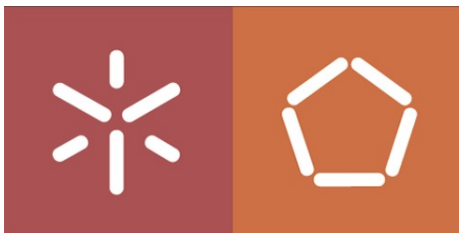
Universidade do Minho

Departamento de Informática



MEI - Mestrado em Engenharia Informática

Computação Gráfica

Visualização em Tempo Real

Grupo 4

Trabalho Prático

André Araújo PG47842

Diana Ferreira PG46529

Nuno Mata PG44420

22 Junho
2021/2022

Conteúdo

1 Introdução

Este relatório surge no âmbito da Unidade Curricular Visualização em Tempo Real incorporada no mestrado de Engenharia Informática, mais concretamente no perfil de Computação Gráfica.

O documento diz respeito ao tema **Renderização não fotorrealista**, que é uma área da Computação Gráfica onde engenheiros e informáticos tentam animar e transformar objetos em pinturas, desenhos, desenhos animados e outras opções que não apresentam fotorrealismo.

Existem várias técnicas para a criação de imagens que não apresentam fotorrealismo, das quais se enquadram as técnicas de rendering. Dentro das técnicas de rendering, existe uma categoria que se concentra na criação de imagens estilo "cartoon", é nesta categoria que surge a técnica principal elaborada no nosso projeto, o *Toon Shading*. O ***Toon Shading*** é um tipo de renderização não fotorrealista desenvolvido para fazer a transformação de objetos 3D para que pareçam planos (2D) através da redução de cores nas sombras em vez de um gradiente de sombra. Foi ainda implementada a técnica de *Toon Shading per pixel* que utiliza o modelo de iluminação de *Phong*, lecionado nas aulas, na computação da iluminação do objeto de forma a determinar informações mais realistas dos pontos de iluminação na superfície do objeto.

Ao longo do documento vão ser explicados os conceitos aqui mencionados, de forma a melhor se compreender a implementação do algoritmo. Vão ser também demonstradas as várias fases de desenvolvimento para se compreender a progressão do projeto, e serão ainda expostas dificuldades encontradas na implementação de algumas funcionalidades. Por fim são apresentados os resultados obtidos e também é demonstrada a transformação obtida no objeto inicial com o algoritmo desenvolvido, para a criação do mesmo objeto com um estilo "cartoon".

2 Contextualização

Este capítulo tem o intuito de explicar sobre o que se trata o tema escolhido, de **Renderização não fotorrealista** (*NPR*), e também alguns aspetos e decisões mais relevantes tomadas durante a implementação e desenvolvimento do projeto. Vão ainda ser abordadas, mais em detalhe, algumas das tecnologias utilizadas e a razão pela sua escolha comparativamente a outras opções semelhantes que existem.

Resumindo a poucas palavras, a renderização não fotorrealista é o processo de geração de imagens que não aspira alcançar o realismo. A *NPR* é uma área da computação gráfica que visa permitir uma ampla variedade de estilos de arte expressivos para o desenvolvimento de arte digital, sendo que o seu objetivo é oposto àquele da computação gráfica tradicional que se concentra em produzir imagens fotorrealistas. As ideias-chave na criação de imagens não fotorrealistas são principalmente: (1) a abstração - remover detalhes sem importância -, (2) a ambiguidade - remover detalhes importantes deliberadamente -, e (3) o ênfase - destacar detalhes importantes. Alguns exemplos da aplicação de NPR pode ser vista em filmes de animação (muito utilizada pela *Disney* e *Pixar*), também muito utilizada em videojogos, animações 2D e 3D, desenhos animados, etc.

As técnicas para criação de imagens não fotorrealistas podem ser variadas e combinadas. Geralmente, distinguem-se pelo grau de automatismo (desde as totalmente criadas pelo computador até às que resultam do trabalho do artista) e pertencem a um destes grupos:

- *Técnicas de renderização*, onde o efeito não fotorrealista é criado através do próprio motor de renderização;
- *Técnicas de shading*, onde o efeito não fotorrealista é criado através dos shaders utilizados;
- *Técnicas de pós-processamento*, onde o efeito não fotorrealista é criado através de filtros ou de manipulação em pós-produção.

Dentro das técnicas de renderização existem várias categorias que pretendem diferenciar entre as artes/estilos da imagem que vai ser produzida. Estas diferenças podem ser evidentes em parâmetros como os traços, texturas, tons, contornos e escalas. As categorias de NPR são:

- *Pen and ink illustrations:* Categoria que se concentra na utilização de arte nas linhas, traços e cruzamento de linhas para criar imagens;
- *Technical illustrations:* Categoria que utiliza contornos e sombras com coloração mate, principalmente utilizado em imagens para manuais técnicos;
- *Scientific visualization:* Categoria que através do desenho de linhas e técnicas de *splatting* (em computação gráfica, o splatting é um método para combinar diferentes texturas) pretende aumentar a precisão da imagem, usado, por exemplo, em visualizações científicas;
- *Painterly rendering:* Categoria que através da utilização de cores, traços, texturas e outras técnicas exclusivas da arte da pintura cria uma imagem com o estilo de uma pintura;
- *Cartoon rendering:* Categoria que através da utilização de sombras, distorção e desenho de contornos consegue alcançar efeitos semelhantes aos de um desenho animado ou cartoon. Nesta categoria que é utilizada a técnica *Toon Shading*, que é usada para a criação de imagens com um "aspeto de cartoon".

No nosso caso, como mencionado no capítulo anterior, dentro das técnicas de NPR usamos o *Toon Shading*. O *Toon Shading* funciona dividindo o modelo em três regiões: realces, áreas iluminadas e áreas não iluminadas, e sombreando-as com cores "planas", retirando o gradiente subtil nas cores do objeto (que é um atributo de um objeto realista). Assim, o objeto vai ter uma forma mais "achatada"(*flattened*) o que vai reduzir o seu aspeto de profundidade aproximando-se de um objeto com aspeto 2D.

3 Implementação

Neste capítulo é retratado todo o processo de *Non Photorealistic Rendering* efetuado. Devido à complexidade do problema e à familiarização com a técnica escolhida, optamos desde início pela utilização de *Toon Shading*, também conhecida como *Cel Shading*. Em todo o processo de desenvolvimento foi utilizada a linguagem GLSL, recorrendo-se, ainda, a ferramentas fornecidas pelos docentes para auxílio e facilitismo, como o *software* Nau3D e a respetiva documentação.

No que diz respeito à implementação da técnica *Toon Shading*, inicialmente, optamos pela reutilização do código da matéria em questão fornecido pelos docentes, nomeadamente *Toon Shading Per Vertex* e *Toon Shading Per Pixel* retratados nos próximos subcapítulos. Como já havia sido mencionado, a técnica *Toon Shading* é um conjunto de outras técnicas inseridas na renderização de imagens 3D de modo a que o resultado final se assemelhe a desenhos 2D. Desta forma, pretende-se transformar o modelo 3D escolhido numa espécie de desenho 2D, utilizando *Outlines*, ou seja, contornos escuros, e uma iluminação menos realista, próxima da iluminação utilizada em desenhos animados.

3.1 *Modelo 3D*

O modelo 3D escolhido é do tipo 'obj' e contém uma textura 'jpg'. Na implementação inicial de *Toon Shading Per Pixel* e *Toon Shading Vertex* a textura não foi utilizada. No entanto, na versão final de *Toon Shading Per Pixel* já fizemos uso da mesma. A Figura 2 apresenta o modelo 3D escolhido com a respetiva textura colocada.

Figura 2: Modelo 3D selecionado: Vaso de Planta com a textura colocada.

3.2 *Toon Shading Per Vertex*

Inicialmente, por meio da reutilização de código fornecido pelos docentes da unidade curricular, aplicamos *Toon Shading per Vertex* ao nosso modelo 3D, sem a aplicação de textura. No entanto, o resultado obtido (Figura 3) não foi o expectável, ou seja, o resultado não foi tão próximo a um desenho 2D. Para a aplicação de *Toon Shading per Vertex*, foram implementados 2 *shaders*, chamados de *toonV.frag* e *toonV.vert*. Tal como o nome indicia, *Toon Shading Per Vertex* é efetuado no *Vertex Shader*, modificando cada vértice com base num algoritmo fixo e afetando cada vértice processado enquanto ativado.

Figura 3: Resultado obtido com *Toon Shading Per Vertex*.

3.3 *Toon Shading Per Pixel*

Após o resultado não expectável obtido através de *Toon Shading per Vertex*, decidimos utilizar outra abordagem e optar por implementar outros dois *shaders* (*toonP.frag* e *toonP.vert*) para aplicar *Toon Shading per Pixel* ao nosso modelo 3D sem textura, de modo a obter um resultado mais próximo a um desenho 2D. O *Toon Shading Per Pixel* é efetuado no *Fragment Shader*, utiliza o modelo de iluminação de *Phong* e tem como objetivo calcular a cor por fragmento. Nesta abordagem, o *Vertex Shader* apenas prepara os dados para os cálculos localizados no *Fragment Shader*.

Para aplicar *Toon Shading Per Pixel*, no *Fragment Shader* especificamos uma estrutura de Dados que contém a normal, a direção da luz (vetor em direção à luz) e "*eye*", que é o vetor do ponto ao olho. Em relação ao *Vertex Shader*, este tem de calcular os vetores anteriores por vértice, para que sejam posteriormente interpolados e passados para o *Fragment Shader*. Para além disso, no *Fragment Shader*, também é declarada a variável *intensity*, que é calculada através do *dot product* entre o vetor normal e o vetor de direção de luz já transformados e normalizados. Posteriormente, a cor é calculada com base no valor de *intensity* das zonas do modelo 3D, sendo que no caso da *intensity* ser superior a 0.90, o *output* da cor é igual ao valor de *diffuse*, ou, por exemplo, no caso desta ser superior a 0.5, o *output* da cor é igual ao valor de *diffuse* multiplicado pelo respetivo 0.5.

Como é possível observar na Figura 4, obtivemos um resultado mais irrealista e mais semelhante a um desenho 2D do que a abordagem anteriormente apresentada (*Toon Shading Per Vertex*). Também é possível observar as diferentes cores resultantes, que foram calculadas a partir da intensidade de cada zona do modelo 3D.

Figura 4: Resultado obtido com *Toon Shading Per Pixel*.

No entanto, devido à simplicidade do projeto, decidimos aumentar a sua complexidade com a introdução de Texturas e *Outlines*. Desta forma, implementamos novamente mais 2 *shaders*, chamados de *shine.frag* e *shine.vert* com o intuito de aplicar *Toon Shading Per Pixel* ao nosso modelo 3D texturizado, ou seja, com texturas integradas, sendo *Toon Shading Per Pixel* a abordagem onde obtivemos melhores resultados inicialmente. Para este efeito, reutilizamos também o código relativo à colocação de texturas fornecido pelos docentes da unidade curricular. Ao código de *Toon Shading Per Pixel* anterior realizamos pequenas alterações, nomeadamente a introdução da variável *shininess*, implementada para que possa ser alterada em tempo real. Para além disso, apesar do cálculo da cor ser baseado ainda no valor de *intensity*, esta é agora calculada de forma diferente, ou seja, no caso do valor de *intensity* ser superior a 0.90, é necessário calcular o *half-vector*, a intensidade especular, sendo esta o *dot product* entre o *half-vector* e a normal já normalizados, e, por último, calcular o termo especular em *spec*, envolvendo a variável *shininess*.

3.3.1 Texturas

Para implementar a parte da texturização, no *shine.frag* declaramos, inicialmente, uma variável do tipo *uniform sampler 2D* e uma variável na estrutura *Data*, chamada *texCoord*, que tem como objetivo receber as coordenadas de textura. Para além disso, tivemos também de declarar uma nova variável *eColor*, que com as duas variáveis mencionadas anteriormente recebe/obtém a cor da textura. Em relação ao *output* da cor, a *eColor* passa a ser uma das componentes que conjuntamente com outras está a multiplicar com a *intensity*.

3.3.2 *Outlines*

Para a obtenção do tão desejado efeito de desenho, ou seja, efeito de *cartoon*, decidimos implementar *Outlines*, que são contornos numa cor específica ao longo das silhuetas do modelo 3D, que podem ser de cor preta ou de qualquer outra cor. No caso do nosso projeto, a cor do *Outline*, chamada "*outlineColor*"é uma componente que pode ser alterada em tempo real e apresentar a cor que quisermos consoante os valores escolhidos.

Existem várias técnicas para obter *Outlines*. Para simular este efeito declaramos duas espessuras gerais de contorno, sendo uma para áreas totalmente iluminadas ("*_LitOutlineT*") e outra para áreas não iluminadas ("*_UnlitOutlin*"), de acordo com o termo de reflexão difusa do modelo de reflexão de *Phong*. Entre estes extremos, o parâmetro de espessura pode ser interpolado (novamente de acordo com o termo de reflexão difusa). A Figura 5 apresenta a implementação dos *Outlines*, sendo a instrução *mix* utilizada para efetuar a interpolação linear entre ("*_UnlitOutlin*") e ("*_LitOutlineT*") com o parâmetro *shininess*. O valor resultante, ou seja, o valor interpolado é posteriormente utilizado como um limiar para determinar se um ponto está suficientemente próximo da silhueta. Se estiver, a cor do fragmento é definida para a cor do contorno, "*outlineColor*". É de salientar que as componentes "*_LitOutlineT*"e "*_UnlitOutlin*"também podem ser alteradas em tempo real.

```
//Outlines
if (dot(e, n)  < mix(_UnlitOutlineThickness, _LitOutlineThickness, intensity)) {
    vec3 fragmentColor = vec3(diffuse) * vec3(outlineColor);
    colorOut = vec4(fragmentColor, 1.0);
```

Figura 5: Parte do código referente a *Outlines*.

4 Testes e Resultados Obtidos

Neste capítulo são apresentados todos os testes efetuados e os respetivos resultados obtidos.

Inicialmente, aplicamos *Toon Shading Per Vertex* ao modelo 3D (sem textura). O resultado do mesmo foi apresentado no capítulo anterior e como anteriormente mencionado, este não foi o expectável. Desta forma, optamos por seguir outra abordagem, ou seja, aplicar *Toon Shading Per Pixel* ao modelo 3D (sem textura). Devido à simplicidade do projeto, decidimos aumentar a sua complexidade com a introdução de Texturas e *Outlines*. Ainda com o mesmo objetivo, implementamos componentes que podem ser alteradas em tempo real, nomeadamente "*shininess*", "*outlineColor*", "*_LitOutlineT*"e "*_UnlitOutlin*", como é possível observar na interface apresentada na Figura 6. É de notar que estas componentes são apenas relativas ao *Operator* "*shine*", que é referente aos dois *shaders shine.frag* e *shine.vert*. O resultado obtido com a abordagem *Toon Shading Per Pixel* + Texturas + *Outlines* é apresentado na Figura 7 (*Operator* "*shine*").

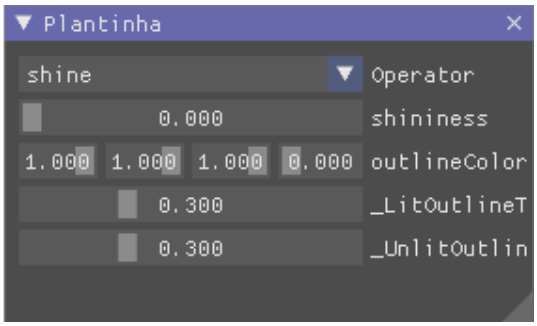


Figura 6: Interface do *Operator* "*shine*"

Figura 7: Resultado *Toon Shading Per Pixel* + Texturas + *Outlines* (*Operator* "*shine*").

Dentro da Interface "Plantinha", podemos alternar por 4 *Operators* diferentes (Figura 8, nomeadamente:

- "*textures*": Este utiliza os *shaders* texture.frag e *texture.vert*, tendo como objetivo servir de comparação com o resultado obtido no *Operator* "shine"e mostrar o modelo 3D texturizado, ou seja, com texturas integradas.
- "*toon*": Este utiliza os *shaders* toonV.frag e *toonV.vert*, sendo relativo ao *Toon Shading Per Vertex* efetuado inicialmente. Tem como objetivo servir de comparação com o resultado obtido no *Operator* "shine". É de notar que o modelo 3D aqui utilizado não tem texturas integradas.
- "*lighting*": Este utiliza os *shaders* toonP.frag e *toonP.vert*, sendo relativo ao *Toon Shading Per Pixel* efetuado inicialmente, ou seja, com um modelo 3D sem texturas integradas. Tem como objetivo servir de comparação com o resultado obtido no *Operator* "shine".
- "*shine*": Este utiliza os *shaders* shine.frag e *shine.vert*, sendo relativo ao *Toon Shading Per Pixel* + Texturas + *Outlines*. Este apresenta o resultado final do nosso projeto e contém componentes que podem ser alteradas em tempo real, como a "*shininess*", "*outlineColor*", "*_LitOutlineT*"e "*_UnlitOutlin*".

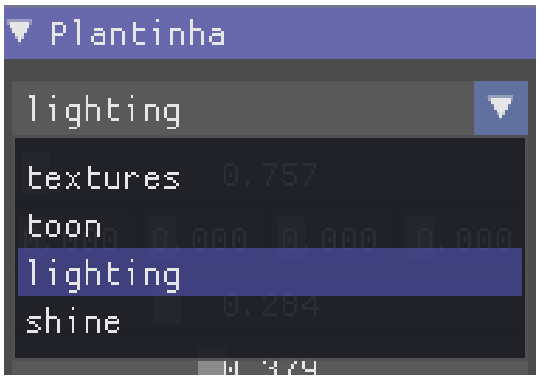


Figura 8: *Operators* da Interface "Plantinha": *shine*, *textures*, *toon* e *lighting*.

5 Conclusão

Ao longo da realização deste trabalho prático conseguimos aplicar os conhecimentos e técnicas lecionadas nas aulas de forma a obter uma implementação de um algoritmo que implementa a técnica de *Non Photorealistic Rendering* conhecida como *Toon Shading* de forma bastante satisfatória tendo isto em conta podíamos ter obter melhores resultados caso utilizássemos um modelo 3D com uma textura mais simples.