. (d) Função de min com função de turbulência e 0.1. (d) Função de min com função de turbulência e 0.05.

Em seguida, foi criada outra função de cálculo da coordenada z utilizando dessa vez a função de soma de oitavas. Essa função se aplicada diretamente para o cálculo de z dá um resultado parecido com o acima em que a função de turbulência é usada, e pode ser visto na Figura 18 (a). Isso acontece também pelo fato de todos os pontos terem normais que estão sendo perturbadas.

Utiliza-se a mesma estratégia de antes e a função de *min* é usada como forma de restringir a função de soma de oitavas. Os mesmos valores são usados. Nesse caso, as imperfeições se assemelham mais com corrosões, como pode-se perceber na Figura 18 (d) e Figura 18 (e).

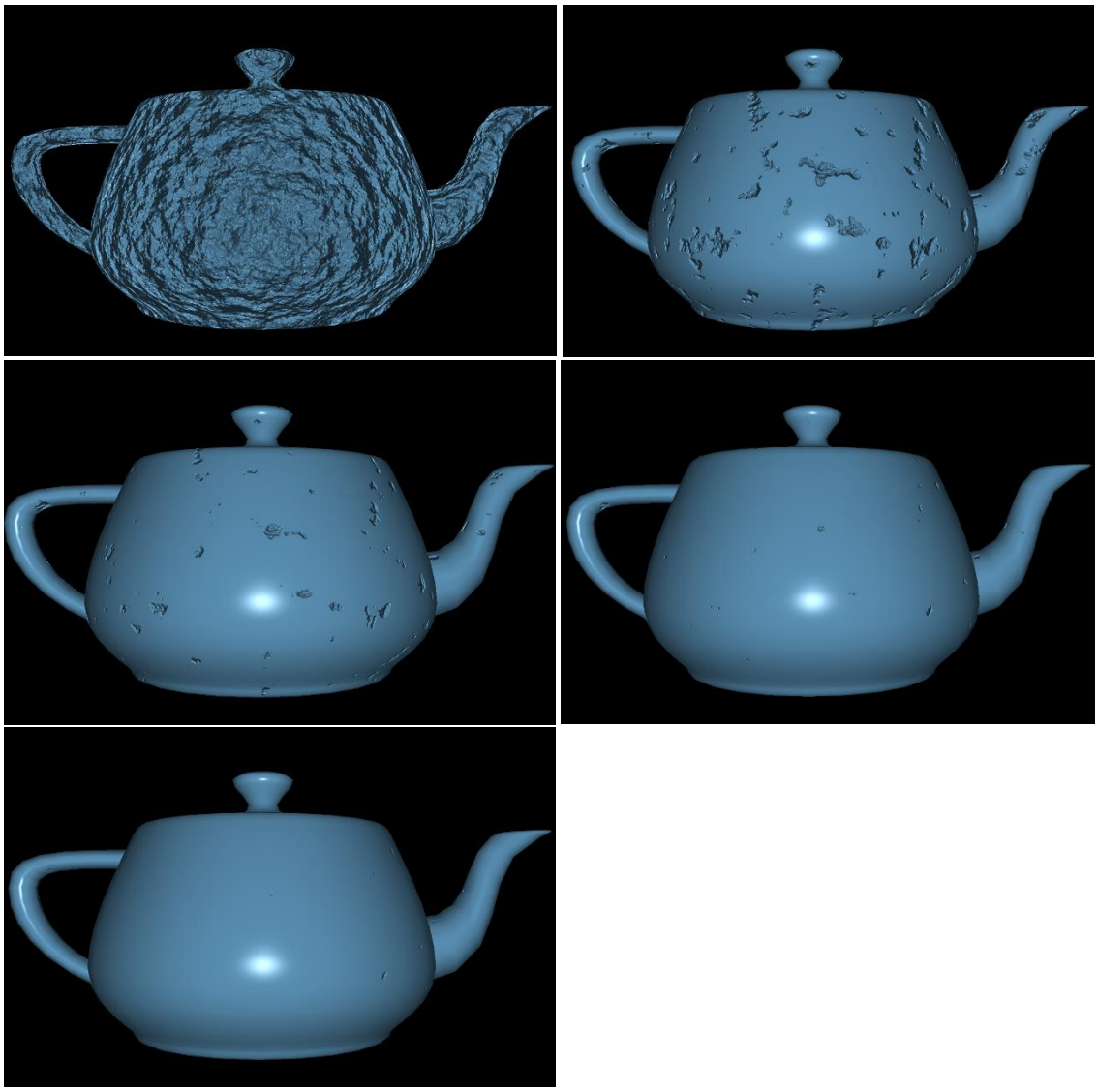

Figura 18: Usando função de soma de oitavas. (a) Função de soma de oitavas aplicada diretamente. (b) Função de min com função de soma de oitavas e 0.3. (c) Função de min com função de soma de oitavas e 0.2. (d) Função de min com função de soma de oitavas e 0.1. (d) Função de min com função de soma de oitavas e 0.05.

A função de criação de mármore também foi testada e apresentou resultados diferentes com imperfeições com uma aparência de arranhões como se o objeto estivesse quebrado. A função que foi utilizada foi a mesma de geração de veios de mármore utilizada para simular o vaso. No entanto, novamente foi utilizada a função de mínimo pois se a função de mármore fosse diretamente aplicada o resultado gerado, como mostrado na Figura 19 (a), ficava com imperfeições muito grandes e grosseiras.

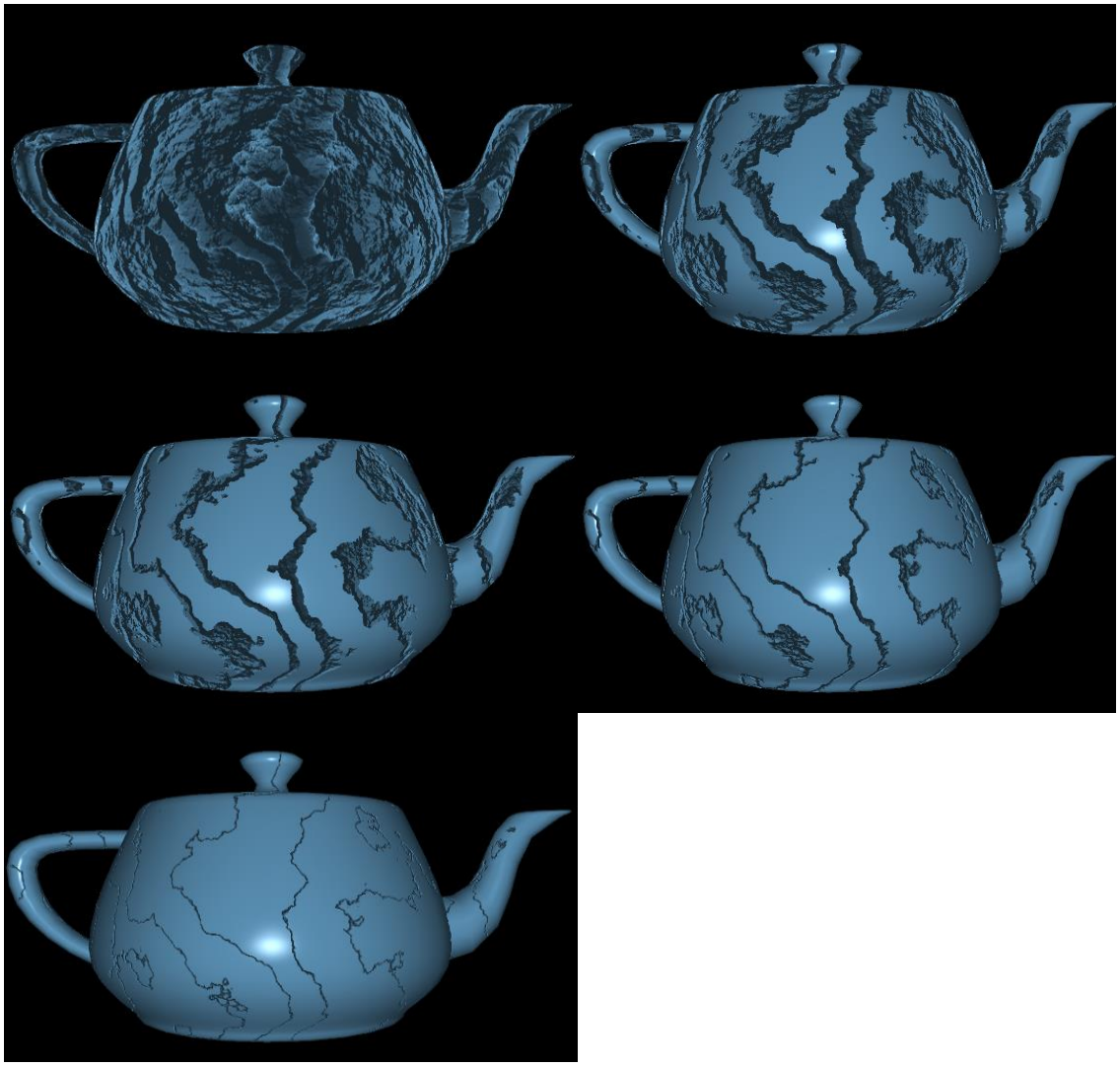

Figura 19: Usando função de mármore. (a) Função de mármore aplicada diretamente. (b) Função de min com função de mármore e 0.3. (c) Função de min com função de mármore e 0.2. (d) Função de min com função de mármore 0.1. (d) Função de min com função de mármore e 0.05.

3.2.4 Resultados do *bump* junto com o PBR

Figura 20: Malha de bule com iluminação baseada em física para metais, com rugosidade modificada pelo ruído. (a) Função de soma de oitavas. (b) Função de turbulência. (c) Função de mármore.

3.1.5 Resultados do *bump* com cor

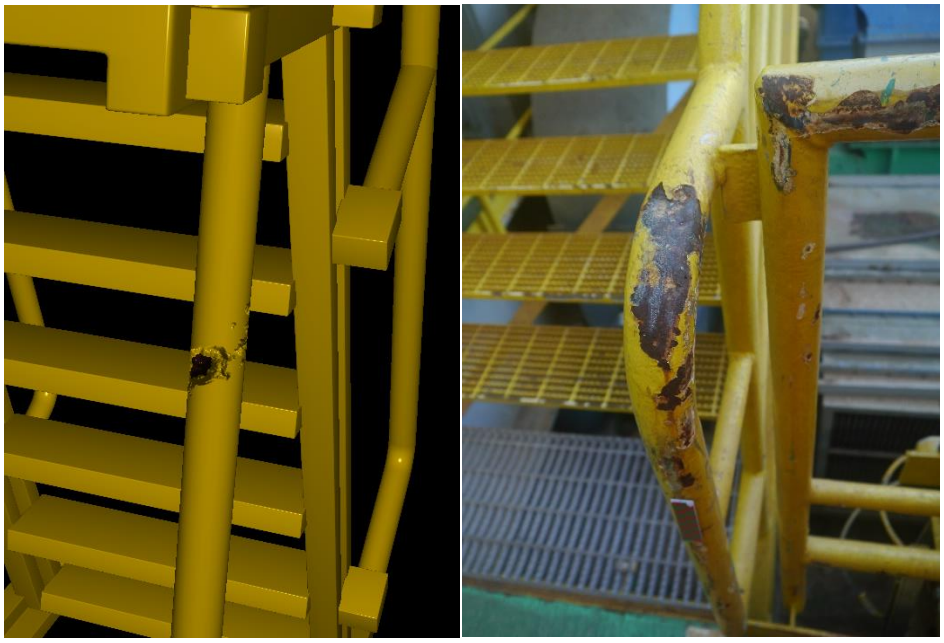

Figura 21: Comparação entre malha de escada com bump e com cor e escada real de indústria.

4. Conclusão

Nesse trabalho, pesquisas e testes foram feitos para se entender um pouco melhor os parâmetros do ruído de Perlin. Também foram feitos testes de iluminação de PBR para simulação de metais.

Porém, a parte mais importante foi a de se criar imperfeições como sujeira, corrosões, oxidações etc utilizando os conceitos citados acima. Nessa parte, texturas e *bumps* foram criados proceduralmente e testados 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. Por exemplo, a função de soma de oitavas quando usada para criação dos *bumps* apresentava características que se assemelhavam a corrosões. No entanto, controlar onde essas imperfeições ficarão e qual será o tamanho delas, o que às vezes pode ser algo desejado, é mais complexo pois estamos lidando com aleatoriedade.

Além disso, dependendo da discretização e tamanho da malha, os resultados variavam bastante. Uma malha muito grande e discretizada apresentava resultados não tão bons quanto as malhas apresentadas que continham menos vértices e eram menores, pois como o ruído era aplicado por texel, muitas vezes uma malha com muitos vértices ficava com sujeiras ou imperfeições muito pequenas e em muitas quantidades.

Uma outra dificuldade durante o trabalho foi a aplicação de cores exatamente onde as imperfeições do bump estavam. Nem sempre a cor ficava exatamente onde deveria, variando dependendo da malha.

No entanto, foram conseguidos resultados variados de geração de imperfeições em malhas, que apresentaram um padrão parecido dependendo da função usada na maioria das malhas testadas. Essas técnicas podem ser aplicadas em outras malhas para que elas se tornem mais realistas.

Referências Bibliográficas

[1] AKENINE-MOLLER, Tomas; HAINES, Eric; HOFFMAN, Naty. Real-time rendering. AK Peters/CRC Press, 2018.

[2] CELES, Waldemar. Disponível em: <https://webserver2.tecgraf.puc-rio.br/~celes/docs/inf2610/textura_procedural.pdf> Acesso em: 01 de jun. de 2019.

[3] DE VRIES, Joey. Theory. Disponível em: <https://learnopengl.com/PBR/Theory> Acesso em: 01 de out. de 2019.

[4] Ebert, D. S., Musgrave, F.K., Peachey, D., Perlin, K., Worley, S. Texturing and Modeling – A procedural approach. 3 ed. USA: Elsevier Science, 2003.

[5] GUSTAVSON, Stefan. GLSL textureless classic 3D noise "cnoise". Disponível em: < https://github.com/ashima/webgl-noise/blob/master/src/classicnoise3D.glsl> Acesso em: 01 de jun. de 2019.

[6] Lagae, A., Lefebvre, S., Cook, R., DeRose, T., Drettakis, G., Ebert, D., Lewis, J., Perlin, K. and Zwicker, M. (2010). A Survey of Procedural Noise Functions. Computer Graphics Forum, 29(8), pp.2579-2600.

[7] Perlin, K. 1985. An image synthesizer. In B. A. Barsky, ed., Computer

Graphics (SIGGRAPH ’85 Proceedings), 19(3):287–296.

[8] PHARR, Matt; JAKOB, Wenzel; HUMPHREYS, Greg. Physically based rendering: From theory to implementation. Morgan Kaufmann, 2016.

[9] Pietroni, N., Cignoni, P., Otaduy, M.A. and Scopigno, R. A survey on solid texture synthesis. JOURNAL OF LATEX CLASS FILES, VOL. 6, NO. 1, JANUARY 2007.

[10] Robert Cook and Kenneth Torrance. A reflectance model for computer graphics. Computer Graphics (Proceedings of SIGGRAPH 81), pages 244-253, 1981.

[11] TATARINOV, Andrei. Perlin noise in real-time computer graphics. In: GraphiCon. 2008. p. 177-183.

[12] WYNN, Chris. An introduction to BRDF-based lighting. Nvidia Corporation, 2000.