

Introdução à Computação Gráfica

Atividade Prática 4 - Mapeamento e Filtragem de Texturas

Egídio Neto Alves de Araújo – 20180087632
Francisco Siqueira Carneiro da Cunha Neto – 20190029015

16 de novembro de 2021

1 Atividade Desenvolvida

Nessa atividade foi realizada uma análise comparativa dos seguintes tipos de filtragem de textura:

- Nearest Neighbor (Seção 2)
- Bilinear (Seção 3)
- Mipmapping (Seção 4)
- Anisotrópico (Seção 5)

Afim de obter capturas de tela com exemplos de cada filtragem, a biblioteca ThreeJS foi utilizada para renderizar um cubo com textura e utilizando cada um dos filtros, seguindo o código disponível no template fornecido.

2 Nearest Neighbor

O método *Nearest Neighbor* é o mais simples dos analisados nesta atividade. Nele, a cor de cada fragmento é decidida simplesmente arredondando suas coordenadas u e v , isto é, escolhendo o texel que é seu vizinho mais próximo (dai o nome). Esta filtragem não se preocupa em tratar casos de magnificação ou minificação e portanto renders que a utilizam são suscetíveis tanto a um efeito *pixelado* quando vários fragmentos são mapeados para um mesmo texel (magnificação, exemplificada na Fig. 1) quanto a um efeito de “ruído” quando vários texels são projetados no mesmo fragmento (minificação, exemplificado na Fig. 2).

Nas próximas seções comparamos a filtragem *Nearest Neighbor* com outras que tentam tratar esses dois problemas.

3 Bilinear

A filtragem de textura *bilinear* é alcançada realizando interpolações linear nos texels próximos às coordenadas calculadas pela função de projeção dos fragmentos da geometria tanto no eixo X, quanto no eixo Y. Garantindo à textura uma aparência mais suave, em contraste ao método do *nearest neighbor*. Abaixo (Fig. 3 e 4) vemos uma comparação dos métodos citados.

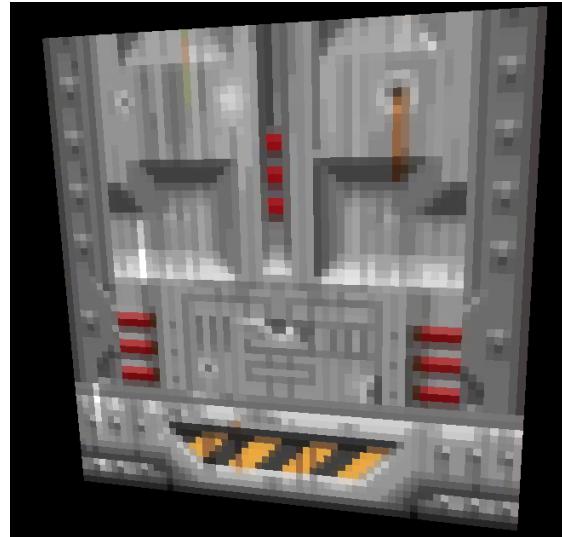


Figura 1: Exemplo de magnificação gerado pelo Nearest Neighbor.

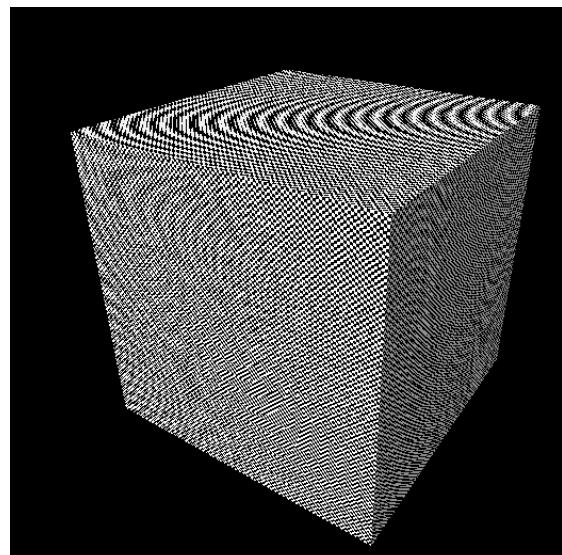


Figura 2: Exemplo de minificação gerado pelo Nearest Neighbor.

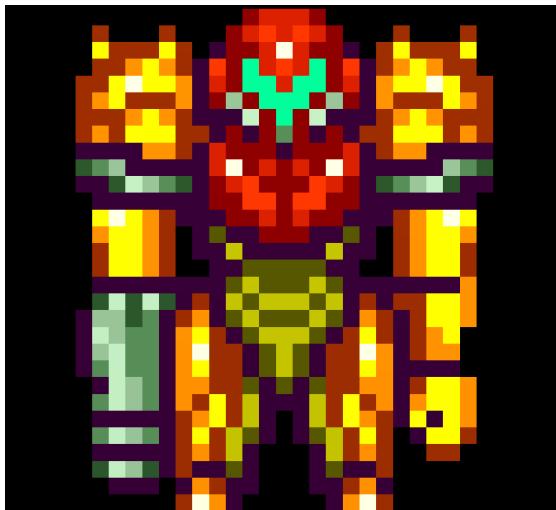


Figura 3: Imagem feita utilizando o método de filtragem Nearest Neighbor



Figura 4: Imagem feita utilizando o método de filtragem Bilinear

A diferença é aparente, enquanto no caso da filtragem *Nearest Neighbor* podemos claramente ver os *texels* individuais do sprite [1] da personagem Samus [2], no caso da filtragem *bilinear* temos a interpolação das cores do *texels* deixando a textura com uma aparência mais suave e melhorando alguns detalhes como o brilho espelhado na armadura que passou a ter uma aparência mais natural, como também os espinhos na ombreira.

Entretanto, a filtragem *bilinear* pode fazer com que detalhes pequenos de uma imagem sejam perdidos ao serem interpolados. A suavização da imagem causa um efeito de desfoco que faz certas texturas não ficarem da maneira que foram originalmente pensadas, portanto, detalhes menores podem ser removidos ou mitigados, principalmente quando ocorre uma alta saturação de cores ao redor, isso é especialmente comum em texturas com uma resolução maior e que possuem detalhes com poucos *texels*. Essa característica é um problema especialmente em jogos retrô, onde a arte foi feita com uma resolução específica em mente e em certos jogos

um *pixel* pode fazer diferença na imagem final. Abaixo temos um exemplo do problema citado utilizando um asset[3] do jogo DOOM[4](Fig. 5 e 6), onde apesar da imagem que utiliza o filtro *bilinear* possuir uma melhor representação de sombra e consequentemente profundidade, detalhes do portão passaram a ficar com uma aparência desfocada e alguns detalhes menores sumiram quase que completamente, em especial no centro e parte inferior do portão.

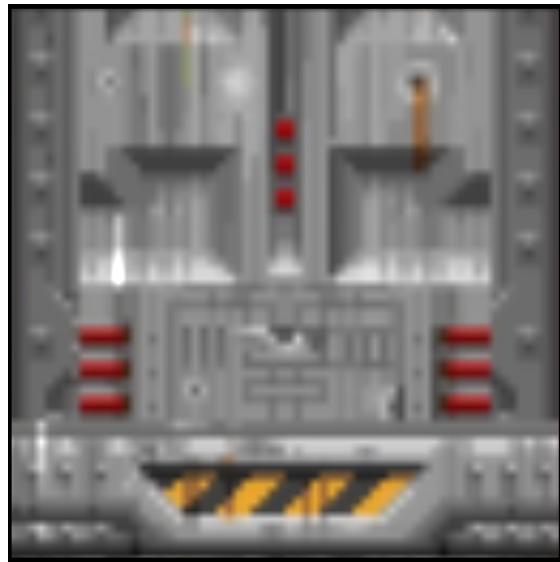


Figura 5: Imagem mostrando o efeito de desfoco nos detalhes

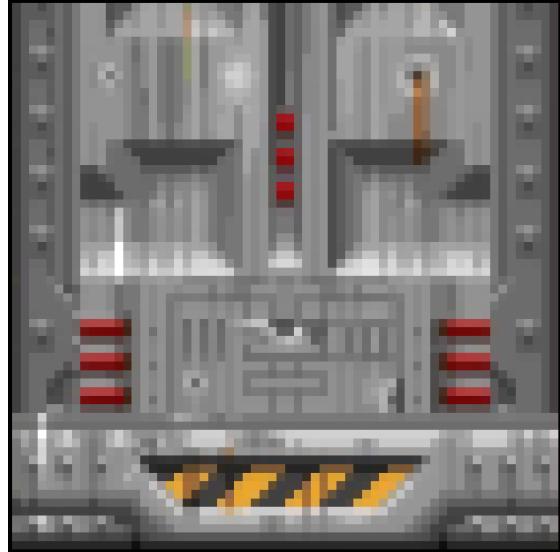


Figura 6: Imagem do portão com detalhes mais vivos

4 Mipmapping

A técnica *Mipmapping* foi desenvolvida com o objetivo de mitigar os efeitos da minificação. Ela funciona gerando (previamente à renderização) várias texturas com uma resolução gradativamente menor que a original, cada *texel* destas calculado pelo média de 4 *texels* vizinhos da textura anterior. Dessa

forma, um fragmento que teria de escolher entre muitos texels com pouca diferença entre eles no *Nearest Neighbor*, escolherá ao invés disso a média das cores desses texels, reduzindo a minificação.

Na Fig. 7, podemos ver um exemplo dessa filtragem. Podemos compará-la com a Fig. 8 (idêntica a Fig. 2, replicada aqui a título de comparação) que mostra o mesmo *render* usando *Nearest Neighbor*.

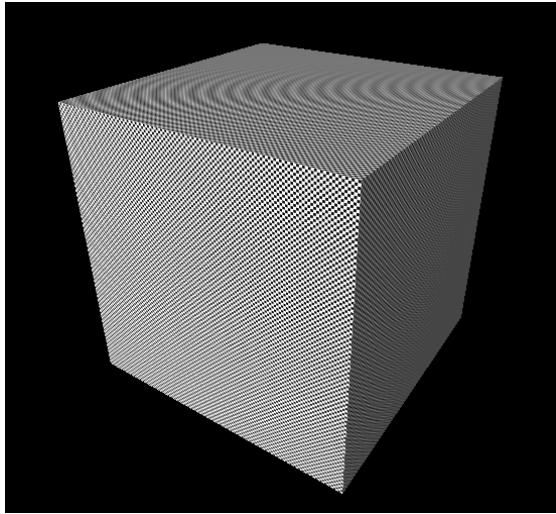


Figura 7: Objeto com textura utilizando filtragem Mipmapping trilinear.

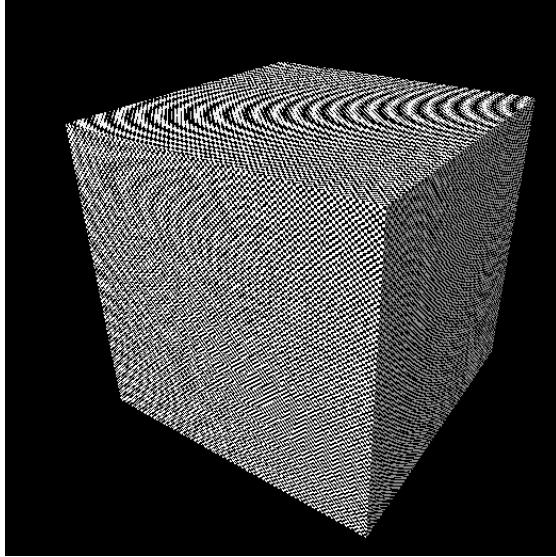


Figura 8: Exemplo de minificação gerado pelo *Nearest Neighbor*.

É imediatamente notável que a aparência “ruidosa” da imagem é quase completamente eliminada. Os fragmentos no *Nearest Neighbor* não conseguem seguir o padrão *checkerboard* da textura, mas continuam assumindo apenas as cores brancas e pretas originais, levando a um certo “caos”. Já no *Mipmapping*, esses fragmentos assumem cinzas, cores intermediárias às originais, dando a textura uma aparência mais suave. Essa efeito simula uma impressão de desfoque pela distância, como acontece no mundo natural em que cores,

quando observadas a distância, começam a se misturar e fica difícil determinar cada cor individual (esse mesmo efeito que permite o funcionamento de um pixel).

Esse tipo de filtragem, contudo, tem um custo computacional mais alto do que um simples *Nearest Neighbor*, sendo necessário 33% de memória adicional para cada textura para armazenar suas subdivisões.

Existem algumas variações da filtragem, que determinam principalmente duas questões: qual texel escolher para um fragmento na textura reduzida, e o que fazer quando a quantidade de texels para um fragmento está numa proporção entre as texturas disponíveis. Para ambas as questões existem duas opções: escolher o vizinho mais próximo, ou realizar uma interpolação (bilinear no primeiro caso e linear no segundo).

A Fig. 7 reflete o uso de interpolação bilinear na escolha do texel e interpolação linear na escolha da textura. Essa combinação é conhecida como filtragem trilinear, e em geral é a que apresenta melhores resultados mas com o maior custo computacional. Isso porque na filtragem trilinear é necessário fazer 8 amostras de texel para cada fragmento que se deseja colorir.

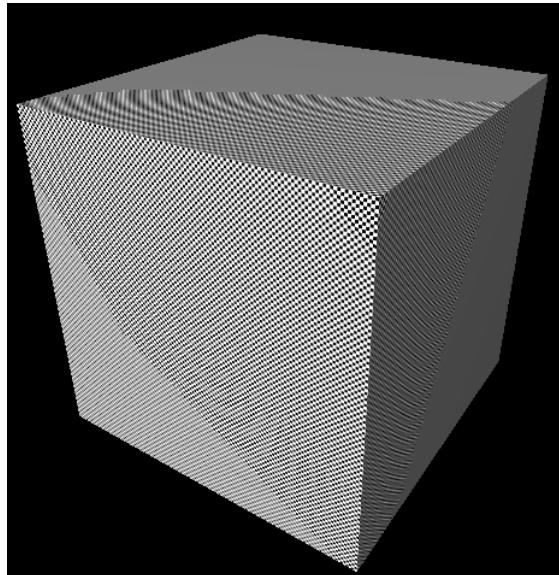


Figura 9: Filtragem Mipmapping NN na escolha de textura e linear na escolha do texel.

Um exemplo de *render* usando interpolação bilinear na escolha do texel e vizinho mais próximo na escolha da textura pode ser visto na Fig. 9. É notável que existe uma suavização na aparência do cubo mas, devido ao método de escolha da textura, após uma certa distância o objeto é colorido inteiramente pelo mesmo tom de cinza. Como não há interpolação entre diferentes níveis de textura, a partir de um certo ponto se é escolhida uma textura cujos texels são todos da mesma cor, e por isso há esse contraste forte na cor do cubo. Essa variação performa, computacionalmente falando, melhor do que a filtragem trilinear, necessitando apenas de 4 amostras por fragmento.

A Fig. 10 mostra um *render* usando vizinho mais próximo na escolha do texel e interpolação linear na escolha da textura. Comparando com o caso anterior, pode-se perceber que há

uma mudança mais gradual até o cubo ficar completamente cinza, simulando melhor a visão real que perde o foco gradativamente com a distância. Contudo, o método de escolha do texel falha em reduzir a impressão de ruído na face frontal do objeto. Essa variação só precisa de duas amostras de texels por fragmento.

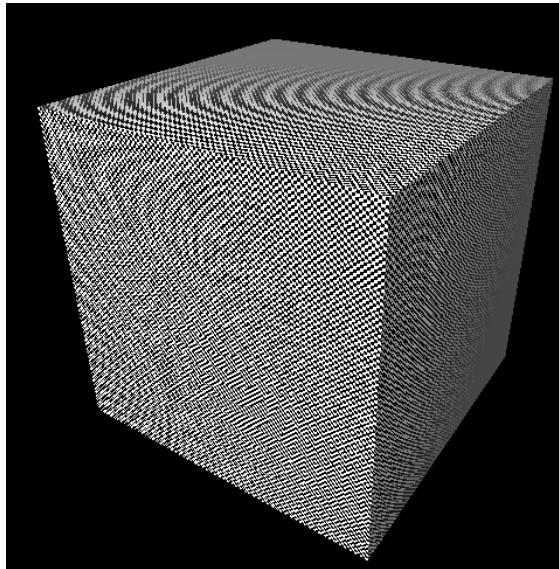


Figura 10: Filtragem Mipmapping linear na escolha de textura e NN na escolha do texel.

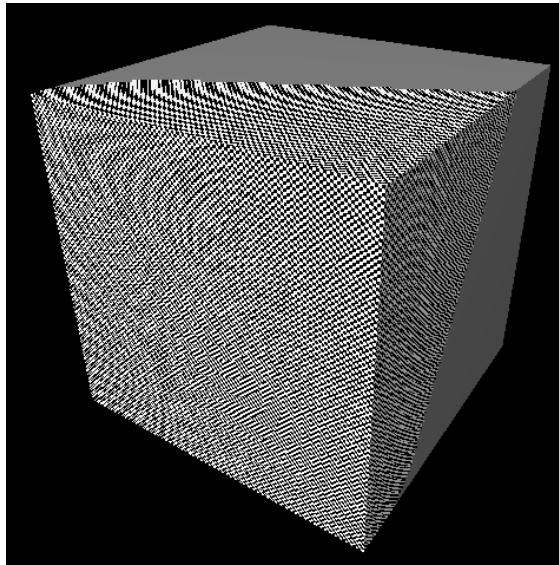


Figura 11: Filtragem Mipmapping NN na escolha de textura e NN na escolha do texel.

Finalmente, a Fig. 11 exemplifica o *Mipmapping* com ambos os parâmetros de escolha sendo “vizinho mais próximo”. Nesse caso, permanece tanto o ruído da face frontal, quanto a mudança abrupta de cores. Esses parâmetros são os mais performáticos do *Mipmapping*, amostrando apenas um texel por fragmento.

5 Anisotrópico

A filtragem *anisotrópica* é feita quando a projeção do *pixel* na textura tem a forma de um trapézio, causada pelo fato da geometria não estar reta em relação à câmera, fazendo com que a técnica de *mipmapping* produza resultados insatisfatórios. A filtragem *anisotrópica* identifica o maior lado do trapézio gerado pela projeção e aplica uma das técnicas de filtragem citadas anteriormente de acordo com a razão do maior lado com o menor, com o *mipmapping* sendo a técnica que produz resultados mais fiéis em troca de um custo computacional maior. Abaixo temos a comparação de uma geometria com a filtragem de *mipmapping* porém sem a filtragem *anisotrópica* (Fig. 12) e outra imagem com a filtragem *anisotrópica* configurada para ir até o nível $x6$ (Fig. 13) e para o nível $x16$ (Fig. 14) utilizando *mipmapping*.

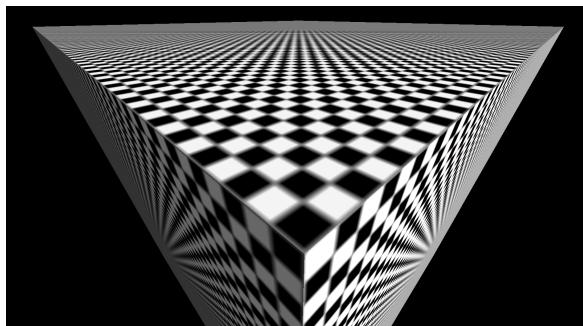


Figura 12: Imagem feita utilizando o método de filtragem de mipmapping sem filtragem anisotrópica

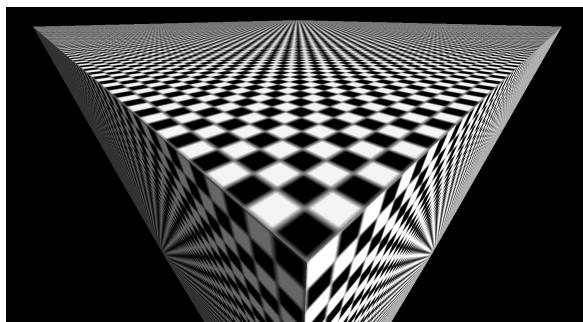


Figura 13: Imagem feita utilizando o método de filtragem de mipmapping e com filtragem anisotrópica em $x6$

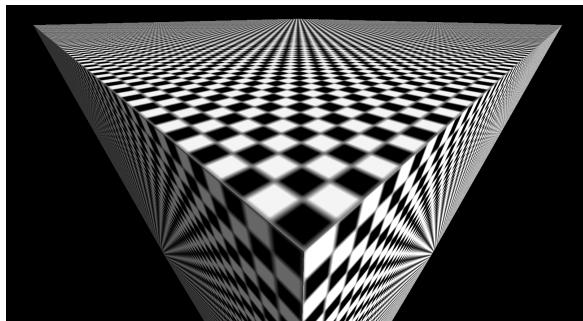


Figura 14: Imagem feita utilizando o método de filtragem de mipmapping e com filtragem anisotrópica em $x16$

A diferença pode ser percebida olhando os pontos mais distantes da geometria, onde a filtragem de *mipmapping* gradualmente torna os *pixels* cinza, já a filtragem *anisotrópica* ainda mantém o aspecto de *checkerboard* da imagem, isolando a perda de detalhe para regiões bem mais afastadas na geometria.

Apesar da filtragem *anisotrópica* produzir resultados muito bons quando configurada para fazer diversas filtragens o custo computacional é muito acentuado para ganhos que talvez não sejam muito perceptíveis ao observador. Com um caso extremo de que possa ser necessário realizar 128 consultas de textura para apenas um fragmento quando configurada para ir até $x16$, em comparação com apenas 8 consultas de texturas se for utilizado apenas o *mipmapping*.

6 Conclusões

As filtragens de textura buscam resolver principalmente dois problemas: a magnificação e a minificação. A partir da análise realizada podemos ver que, em maioria, elas são muito bem sucedidas nesse objetivo. Também fica claro que cada tipo de filtragem de textura possui vantagens e desvantagens, em geral ocorrendo uma troca de poder computacional por suavidade nas imagens. Assim, a decisão de que tipo de filtragem utilizar depende da aplicação, do poder computacional disponível, e da proeminência de magnificação e minificação nas imagens geradas.

Referências

- [1] H. Matsuoka, M. Mashimo e H. Kimura. “Super Metroid.” (1994), endereço: <https://www.spriters-resource.com/fullview/31614/> (acesso em 14/11/2021).
- [2] M. Kano, *Super Metroid*. Nintendo Co. Ltd, 1994.
- [3] J. Romero. “DOOM wall textures.” (2014), endereço: <https://twitter.com/romero/status/542945951720038400> (acesso em 14/11/2021).
- [4] S. Petersen, J. Romero, S. Green e T. Hall, *DOOM*. id Software LLC, 1993.