

# **Atividade 4 - Filtragem de texturas**

**José Felipe Nunes da Silva - 201700196010**

**Rebeca Raab Bias Ramos - 20170070453**

---

## **Introdução**

Este trabalho consiste em apresentar uma análise comparativa dos diversos tipos de filtros de texturas disponíveis no three.js. Tendo em vista que, o uso de filtros às vezes se torna necessário já que, nem sempre a imagem terá o mesmo número de pixels para um texel. Com isso, pode-se ocorrer os efeitos nomeados de *Magnificação* e *Minificação*.

## **Magnificação e Minificação**

Quando uma textura é magnificada, mais de um pixel pode ser mapeado para um único texel. As técnicas de filtragem mais comuns são o vizinho mais próximo (*NearestFilter*) e interpolação bilinear (*LinearFilter*). Uma característica do vizinho mais próximo é que os texels individuais podem se tornar aparentes. Este efeito é chamado de *pixelização* e ocorre porque o método assume o valor do texel mais próximo do centro de cada pixel durante a ampliação, resultando em uma aparência em blocos. Embora a qualidade deste método às vezes seja ruim, ele requer apenas um texel para ser buscado por pixel. Já na interpolação bilinear, para cada pixel são encontrados os quatro texels vizinhos, que são interpolados linearmente em duas dimensões para encontrar um valor combinado para o pixel. O resultado é mais desfocado e grande parte dos recortes de usar o mais próximo método vizinho desapareceu.

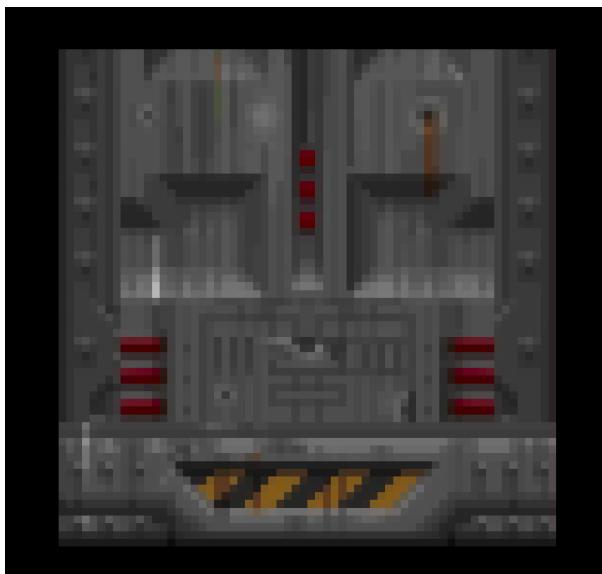
Quando uma textura é minimizada, vários texels podem cobrir a célula de um pixel. Para obter um valor de cor correto para cada pixel, deve-se integrar o efeito dos texels que influenciam a cor do pixel. No entanto, é difícil determinar com precisão a influência exata de todos os texels perto de um determinado pixel. Devido a essa limitação, vários métodos diferentes são usados nas GPUs. Um método é usar o vizinho mais próximo(*NearestFilter*), que funciona exatamente como o correspondente filtro de ampliação, ou seja, ele seleciona o texel que é visível no centro da célula do pixel. Este filtro pode causar graves problemas de *aliasing*. Outro filtro frequentemente usado é a interpolação bilinear(*LinearFilter*), novamente funcionando exatamente como o filtro de ampliação. Este filtro é apenas um pouco melhor do que o vizinho mais próximo para a

minificação. Consiste em combinar quatro texels em vez de usar apenas um, mas quando um pixel é influenciado por mais de quatro texels, o filtro falha e produz *aliasing*.

Os resultados dos efeitos de ampliação e tipos de filtragem são discutidos nas sessões a seguir.

## Implementação

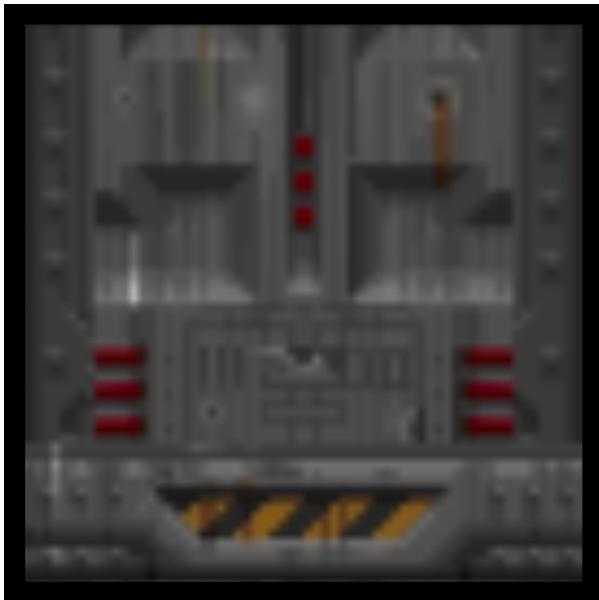
A implementação dos testes comparativos foi realizada a partir do *framework* na linguagem JavaScript fornecido pelo professor, fazendo uso da biblioteca `THREE.js`. O resultado inicial com o framework fornecido pelo professor é exibido na figura a seguir.



Na figura acima, é exibido o ponto inicial do trabalho, tendo como destaque o uso da textura do portão utilizada no jogo DOOM, sofrendo o efeito de magnificação e sendo aplicado o filtro de vizinho mais próximo, tal configuração pode ser notada no trecho a seguir do código fonte.

```
image.onload = function() {
    texture.needsUpdate = true;
    texture.magFilter = THREE.NearestFilter; // Magnificação com filtro vizinho
    mais próximo
    texture.minFilter = THREE.NearestFilter; // Minificação com filtro vizinho
    mais próximo
    texture.anisotropy = 1; // índice de anisotropia
    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.RepeatWrapping;
};
```

A alteração da linha destacada no trecho do código fonte acima para `texture.magFilter = THREE.LinearFilter;` implica na utilização da filtragem bilinear para o efeito de magnificação e o resultado é exibido na figura abaixo.



Os resultados exibidos daqui pra frente utilizam uma configuração diferente da inicialmente fornecida pelo professor. Foram alteradas a textura, que agora é repetida ao decorrer da geometria, a posição da câmera e as dimensões da geometria (inicialmente cúbica). A nova configuração é exibida no trecho de código a seguir.

```
image.src = comeia; // -> imagem no formato base64

texture = new THREE.Texture(image);

image.onload = function() {
    texture.needsUpdate = true;
    texture.magFilter = THREE.NearestFilter;
    texture.minFilter = THREE.NearestFilter;
    texture.anisotropy = 1;
    texture.wrapS = THREE.RepeatWrapping;
    texture.wrapT = THREE.RepeatWrapping;
};

// repetição da textura
texture.repeat.set(5, 20);

// ...
```

```
camera.position.z = 3;  
camera.position.y = -6.3;  
  
// ...  
  
let geometry = new THREE.BoxGeometry(5, 10, 1);
```

O funcionamento dos filtros citados, além de outros, são discutidos a seguir, assim como os resultados obtidos.

## **NearestFilter**

Na magnificação, como já mencionado o filtro *NearestFilter*, irá produzir um resultado *pixelizado* na imagem. Isso está diretamente relacionado ao número de fragmentos que estão sobre um mesmo *texel*, quanto maior o número de fragmentos mais *pixelizado* será o resultado. Vale ressaltar também que, quando a textura está sendo projetada sobre um objeto que ocupa mais *pixels* do que a textura original, tem-se que alguns *pixels* do objeto serão coloridos de acordo com o *texel* mais próximo da textura, justificando o efeito *pixelizado* na imagem.

Na minificação, assim como descrito acima para cada fragmento de textura que está em cima de determinado pixel é gerado um par de coordenadas de textura, que será projetado sobre a textura e o *texel* mais próximo deste ponto será escolhido para colorir o pixel final. Com isso, é possível notar a presença de ruídos ao olhar para a imagem mais afastada.

A parte do código fonte a ser alterada corresponde estas linhas a seguir.

```
texture.magFilter = THREE.NearestFilter; // Magnificação com filtro vizinho  
mais próximo  
texture.minFilter = THREE.NearestFilter; // Minificação com filtro vizinho mais  
próximo  
texture.anisotropy = 1; // índice de anisotropia
```

O resultado é visto na figura a seguir. Neste exemplo, é notável a presença do efeito de pixelização da textura nos fragmentos na região inferior da tela, além do ruído elevado no topo da figura, características já mencionadas do filtro vizinho mais próximo.



## LinearFilter

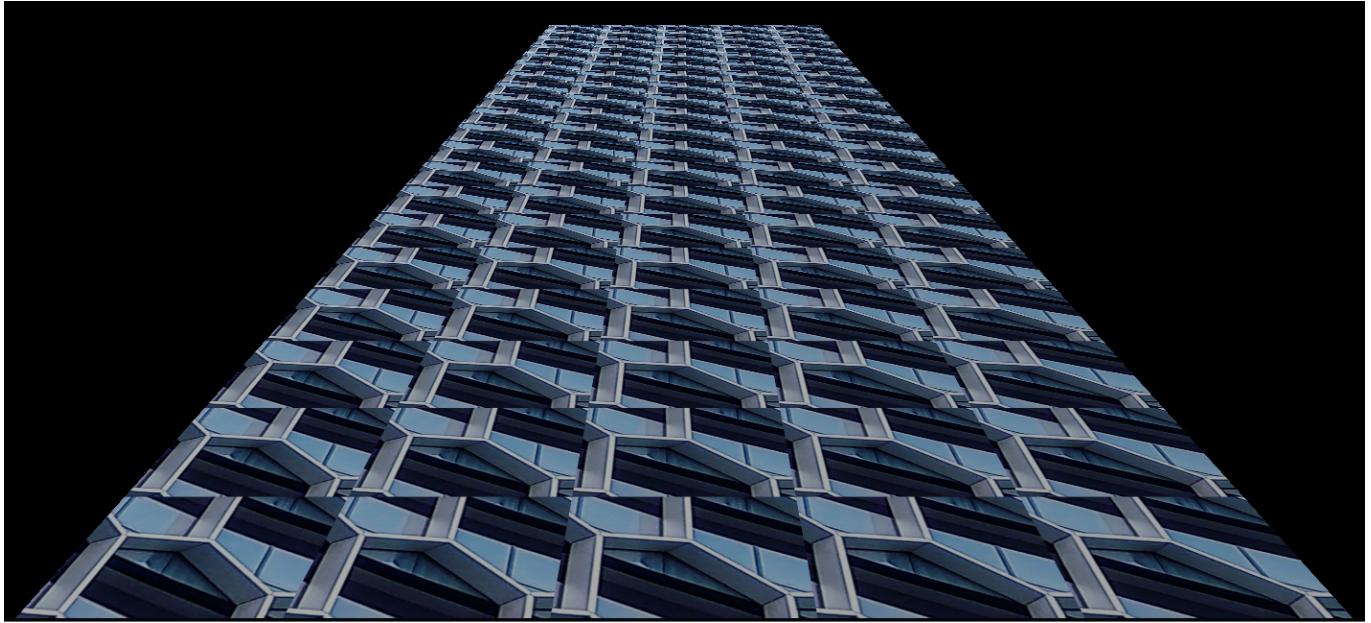
Na magnificação, a interpolação bilinear (às vezes chamada interpolação linear) avalia os quatro *texels* da textura que estão mais próximos do *pixel* do objeto que deseja-se colorir. É importante destacar que o *texel* da textura que estiver mais próximo do *pixel* do objeto influenciará mais na cor do *pixel*, do que o mais distante. Em síntese, este tipo de filtragem encontra os quatro *texels* vizinhos e interpola linearmente em duas dimensões para encontrar um valor combinado para o pixel, produzindo um resultado mais desfocado. Outro ponto a ser citado é que, este método possui um custo maior do que o *NearestFilter*, pois, requer uma avaliação de quatro *texels* ao invés de um, no processo de decisão da cor do pixel do objeto.

O filtro de interpolação linear é apenas um pouco melhor do que o vizinho mais próximo abordagem para minificação. Pois como mencionado, ele combina quatro texels em vez de usar apenas um, mas quando um pixel é influenciado por mais de quatro texels, o filtro logo falha e produz *aliasing*.

A parte do código fonte a ser alterada corresponde estas linhas a seguir.

```
texture.magFilter = THREE.LinearFilter; // Magnificação com filtro bilinear  
texture.minFilter = THREE.LinearFilter; // Minificação com filtro bilinear  
texture.anisotropy = 1; // índice de anisotropia
```

A seguir é exibido o resultado. O efeito de suavização da pixelização é mais notável na parte inferior da tela, onde o efeito de magnificação é mais evidente, no topo da figura matém-se um certo ruído.



## **Mipmap e Anisotropia**

O método mais popular de suavização de texturas é chamado de *mipmap*. Representa um processo no qual a textura original é filtrada repetidamente em imagens menores. Quando o filtro de minimização de *mipmap* é usado, a textura original é aumentada com um conjunto de versões menores da textura antes que a renderização real ocorra.

A textura (no nível zero) é reduzida para um quarto da área original, com cada novo valor de *texel* frequentemente calculado como a média de quatro *texels* vizinhos na textura original. A nova textura de nível um às vezes é chamada de subtextura da textura original. A redução é realizada recursivamente até que uma ou ambas as dimensões da textura sejam iguais a um *texel*. O conjunto de imagens como um todo costuma ser chamado de cadeia de *mipmap*.

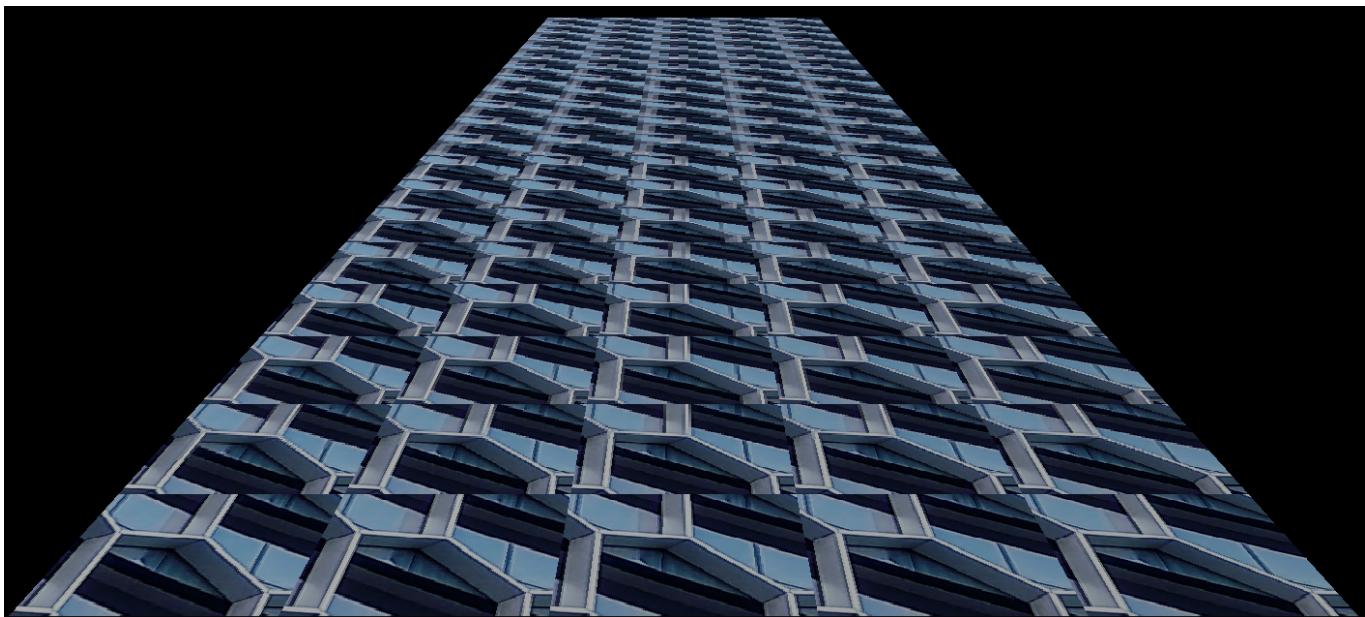
E o conceito de anisotropia se refere ao número de amostras obtidas ao longo do eixo através do pixel que possui a maior densidade de *texels*. Um valor mais alto fornece um resultado menos borrado do que um *mipmap* básico, ao custo de mais amostras de textura sendo usadas.

## **NearestMipmapNearestFilter**

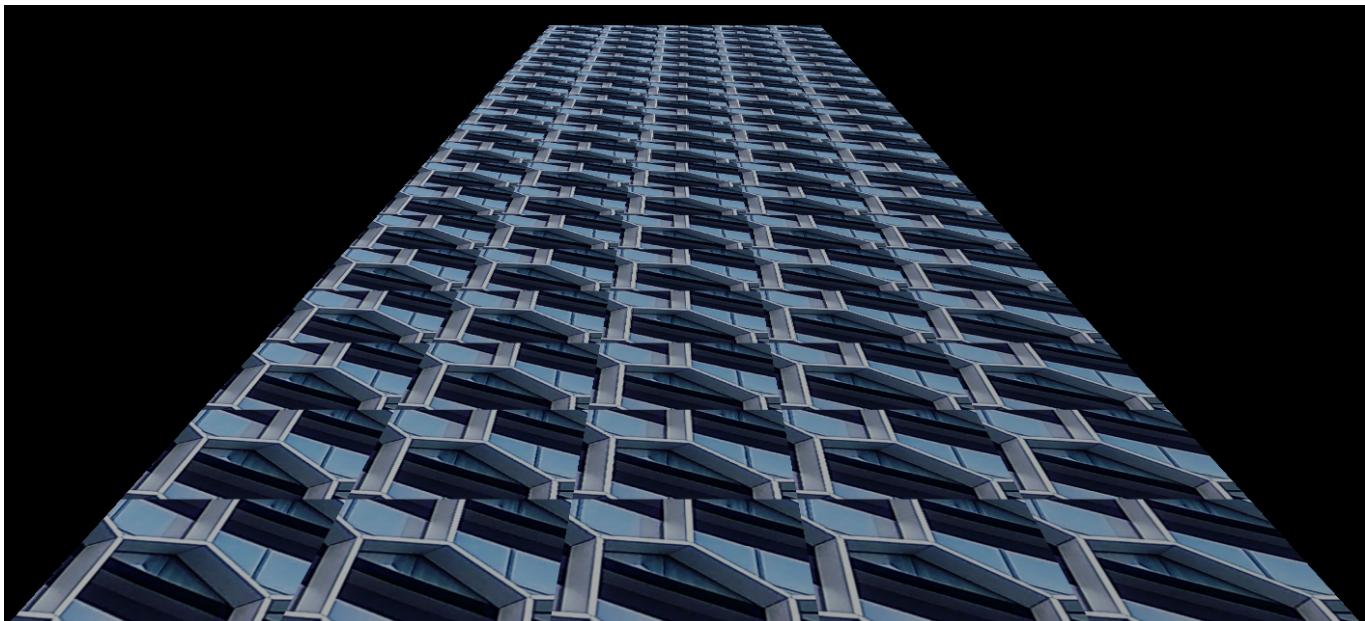
Em resumo este filtro consiste na escolha do *mip* apropriado e, em seguida, escolha de um pixel, utilizando o critério do vizinho mais próximo.

Abaixo é exibido o trecho do código fonte alterado para a aplicação do filtro em questão na minificação e o bilinear na magnificação.

```
texture.magFilter = THREE.LinearFilter; // Magnificação com filtro bilinear  
texture.minFilter = THREE.NearestMipmapNearestFilter;  
texture.anisotropy = 1; // índice de anisotropia
```



É possível notar na imagem acima o ruído produzido pela pixelização da textura. Ao aumentar a quantidade de amostras, isto é, aplicar o filtro anisotrópico com índice 4, é possível notar uma melhora neste quesito. Isto é feito ao aplicar tal alteração ao código fonte: `texture.anisotropy = 4;`. O resultado é exibido abaixo.



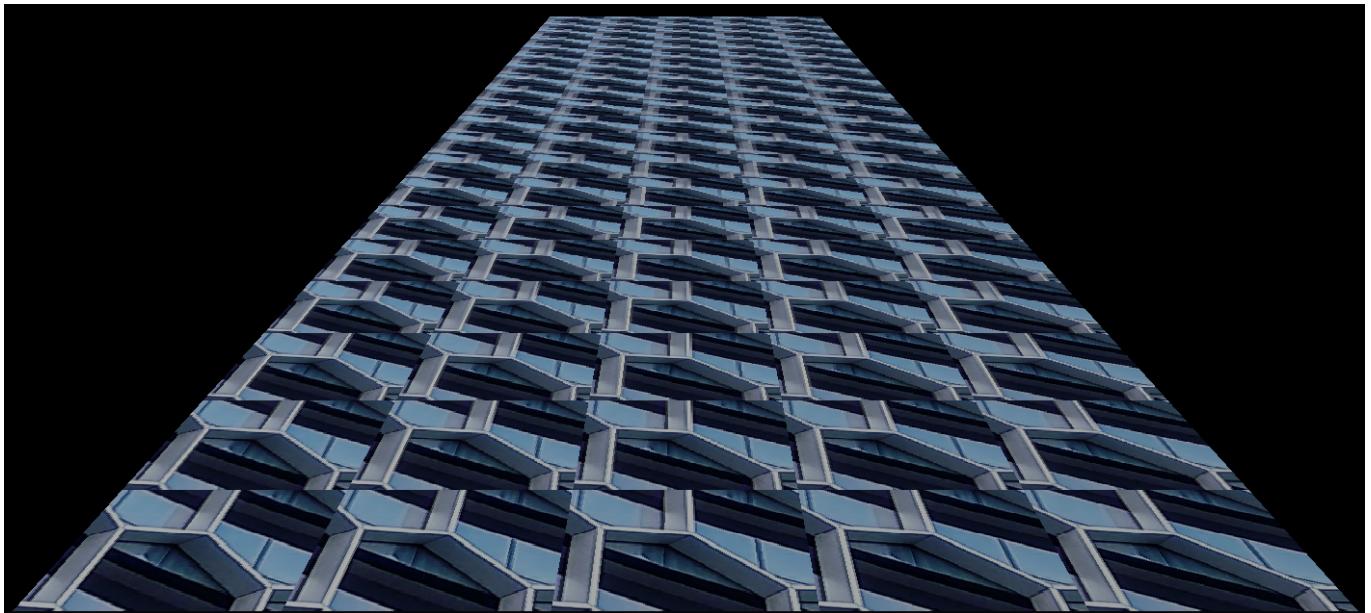
Ao aplicar uma filtragem anisotrópica com 16 amostras (`texture.anisotropy = 16;`) não é perceptível melhorias em relação à filtragem anterior. O resultado é exibido abaixo.



## ***NearestMipmapLinearFilter***

Basicamente este filtro realiza a escolha de dois *mips*, escolhe um *pixel* de cada e interpola linearmente os dois.

```
texture.magFilter = THREE.LinearFilter; // Magnificação com filtro bilinear  
texture.minFilter = THREE.NearestMipmapLinearFilter;  
texture.anisotropy = 1; // índice de anisotropia
```



É possível notar que assim como na filtragem anterior que houve suavização do ruído ao realizar a alteração ao código fonte: `texture.anisotropy = 4;`. O resultado é exibido abaixo.



Assim como no exemplo anterior, ao aplicar uma filtragem anisotrópica com 16 amostras (`texture.anisotropy = 16;`) não é perceptível melhoras em relação ao mesmo filtro com 4 amostras. O resultado é exibido abaixo.



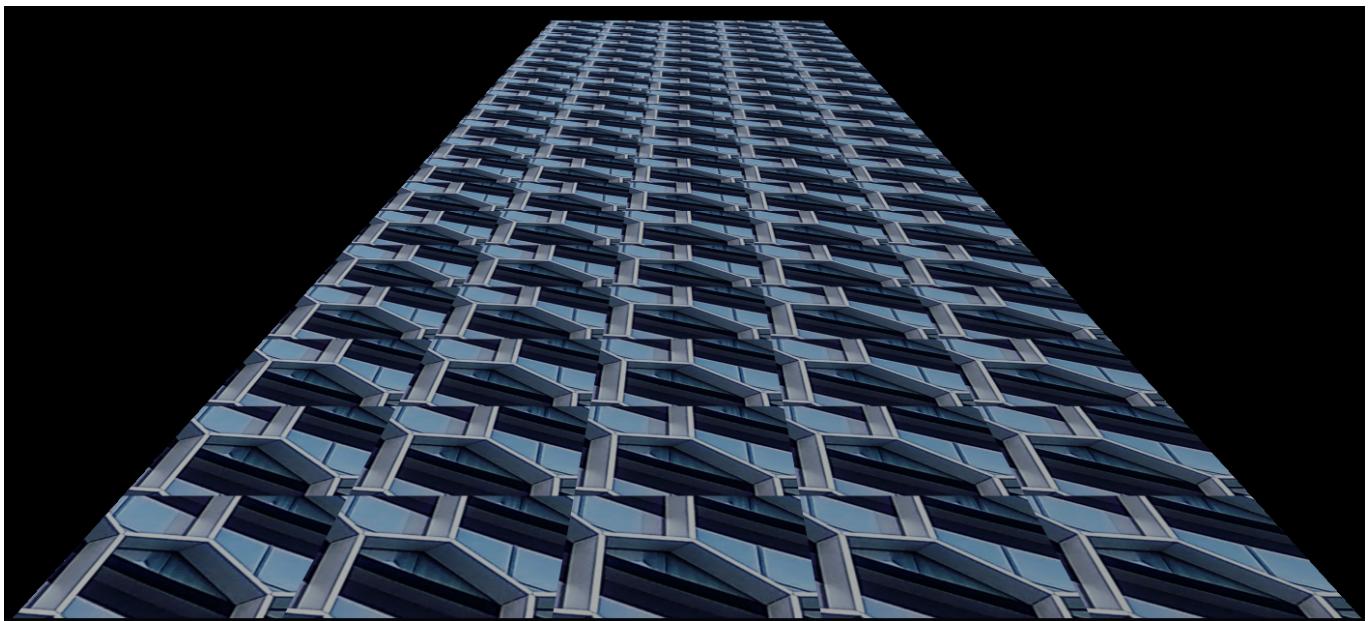
## ***LinearMipmapNearestFilter***

Este filtro efetua a escolha do *mip* apropriado, em seguida, escolhe quatro *pixels* e realiza a interpolação bilinear destes.

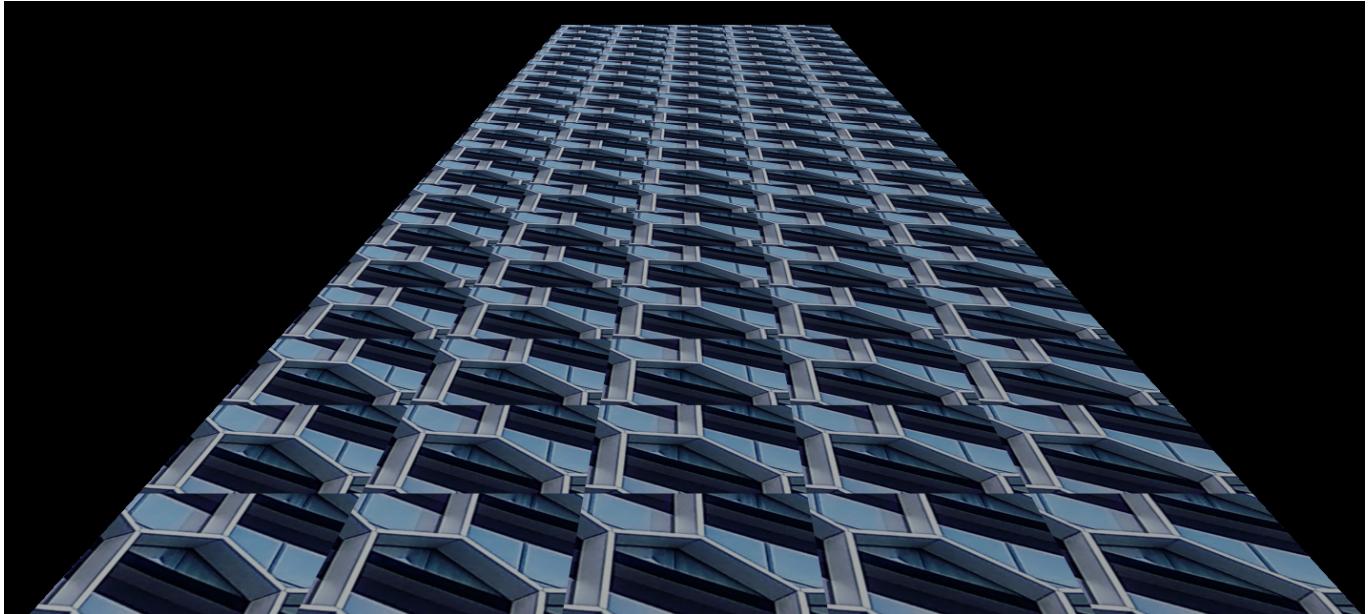
```
texture.magFilter = THREE.LinearFilter; // Magnificação com filtro bilinear  
texture.minFilter = THREE.LinearMipmapNearestFilter;  
texture.anisotropy = 1; // índice de anisotropia
```



Neste tipo de filtragem é ainda mais notável a suavização do ruído ao realizar a alteração ao código fonte: `texture.anisotropy = 4;` aumentado o nível de amostragem do filtro, principalmente na região superior da tela. O resultado é exibido abaixo.



Desta vez, ao aplicar uma filtragem anisotrópica com 16 amostras (`texture.anisotropy = 16;`) é perceptível uma pequena melhora em relação ao mesmo filtro com 4 amostras. O resultado é exibido abaixo.



## ***LinearMipmapLinearFilter***

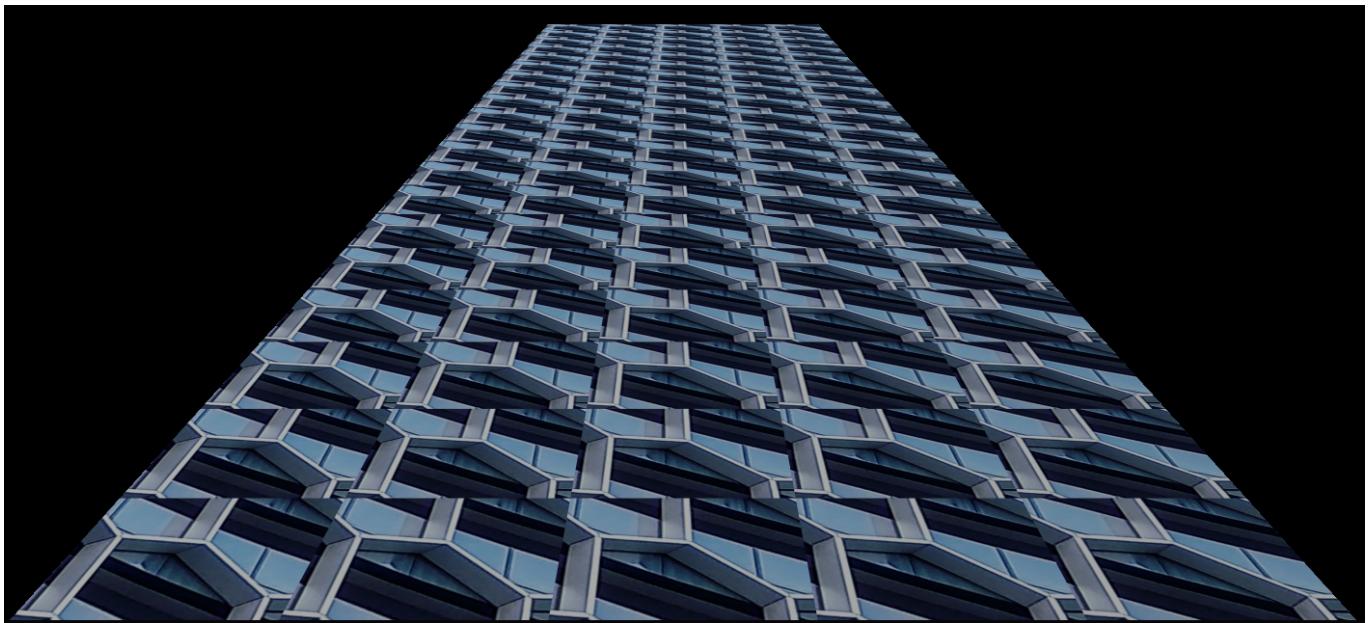
Por fim, o filtro mencionado faz a escolha de dois *mips*, e efetiva a escolha de quatro *pixels* de cada e interpola todos os oito em um *pixel*. Também conhecido como filtro de interpolação trilinear.

```
texture.magFilter = THREE.LinearFilter; // Magnificação com filtro bilinear  
texture.minFilter = THREE.LinearMipmapLinearFilter;  
texture.anisotropy = 1; // índice de anisotropia
```

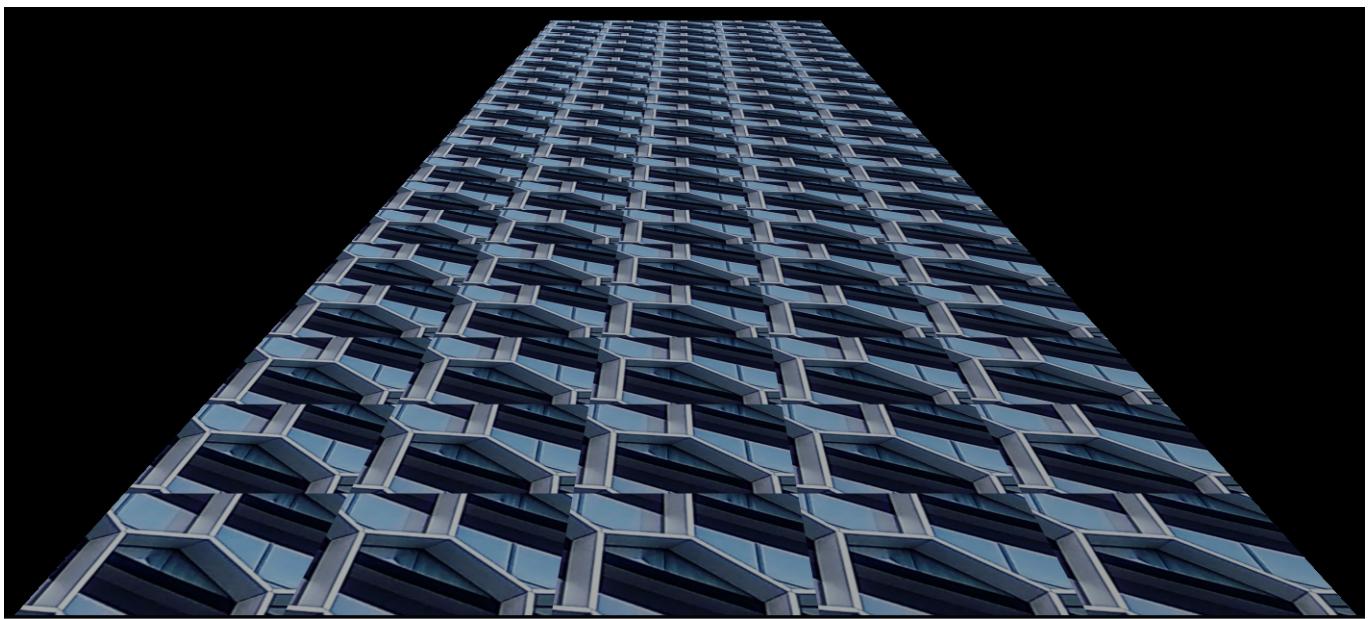


Assim como na filtragem mencionada anteriormente, neste tipo de filtragem é bastante notável a suavização do ruído ao realizar a alteração ao código fonte: `texture.anisotropy`

`= 4;` aumentado o nível de amostragem do filtro, principalmente na região superior da tela. O resultado é exibido abaixo.



Mais uma vez, ao aplicar uma filtragem anisotrópica com 16 amostras (`(texture.anisotropy = 16;)`) é perceptível uma pequena melhora em relação ao mesmo filtro com 4 amostras. O resultado é exibido abaixo.



## Dificuldades e possíveis melhorias

A principal dificuldade encontrada se dá pelo fato de que a comparação é de caráter subjetiva, dizendo respeito à percepção do observador quanto as imagens produzidas pela aplicação da filtragem de figuras.

A melhoria implementada foi a alteração da geometria e posição da câmera para melhor percepção da aplicação dos efeitos.

## Referências

Livro do Peter Shirley, capítulo 6.

[Documentação do Three.js](#)