Computação Gráfica
Relatorio Final

a61007 - César Perdigão
a61009 - Luis Caseiro
a61030 - Diogo Alves
a61075 - Ricardo Branco
a61078 - Pedro Maia

Junho 2013

Conteúdo

Capítulo 1

Resumo

Com base no que foi adquirido nas aulas da unidade curricular o grupo começou por pesquisar um pouco sobre as funcionalidades do *OpenGL* e também o que o *GLUT* podia oferecer de forma a aprender alguns "truques"para poupar trabalho e tempo.

Após esta fase de pesquisa começou-se por definir primitivas com as quais, foi possivel construir objetos, por exemplo, um sofa pode ser feito a partir de cubos e cilindros. As primitivas construidas foram o cubo, cilindro, cone, esfera e plano.

Depois de construir estas o grupo construiu objetos a partir destas, os objetos construidos foram candeeiros, mesas, cadeiras e copos.

Na terceira fase com o alargamento do conhecimento acerca de como optimizar os objetos, foi pedido para alterar a estrutura das primitivas desenvolvendo-se então *Vertex Buffer Objects* mais conhecidos como **VBOs**, ou seja, todas primitivas foram alteradas e transformadas em *VBOs*, nesta fase colocamos também iluminação(array normais) e coordenadas de textura.

Na quarta fase começou-se por transformar em classes os objetos que iriam formar o bar, utilizando os *VBOs* das primitivas. Depois de corrigir alguns erros dentro destas classes, começou-se a colocar texturas nos objetos e a colocar estes dentro dos limites do bar ficando assim o bar composto com todos os objetos que se achou necessário.

O grupo decidiu implementar como um extra no bar, sombras e alguns objetos, o que o tornou mais real.

Capítulo 2

Objetivos

O objetivo final do projeto é construir um bar, para isso o grupo tem que desenvolver todos os objetos necessários, por exemplo cadeiras e mesas, e tem de definir a iluminação do bar, onde os objetos serão colocados, texturas a aplicar nos objetos e definir tamanhos para estes. Para isso o grupo baseou-se no que foi lecionado na unidade curricular de *Computação Gráfica*.

Capítulo 3

Desenvolvimento

3.1 Primeira Fase

3.1.1 Preliminares

Nas aulas da unidade curricular de *CG* trabalhou-se com uma função da biblioteca **GLUT** *glvertex3f*, esta função recebe como argumento três *floats* e desenha um ponto que tem como coordenadas esses três valores. Utilizou-se para desenhar triangulos as funções *glBegin(...) e glEnd()* que associa três pontos a um triangulo pela regra da mão direita.

3.1.2 Implementação

Na implementação foram utilizadas as funções descritas em cima como foi aprendido nas aulas da uc.

3.1.2.1 Implementação Cubo

O cubo foi implementado de forma muito similar ao plano, pois um cubo são seis planos, dois por cada eixo. Foram implementados três blocos de codigo, em que cada um destes desenhava duas faces, por exemplo a face da esquerda e a face da direita apenas tinham duas diferenças, uma era a **coordenada x** sendo a coordenada x da face esquerda negativa e a da face direita positiva, a outra diferença era a **ordem com que se desenhavam os pontos** pois a orientação das duas faces é diferente.

Cada face do cubo foi desenha através de inumeros quadrados, em vez de ser só um, o numero de quadrados que cada face tem é definido através de um argumento da função, ou seja, a função *cubo(2,5)* irá desenhar um cubo de lado 2 e cada face será composta por 25 quadrados.

3.1.2.2 Erros

Alguns erros que se realizaram nesta fase foram:

- Primitivas implementadas sem serem divididas em camadas, por exemplo, o cubo era constituido por 6 quadrados.
- Primitivas com mais pontos do que necessário, ou seja, pontos que não seriam visiveis estavam implementados, por exemplo, no cilindro estavam implementados pontos desnecessários .
- Erros nas variaveis de ciclo fazendo com que fossem desenhados mais pontos que o necessário, por exemplo, a variavel que controlava o ciclo era um *float*.

3.1.3 Resultados

A imagem abaixo é o cubo resultante da implementação descrita acima

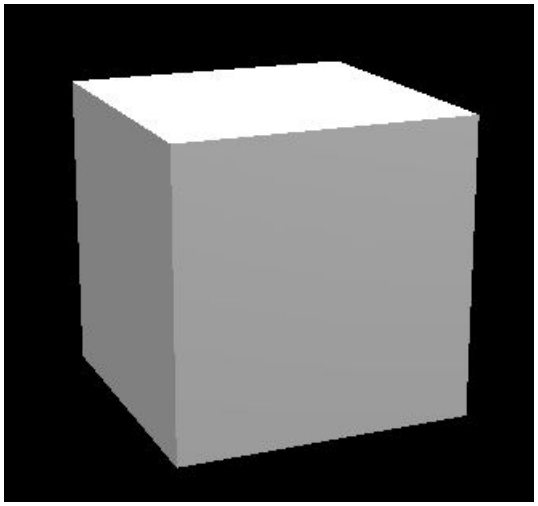

Figura 3.1: Cubo

3.2 Segunda Fase

3.2.1 Preliminares

Para esta segunda fase foi necessário saber como rodar, transportar e escalar as primitivas que foram feitas anteriormente, para isso foram utilizadas três funções essenciais, sendo elas *glTranslatef*, *glRotatef* e *glScalef* foi graças a estas que foi possivel construir objetos para colocar no bar, por exemplo, o sofá que o grupo construiu foi feito a partir de dois cubos e dois cilindros, sendo os cubos transformados em paralelipipedo através da função *glScalef*, os cilindros sofreram uma rotação devido à função *glRotatef* e estes foram colocados em lugares diferentes através da função *glTranslatef*.

Utilizou-se nesta segunda fase uma matriz e duas funções *glPushMatrix* e *glPopMatrix* a matriz e estas funções permitem guardar pontos em que se está atualmente e voltar a estes mais tarde, isto fez com que se poupasse algum trabalho e alguma complexidade de codigo, apesar de algumas vezes a sua utilização não ter sido a mais correta.

3.2.2 Implementação

3.2.2.1 Implementação do sofá

O sofa implementado pelo grupo é constituido por dois paralelipipedos que foram feitos a partir de um "scale"à primitiva do cubo e de dois cilindros que foram "rodados"e "transportados"de forma a transformar o objeto no desejado, ou seja, para fazer o sofá foram utilizadas as primitivas definidas na fase anterior e as funções *glTranslatef*, *glRotatef* e *glScalef*. Utilizamos também as funções *glPushMatrix* e *glPopMatrix* que ajudaram a controlar o uso das funções referidas anteriormente.

3.2.2.2 Erros

Alguns erros realizados nesta fase foram:

- Má utilização das funções *glPushMatrix* e *glPopMatrix*, pois havia mais sempre "pops"do que "pushs"ou ao contrario.
- Dificuldades com a função *glRotatef*, dificuldade em saber que eixo deviamos utilizar.

3.2.3 Resultados

Abaixo encontra-se o sofa resultante:

3.3 Terceira Fase

3.3.1 Preliminares

Nesta fase o grupo teve que transformar todas as primitivas criadas para **VBOs** o que torna as construções mais eficientes. Para o utilizar os *VBOs* primeiro é preciso utilizar a função *glEnableClientState()* para ativar os *buffers*. Existem duas formas de implementar *VBOs*, numa utiliza-se um array de indices na outra alternativa não se utiliza o array de indices. A diferença principal é que com o auxilio do array de indices pode-se reutilizar pontos já definidos, o que não acontece no caso contrário. No fim para gerar os *VBOs* precisamos das funções *glGenBuffers*, *glBindBuffer* e *glBufferData* estas funções geram e desenham os pontos definidos anteriormente. Nesta fase implementou-se também

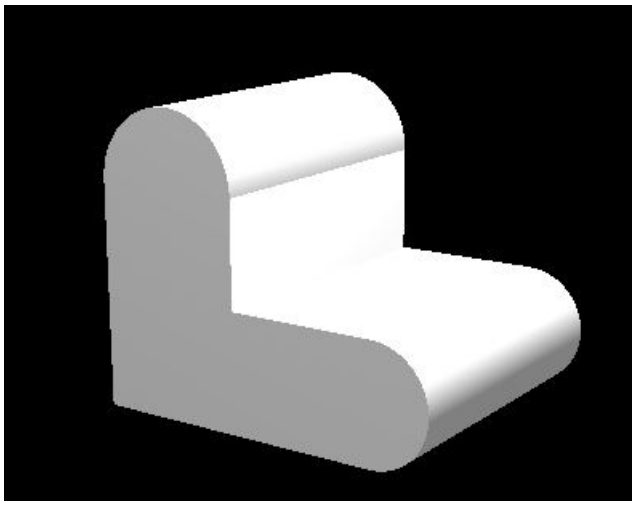

Figura 3.2: Sofa

as coordenadas de textura e a iluminação, sendo representadas também por arrays em que os arrays utilizam as mesmas funções referidas acima modificando apenas os argumentos que estas recebem.

Nesta fase o grupo começou a implementar classes em **C++** sendo cada classe uma primitiva diferente, por exemplo, a classe **Cilindro** é onde se implementa o construtor e se desenha o cilindro.

3.3.2 Implementação

3.3.2.1 Implementação do VBO Cilindro

A implementação do VBO do cilindro pode ser dividida em 2 partes, parte um desenhar a lateral e a parte dois desenhar as bases. Começando pela lateral, a primeira coisa que se fez foi estimar quantos pontos seriam desenhados e alocar espaço para os arrays, *Vertex*, normais, texturas e indices tendo este ultimo um tamanho diferente. Depois preencheram-se os arrays (vertex, normais e texturas) enviando os pontos para a placa gráfica de acordo com o objetivo, ou seja, fazer a lateral de um cilindro. Logo depois preencheu-se o array de indices com a ordem com que os pontos iriam ser desenhados e que pontos iriam ser reaproveitados, para isso pensou-se que os pontos estavam desenhados num array bidimensional em que utilizando as variaveis dos ciclos foi possivel determinar os pontos e assimdeterminar a ordem.

No caso das bases o procedimento foi muito semelhante sendo as diferenças mais significativas, o calculo de normais e coordenadas de textura e o preenchimento do array de indices em que devido ao metodo usado para implementar a primitiva foi necessário somar um offset de forma a não destruir pontos já desenhados.

3.3.2.2 Erros

Aguns erros realizados nesta fase:

- Erros na implementação do array de indices, por vezes ordem errada outras vezes as variaveis usadas estevam incorretas.
- Erros na alocação de espaço para os arrays.

3.3.3 Resultados

Encontra-se abaixo resultado da implementação acima.

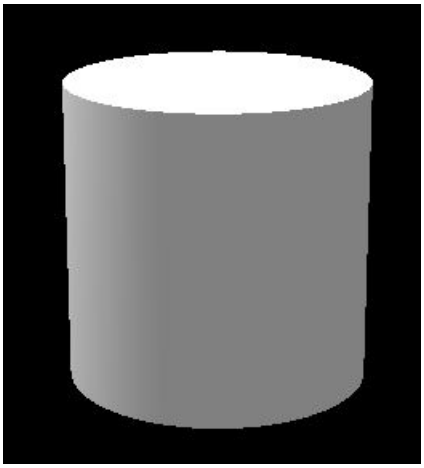

Figura 3.3: Cilindro

3.4 Quarta Fase

3.4.1 Preliminares

Para esta fase os fundamentos teoricos necessarios foram, como na segunda fase foi preciso trabalhar com as funções *glTranslatef*, *glRotatef*, *glScalef*, *glPush-Matrix* e *glPopMatrix*.

Para aplicar texturas a objetos o grupo teve que compreender o que foi lecionado nas aulas da uc e também pesquisar na biblioteca *DevIL* para tratar imagens, as funções que utilizamos foram *glEnable()* que serve para ativar as texturas e as funções *glBindTexture()* e *glBindTexture()* sendo estas responsaveis por desenhar a textura na primitiva.

As texturas foram recolhidas de imagens por isso foram utilizadas tambem as funções *ilGenImages()*, *ilBindImage()* e *ilLoadImage()* para abrir a imagem, para converter em **RGBA** foi utilizada a função *ilConvertImage()*.

As classes de objetos foram criadas de forma a cada objeto ter uma classe, ou seja, na classe **Sofa** apenas seria desenhado um sofa, as classes foram implementadas na linguagem **C++** logo o grupo implementou da forma com que melhor se ajusta com a linguagem.

3.4.2 Implementação

3.4.2.1 Implementação da classe Balcao

A classe Balcao como o seu nome indica vai ser responsavel por desenhar o balcao do bar, este balcão é constituido apenas por algumas variaveis de instancia, sendo elas comprimento, altura e largura do balcão e também a espComp que é a espessura de comprimento, ou seja, a espessura do balcão no eixo do x e a espLarg que é a espessura de largura, ou seja, a espessura do balcão no eixo do z. Como variavel de instancia tem-se também um cubo.

No construtor da classe os argumentos recebidos são comp, alt e larg, sendo respetivamente comprimento, altura e largura. As outras variaveis são calculadas a partir destas, sendo o cubo construido através do construtor da classe Cubo e tem como lado 1.

Em seguida foi implementado a função desenhar que recebe como argumento as texturas a implementar no objeto que irá ser construido. Nesta função são utilizadas inumeras funções da biblioteca *GLUT*, sendo estas funções já referenciadas acima, estas funções transformaram o cubo em paralelipipedo e colocaram-no no sitio suposto de forma a que este forma-se o balcão desejado, por fim colocaram-se as texturas desejadas nos locais desejados acabando assim a função desenhar ficando o balcão pronto para ser colocado no bar.

3.4.2.2 Colocar objetos no Bar

Para colocar os objetos no **Bar** definiu-se primeiro os limites do bar relativamente aos três eixos e depois através das funções *glTranslatef*, *glRotatef*, *glPushMatrix*, *glPopMatrix* e da função *desenhar()* que todos os objetos têm na sua respetiva classe foi possivel construir o bar de forma a este ficar como era pretendido.

3.4.2.3 Erros

- Classes de objetos mal implementadas pois na função de desenhar também era criado o VBO logo sempre que a função era chamada tinha que criar um novo VBO em vez de aproveitar os pontos definidos anteriormente, o que deixava os objetos lentos e ia contra o objetivo dos VBO.Exemplo:

```
Antes:
    glPushMatrix();
    glScalef(espComp,altura,largura);
    cubo=new Cubo(1,DIV);
    cubo->desenhar();
```

```
    glPopMatrix();


Depois:
    No construtor:
        cubo=new Cubo(1,DIV);
    Na função desenhar:
        glPushMatrix();
        glScalef(espComp,altura,largura);
        cubo->desenhar();
    glPopMatrix();
```

- Correção de alguns objetos em que o numero de polignos desenhados era enorme o que também diminuia o desempenho.

3.4.3 Resultados

Os resultados obtidos da implementação acima e de todo o bar foram:

Figura 3.4: Balcao

Figura 3.5: Bilhar

3.5 Extras

3.5.1 Preliminares

Para enriquecer o trabalho, decidiu-se tentar implementar uma tecnica de sombras baseado em *shadow mapping*. Esta tecnica consiste em desenhar a cena do ponto de vista da luz, guardando apenas a informação das profundidades. De seguida desenha-se a cena do ponto de vista da camara mas faz-se um teste de distancia à luz: caso esta distancia seja menor, o objecto é desenhado com luz total, caso contrario o objecto é desenhado apenas com luz ambiente.

3.5.2 Implementação

Para implementar esta tecnica tiveram de se efetuar alguns passos a mais dos acima referidos, isto porque nao foram usados *shaders*, que possivelmente poupariam algum trabalho.

Na inicialização teve de se acrescentar um *Frame Buffer Object* para o desenho da cena do ponto de vista da luz. Este objecto fica ligado a uma textura que armazena a informação de profundidades. Esta textura tem um parametro interno que activa a comparação de profundidades. Do resultado desta comparação resulta um valor de transparencia de 0 ou 1, que faz com que os objectos em sombra sejam transparentes. No *Frame Buffer* foi ainda alterado um parametro para nao escrever os *buffers* de cor, pois não são necessarios.

Na rotina de processar a *frame* são efectuados os seguntes passos:

- Calculam-se as matrizes *projection* e *modelview* da camara e da luz;
- Activa-se o *Frame Buffer* auxiliar e desenha-se a cena do ponto de vista da luz, nesta passagem são ocultadas as faces da frente para evitar o efeito de *shadow acne*, que acontece devido à impercisão do *zbuffer* que causa

que pontos de uma mesma superficie iluminada ora estejam em luz, ora estejam em sombra;

- Desenha-se novamente a cena do ponto de vista da camara apenas com luz ambiente (e uma quantidade baixa de luz difusa para eliminar o efeito plano nos objectos em sombra, que apesar de irrealista, tem um melhor aspecto);
- Calcula-se a matriz de transição entre as coordenadas do mundo para o *Clip Space* da luz;
- Ativa-se a textura que contem o *shadowmap*, com geração automatica de coordenadas tendo como referencia a matriz calculada anteriormente, coordenadas estas geradas no modo *eye linear* (os objectos movem-se na textura);
- É de novo desenhada a cena agora com luz total e com o *blend* ligado, para misturar a nova imagem com luz, com a anteriormente gerada, e como os objectos em sombra estao com o valor de *alpha* a 0, não são redesenhados.

3.5.2.1 Erros

Um dos erros iniciais da implementação era o facto de se usar um *Frame Buffer*, o que fazia com que o tamanho máximo do *shadowmap* fosse o tamanho da janela, isto porque o *Frame Buffer* padrão é redimensionado para o tamanho da janela, logo não existe mais espaço para desenhar o mapa de sombras, o que causava que houvesse areas sem sombras e outras com sombras nao adequadas ao local.

Outro erro que tambem ocorria era o aparecimento de sombras atras do ponto de luz, mas este problema ficou resolvido quando se passou a luz a um foco.

3.5.3 Resultados

Figura 3.6: Detalhes de sombra

Capítulo 4

Conclusão

O trabalho permitiu consolidar os conhecimentos adquiridos nas aulas, relativo a todas as temáticas desenvolvidas no trabalho que foram aprendidas nas aulas práticas, como por exemplo a criação de VBOs e aplicação de texturas a objetos mas também algumas tematicas que o grupo pesquisou de forma a completar o trabalho.

Durante o execução deste apresentaram-se algumas dificuldades sendo algumas em termos cognitivos e tecnologicos.

A nivel cogintivo a implementação de VBOs foi algo complicado devido a alguns erros que surgiam em que não se conseguia detetar a origem. Em termos tecnologicos, o hardware de computadores de alguns elementos do grupo não conseguia correr o programa como era suposto e isto fez com que o grupo se atrasasse um pouco em algumas fases.

No futuro podia-se implementar no bar mais luzes de forma a todo o bar ficar iluminado, ajustar melhor as sombras e podia-se também melhorar alguns objetos.

O grupo mostra-se satisfeito com o trabalho desenvolvido pois acha que este foi gratificante em que todo o desenvolvido desde o inicio do semestre tinha um objetivo e ver que no fim o objetivo foi cumprido deixou o grupo feliz.

Capítulo 5

Anexos

Em anexo vão imagens retiradas do trabalho desenvolvido:

5.1 Primitivas

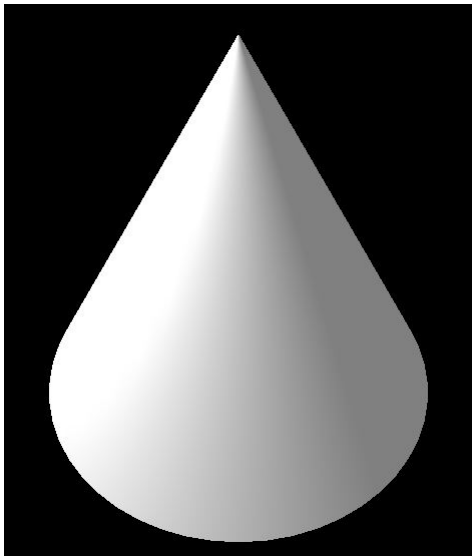

Figura 5.1: Cone

Figura 5.2: Plano

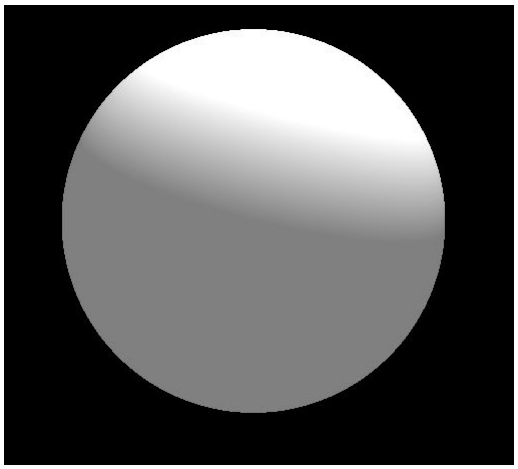

Figura 5.3: Esfera

Figura 5.4: Bar

Bibliografia

http://www.opengl.org/resources/libraries/glut/
http://www.paulsprojects.net/tutorials/smt/smt.html
http://www.opengl.org/wiki/Shadow_Mapping_without_shaders
http://ogldev.atspace.co.uk/index.html