Universidade da Beira Interior
Departamento de Informática

Computação Interativa e Visualização
Estudo de *Edge Detection*

Daniel Pereira Reis
nº 10956

Docente:

Professor Doutor
Abel João Padrão Gomes

15 de Dezembro de 2020

Conteúdo

Lista de Figuras

Capítulo

1

Introdução

1.1 Enquadramento

Este relatório foi elaborado no contexto da disciplina de Computação Interactiva e Visualização do segundo ciclo de Engenharia Informática, na Universidade da Beira Interior (UBI), para tal, foram propostos vários temas sobre os quais trabalhar para uma apresentação e relatório do tema. O tema escolhido foi Edge Detection. Este trabalho, como referido acima, é um relatório sobre o tema escolhido, será a base de uma apresentação sobre o tema em aula.

Assim este relatório é o que servirá por base da dita apresentação e será a compilação de informação relevante academicamente de vários recursos.

1.2 Motivação

No contexto da disciplina de Computação Interativa e Visualização era necessário fazer uma pesquisa sobre temas propostos na aula e tivemos a opção de escolher um destes. Perante os meus interesses decidi que Edge Detection era o mais indicado, pois gosto de ver matemática a ter um feedback impactante não tendo uma grande complexidade de cálculo.

Dado que este filtro é simples de compreender e dá um resultado relativamente simples em relação achei que fosse o melhor para eu explorar.

Capítulo

2

Edge Detection

2.1 Introdução

Deteção de limites, ou *Edge Detection* em inglês, é uma técnica de de identificação de zonas em que há uma mudança grande de brilho na imagem. Esta técnica fornece os limites entre objetos na nossa cena e mudanças na topologia das superfícies. Esta técnica é aplicável a vários níveis como o campo de visão computacional, processamento de imagem e reconhecimento de padrões em imagens.

Esta técnica também cria efeitos que podem ser considerados esteticamente interessantes. O resultado de uma filtragem de uma cena em 3D passa a parecer uma imagem em 2D que foi desenhada a lápis num plano. Também pode ser aplicada a cenas 2D para aplicar o mesmo efeito de desenho sem cores.

2.2 Explicação Teórica da Técnica

O filtro de deteção de limites que é aplicado para demonstração envolve uma convolução por um filtro, também conhecido como um filtro de *kernel*. Este filtro é uma matriz de valores que define como transformar os pixeis da cena formada pela primeira fase de renderização, substituindo os pixeis resultantes da primeira passagem com a soma dos pixeis finais com os que rodeiam estes por meio de pesos para cada um dos pixeis que os rodeiam. Estes pesos para cada pixel que rodeia o pixel que estamos a considerar no momento podem ser definidos definidos de várias maneiras.

Para exemplificar a técnica que está em estudo neste relatório vão ser representados um bule de chá, uma toro e dois cubos como se pode observar na

figura **??**. Numa primeira fase será executado o código para criar a cena com um renderização normal e depois deste processo será feita uma execução do filtro para criar o efeito desejado.

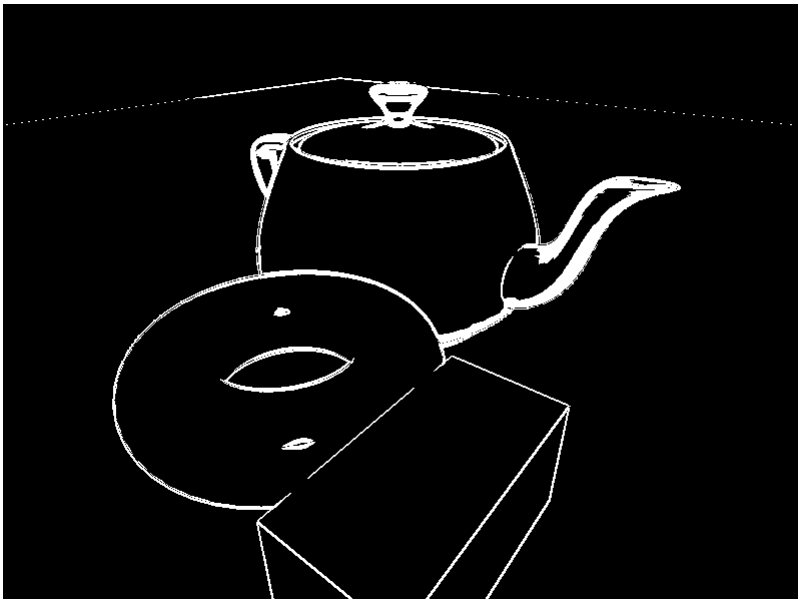

Figura 2.1: Cena no final da representação e o filtro aplicados.

2.3 Explicação teórica da metodologia

Para explicação, tenhamos em conta a figura 2.2 que é um exemplo dado da bibliografia da disciplina, em que este está inserido este relatório, os valores dos pixéis serão multiplicados pela matriz que é o filtro que queremos. Os valores correspondentes ao resultado da aplicação do filtro são a intensidade em tons de cinzento das componentes RGB dos pixéis originais.

	1	0	1		25	26	27
Filter:	0	2	0	Pixels:	28	29	30
	1	0	1		31	32	33

Figura 2.2: Exemplo para demonstração da operação.

Esta aplicação será feita através da multiplicação da matriz do filtro com a matriz de dimensões iguais ao nosso filtro.

Como mencionado previamente isto será aplicado numa segunda fase, pois para se poder manter o frame buffer com a imagem original e o novo framebuffer que acumula estes valores do resultado do filtro para a nossa imagem final que vai ser representada no ecrã.

2.4 Métricas de Avaliação

Pelo título deste relatório, a técnica que está aqui em estudo é o operador de Sobel uma das técnicas mais simples de deteção de limites de uma imagem. Este foi desenhado para aproximar o gradiente da intensidade de cada pixel na imagem. Isto é feito através de duas matrizes 3x3 como representadas no exemplo. O resultado da aplicação do filtro é uma componente vertical e uma componente horizontal, como representado na figura 2.4, de cada uma das multiplicações das matrizes pelos valores dos pixeis por cada um dos filtros.

$$\mathbf{S}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \mathbf{S}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Figura 2.3: Matrizes horizontal e vertical respetivamente.

Com isto as duas componentes poderão ser utilizadas para cálculo da magnitude do gradiente a partir de um certo valor limite. Este gradiente é aproximado pela equação da figura 2.4:

$$g = \sqrt{s_x^2 + s_y^2}$$

Figura 2.4: Equação do cálculo do gradiente.

Este valor de gradiente será comparado com o valor de limite. Se o gradiente for maior que o valor imposto o pixel é categorizado como pixel de limite e será representado com a cor escolhida, branco para o caso em estudo.

2.5 Funcionamento

Para criar a nossa cena base será feita uma renderização da cena com todos os objetos e geometria será aplicado o modelo de Phong numa primeira sub-rotina, para a primeira passagem anteriormente referida, que nos vai criar a imagem que vais ser representada para aplicarmos o nosso filtro de deteção de limites.

Para a segunda passagem será feita uma sub-rotina que fará a nossa filtragem da imagem para criar o efeito desejado. Com este processamento em sub-rotinas o *fragment shader* pode ser chamado apenas uma vez por cada pixel.

Na segunda passagem o valor dos oito pixeis adjacentes, da imagem criada pela primeira fase em Phong, ao que está a ser processado no momento e é calculada a luminosidade destes pixeis e do pixel que estamos a processar através da função luma no código que estamos a apresentar.

Os filtros horizontais e verticais do operador de Sobel são então aplicadas e os resultados são guardados na componente sx e sy do código do *fragment shader*.

No final de calcular estas componentes é possível calcular a magnitude do gradiente pelo quadrado da fórmula anteriormente apresentada. Com isto será possível evitar a raiz quadrada do que foi apresentada na fórmula do gradiente. Com este valor a comparação será feita de a forma que sempre que o valor da magnitude for maior que o valor de limite será desenhado um ponto branco senão não será representado nada no ecrã. Isto revela que foi detetado um limite com o cálculo deste gradiente e que estamos num ponto de transição entre dois limites de objetos na cena.

2.6 Conclusão

Em suma a técnica do operador de Sobel, que estamos aqui a demonstrar, tem um bom desempenho e é fácil de implementar, mas para certos tipos de aplicação em que, por exemplo, fazemos o valor limite ser muito pequeno são criados artefactos que revelam o quão cru é esta técnica.

Esta técnica tem uma alta sensibilidade a variações de altas frequências da cor. É possível reduzir esta sensibilidade às variações com um "*blur pass*" entre o processamento da imagem e o processamento do operador de Sobel. Este comportamento seria verificado devido ao tornar mais suave da imagem base que vai ser processada pelo operador.

Para melhorar o processamento será possível reduzir o numero de acessos à textura inicial calculada com Phong, dado que este processamento é relativamente lento em relação a por exemplo o calculo inicial da imagem.

Capítulo

3

Desenvolvimento

3.1 Introdução

Tendo em conta aquilo que foi mencionado na parte teórica deste relatório passamos, com base nisso à análise da implementação da solução de *Edge Detection*. Esta implementação que é proposta neste relatório tem por base o código do livro em que se baseia a unidade curricular em que este estudo está inserido. Este livro está nas referências do estudo.

Para esta secção houveram algumas adições de objectos para comprovar certas caraterísticas da técnica em estudo, contudo a implementação não difere na parte dos *shaders* que estes já têm a técnica perfeitamente implementada para o estudo que é proposto.

3.2 Implementação

3.2.1 Modificações do Programa Base

Para começar a cena teve de ser modificada para analisar melhor a técnica para poderem ser retiradas conclusões próprias por manipulação do projecto base. Assim depois de alguma manipulação a conclusão a que se chegou foi a adição de dois cubos que se pode verificar na figura 3.1, estes dois cubos demonstrarão que a cor será o que será analisado pelo método.

O passo seguinte definir a nossa nova cena, após a adição dos cubos e dar-lhes uma dimensão. assim os nossos objectos estão definidos na cena ainda sem uma localização e materiais. Este passo apesar de básico não deverá ser esquecido. Esta definição está representada na figura 3.2, como se pode ver estão definidos com arestas de 1 unidade.

```
#ifndef SCENEEDGE_H
#define SCENEEDGE_H

#include "ingredients/scene.h"
#include "ingredients/glslprogram.h"
#include "ingredients/plane.h"
#include "ingredients/torus.h"
#include "ingredients/teapot.h"
#include "ingredients/cube.h"

#include "ingredients/cookbookogl.h"

#include <glm/glm.hpp>

class SceneEdge : public Scene
{
private:
    GLSLProgram prog;

    GLuint fsQuad, pass1Index, pass2Index, fboHandle, renderTex;

    Plane plane;
    Torus torus;
    Teapot teapot;
    Cube cube1;
    Cube cube2;

    float angle, tPrev, rotSpeed;

    void setMatrices();
    void compileAndLinkShader();
    void setupFBO();
    void pass1();
    void pass2();

public:
    SceneEdge();

    void initScene();
    void update(float t);
    void render();
    void resize(int, int);
};

#endif // SCENEEDGE_H
```

Figura 3.1: Definição da cena.

Na figura 3.3 estão agora a ser definidos os materiais que compõe cada cubo. Neste caso estão definidos com exatamente o mesmo material para de certa forma "enganar"o método a não detectar o limite entre os dois cu-

```
SceneEdge::SceneEdge() : angle(0.0f), tPrev(0.0f), rotSpeed(glm::pi<float>() / 8.0f),
                         plane(50.0f, 50.0f, 1, 1),
                         teapot(14, mat4(1.0f)),
                         torus(0.7f * 1.5f, 0.3f * 1.5f, 50, 50),
                         cube1(1.0f),
                         cube2(1.0f)
{}
```

Figura 3.2: Adição das dimensões do cubo.

bos. Isto para efeitos de demonstração será modificado para que os colegas da unidade curricular percebam melhor o funcionamento do método.

```
void SceneEdge::pass1()
{
MyRegion

    //Adição dos cubos abaixo

    prog.setUniform("Light.Position", vec4(0.0f, 0.0f, 0.0f, 1.0f));
    prog.setUniform("Material.Kd", 0.95f, 0.45f, 0.32f);
    prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
    prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
    prog.setUniform("Material.Shininess", 100.0f);
    model = mat4(1.0f);
    model = glm::translate(model, vec3(3.0f, 1.0f, 3.0f));
    model = glm::rotate(model, glm::radians(90.0f), vec3(1.0f, 0.0f, 0.0f));
    setMatrices();
    cube1.render();

    prog.setUniform("Light.Position", vec4(0.0f, 0.0f, 0.0f, 1.0f));
    prog.setUniform("Material.Kd", 0.95f, 0.45f, 0.32f);
    prog.setUniform("Material.Ks", 0.95f, 0.95f, 0.95f);
    prog.setUniform("Material.Ka", 0.1f, 0.1f, 0.1f);
    prog.setUniform("Material.Shininess", 100.0f);
    model = mat4(1.0f);
    model = glm::translate(model, vec3(3.0f, 1.0f, 4.0f));
    model = glm::rotate(model, glm::radians(90.0f), vec3(1.0f, 0.0f, 0.0f));
    setMatrices();
    cube2.render();
}
```

Figura 3.3: Definição de materiais e localização dos cubos na cena.

3.2.2 *Shaders*

No que toca a *shaders* não existiram modificações dado que o comportamento já é o requerido, de qualquer forma há uma necessidade de explicar o que está a acontecer por trás do resultado da cena.

Na figura 3.4 temos a representação da primeira sub rotina que irá aplicar o modelo de Phong que será a imagem base que será executada por parte da nossa cena. Com isto se quer dizer que a cria a cena inicia para depois ser processada pelo método de *edge detection*. Este tipo de implementação permite-nos assim utilizar um único *fragment shader* para a execução do nosso método.

```
vec3 phongModel( vec3 pos, vec3 norm )
{
    vec3 s = normalize(vec3(Light.Position) - pos);
    vec3 v = normalize(-pos.xyz);
    vec3 r = reflect( -s, norm );
    vec3 ambient = Light.Intensity * Material.Ka;
    float sDotN = max( dot(s,norm), 0.0 );
    vec3 diffuse = Light.Intensity * Material.Kd * sDotN;
    vec3 spec = vec3(0.0);

    if( sDotN > 0.0 )
        spec = Light.Intensity * Material.Ks *
                pow( max( dot(r,v), 0.0 ), Material.Shininess );

    return ambient + diffuse + spec;
}
```

Figura 3.4: Aplicação de modelo de Phong na cena.

Nesta ultima figura da secção temos a aplicação directa da teoria do que definimos como o que seria o nosso método, o retirar da nossa cor para tons de cinzento, aplicar o calculo matricial para a componente horizontal e vertical e finalmente calcular a magnitude do gradiente como falado na secção teórica.

Se este valor for superior ao valor limite criamos o nosso valor de um pixel branco, caso contrario um pixel a preto.

```
subroutine( RenderPassType )
vec4 pass2()
{
    ivec2 pix = ivec2(gl_FragCoord.xy);

    float s00 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(-1,1)).rgb);
    float s10 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(-1,0)).rgb);
    float s20 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(-1,-1)).rgb);
    float s01 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(0,1)).rgb);
    float s21 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(0,-1)).rgb);
    float s02 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(1,1)).rgb);
    float s12 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(1,0)).rgb);
    float s22 = luminance(texelFetchOffset(RenderTex, pix, 0, ivec2(1,-1)).rgb);

    float sx = s00 + 2 * s10 + s20 - (s02 + 2 * s12 + s22);
    float sy = s00 + 2 * s01 + s02 - (s20 + 2 * s21 + s22);

    float g = sx * sx + sy * sy;

    if( g > EdgeThreshold )
        return vec4(1.0);
    else
        return vec4(0.0,0.0,0.0,1.0);
}
```

Figura 3.5: Aplicação de *edge detection*.

3.3 Conclusão

Assim concluindo a análise tanto do código já criado e as modificações feitas, espera-se que se tenha proporcionado um melhor entendimento sobre a técnica por aplicação desta a nível de código, como mencionado o resto do código está no livro de base da unidade curricular em que este estudo está inserido.

Aplicação das modificações propostas deverá criar os resultados criados neste estudo.

Bibliografia

[1] David Wolff. *OpenGL 4.0 Shading Language cookbook*. Clarendon Press, 2011.