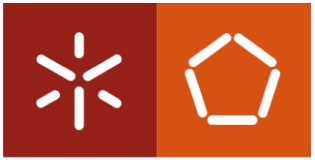


Escola de Engenharia

Departamento de Informática

Universidade do Minho

Licenciatura em Engenharia Informática

Projecto de Computação Gráfica

"Bar"

Bruno Ferreira, A61055
Serafim Pinto, A61056

Grupo 19

Braga, Maio de 2013

Conteúdo

Lista de Figuras

1 Resumo

Neste relatório apresentamos todos os passos e decisões tomadas na construção do trabalho prático de Computação Gráfica.

Este projecto consiste na representação de um Bar, enquanto contrução geométrica. O Bar poderá conter elementos como mesas, cadeiras, copos, entre outros.

Numa primeira parte deste relatório fazemos uma breve introdução ao projecto, seguindo-se a análise do seu desenvolvimento ao longo das várias fases. Depois apresentamos uma descrição detalhada das técnicas usadas e, por fim, temos a conclusão.

2 Introdução

No desenvolvimento do presente projecto de Computação Gráfica pretendemos, para além de aplicar os conhecimentos leccionados na Unidade Curricular, desenvolver sensibilidade para a criação de aplicações de elevada complexidade gráfica. Propomos-nos portanto a criar um espaço com elementos comummente observados num Bar.

Este trabalho está dividido em quatro fases distintas. Na primeira fase apenas são feitas as primitivas: plano, cubo, cilindro, esfera. Numa segunda fase, devem ser apresentados alguns itens que deverão estar no espaço final, tais como: mesas, cadeiras, copos e candeeiros. A terceira fase consiste na preparação das primitivas das fases anteriores para a inclusão de texturas e iluminação. Nessa fase, devemos também optimizar o projecto utilizando *Vertex Buffer Objects*. Na quarta e última fase deve ser montado o Bar propriamente dito, fazendo uso de tudo o que foi criado até ao momento. Além disto, a visualização do modelo deve ser feita recorrendo a uma câmara de movimento livre, com o uso do teclado e rato. Posto este problema, que está descrito no enunciado do trabalho, deveremos implementar e tornar possível a criação deste cenário seguindo os requisitos que serão apresentados no capítulo seguinte.

O projecto será desenvolvido em C++ no software Visual Studio 2012. Utilizaremos também a biblioteca gráfica do OpenGL, os utilitários GL (GLU) e o GL Extension Wrangler (GLEW).

3 Descrição do Trabalho e Análise de Resultados

Neste capítulo vamos descrever o desenvolvimento de cada um dos requisitos a ser apresentados. Descreveremos também os principais passos da sua implementação e estruturação e faremos uma breve análise dos resultados obtidos com estas mesmas implementações.

3.1 Requisitos

Durante esta secção apresentamos em detalhe os requisitos propostos em cada fase para o desenvolvimento do nosso projecto.

3.1.1 Primeira Fase

- Construir uma biblioteca de primitivas: plano, cubo, cilindro e esfera;
- Cada primitiva deve ser feita numa função que desenhe a primitiva centrada na origem;
- A função para cada primitiva deve ter um conjunto de parâmetros que permita desenhar de acordo com a dimensão e resolução pretendidas;
- Construção de uma aplicação OpenGL que permita a visualização de cada primitiva separadamente.

3.1.2 Segunda Fase

- Apresentar rotinas que desenhem os seguintes objectos: mesas, cadeiras, copos e candeeiros;
- Desenhar o espaço limite do bar, ou seja, paredes chão e tecto;
- À exceção do copo, todos os outros objectos devem ser desenhados à custa da biblioteca de primitivas da primeira fase, e de transformações geométricas. O copo deve ter uma rotina própria para a sua construção;

3.1.3 Terceira Fase

- Definição das coordenadas de textura e definição das normais nas primitivas;
- Utilização de VBOs;
- Construção de uma aplicação que permita a visualização de cada composição geométrica.

3.1.4 Quarta Fase

- Aplicação com a apresentação da composição geométrica final, ou seja, um bar com mesas, candeeiros, cadeiras e copos (construídos apenas com recurso às primitivas e transformações geométricas desenvolvidas nas fases anteriores);
- Utilização de texturas e iluminação.
- Visualização da cena utilizando uma câmara de movimento livre, sem necessidade de detecção de colisões.

3.2 Implementação

Nesta seccção descrevemos de forma incremental o desenvolvimento do projecto ao longo das 4 fases.

3.2.1 Primeira Fase

A primeira fase pedia que fosse construída uma biblioteca com primitivas geométricas, aqui explicamos como nesta fase do projecto construímos cada uma delas. Estas primitivas foram numa classe à qual chamámos **Primitivas**. Para facilitar a visualização destas, elaborámos também uma *main.cpp*, onde é possível através do teclado mover a câmara e visualizar o objecto de ângulos diferentes.

Criámos também o módulo **Utilities**, onde temos pequenas funções que nos facilitam na escrita do código tornado-o mais fácil de ler (p.e função que passa de graus para radianos). Além disto, temos um menu associado ao botão direito do rato onde o utilizador pode escolher qual a primitiva a visualizar, e visualizar o objecto de várias formas (figura colorida, apenas linhas ou apenas pontos).

Plano

A função que desenha o plano recebe como argumentos, o comprimento, a largura e ainda o número de camadas que o plano terá. Como queremos o nosso plano com um determinado número de camadas, dividimos as variáveis comprimento e largura pelas camadas pretendidas, e guardamos o resultado em *x2* e *z2* respectivamente (sobre camadas, ver secção 3.4.1).

Como também queremos desenhar centrado na origem, dividimos o comprimento e a largura por dois, e guardamos o resultado nas próprias variáveis. E é assim, com base nessas variáveis que desenhamos os triângulos que compõem o nosso plano. A Figura 1 permite visualizar o nosso plano nesta fase:

Figura 1: Plano com 5 de comprimento, 4 de largura e 3 camadas

Cubo

A função que desenha o cubo recebe como argumento o comprimento do lado e o número de camadas no eixo dos X e dos Z. Tal como no plano, para centrar o cubo na origem, a variável lado é dividida por dois. Para desenhar esta primitiva, como seria de esperar desenhamos seis faces, e cada uma delas desenhada da mesma maneira que é desenhado um plano.

Optámos por criar um cubo e não um paralelipipedo, pois um paralelipipedo pode ser obtido escalando um cubo.

Visto ser ineficiente, não recorremos à função que cria o plano para construir o cubo (ver secção 3.4.2)

Cilindro

Este primitiva foi desenhada de um método muito semelhante ao usado no execício da aula prática. A função recebe como argumentos o raio, a altura, o número de camadas verticais e o número de camadas horizontais. Existe uma varíavel *delta* que resulta da operação

```
float delta = 2 * M_PI / fatias;
```

Ficando guardada nesta variável o ângulo necessário para obter as coordenadas polares. Mais uma vez como a primitiva deve ser desenhada na origem, dividimos a altura por dois. O Cilindro é assim desenhado com recurso a coordenadas polares

(ver secção 3.4.3). Posteriormente o cálculo deste ângulo foi modificado para passar a ser calculado a cada iteração para evitar erros de vírgula flutuante (ver secção 3.4.4).

Esfera

A função que desenha esta primitiva recebe três argumentos: o raio da esfera, o número de camadas verticais e o número de camadas horizontais. É através de dois ângulos e funcões trignométricas que conseguimos desenhar a primitiva. Estas variáveis são:

```
float alpha = 2 * M_PI / fatias;
float beta = M_PI / seccoes;
```

Estamos agora a utilizar coordenadas esféricas (ver secção 3.4.3). Existem três ciclos, um que desenha a parte de baixo, outro que desenha a parte de cima e por fim um que desenha as secções intermédias. Posteriormente o cálculo destes ângulos foi modificado para passar a ser calculado a cada iteração para evitar erros de vírgula flutuante (ver secção 3.4.4).

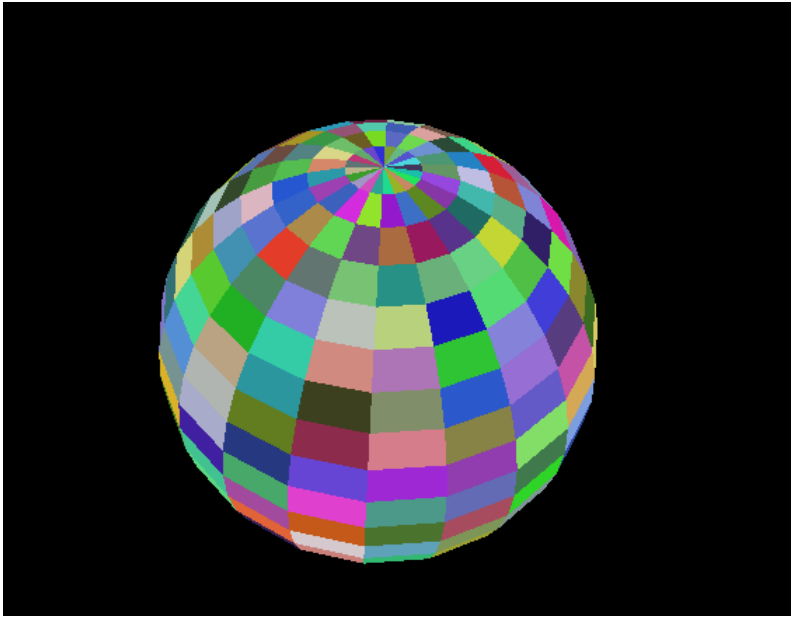


Figura 2: Esfera com 2 de raio, 20 camdas hotizontais e verticais

3.2.2 Segunda Fase

Em relação à primeira etapa, nós decimos organizar o nosso trabalho de modo a tornar as coisas mais simples, e para no futuro ser mais fácil a implementação. Para isso foram criadas algumas novas classes.

Criámos uma classe **CG_OBJ** que define um *Vertex Buffer Object* (sem índices). De forma simplificada, um *VBO* é buffer onde se guarda um conjunto de vértices correspondente a uma primitiva (ver secção 3.4.5).

Para compor vários *VBO* de modo a formar um objecto complexo foi criada a classe **Figuras**. Esta classe *static* apenas tem um *array* de **CG_OBJ** e métodos que compõem elementos desse array para formar figuras complexas. Estes métodos são utilizados pelo *renderScene()* para desenhar objectos construidos por composição de primitivas.

Foram também criadas sub-classes da classe **CG_OBJ** correspondentes às várias primitivas: **Plano**, **Cubo**, **Cilindro** e **Esfera**. De notar que a forma de desenhar as primitivas mudou e, como tal, a nova forma de desenhar primitivas difere consideravelmente da forma que constava na classe **Primitivas**. Além das quatro primitivas, foi adicionada a classe **SolidoRevolucao** que foi usada para desenhar copos (ver secção 3.4.8).

Também implementámos uma câmera FPS e alterámos a interacção através do teclado com a aplicação. De forma a aumentar a escalabilidade do projecto, criámos duas novas classes para o controlo da câmara e teclado. São respectivamente a classe **Camera** e a classe **Input**.

Espaço limite do Bar

Para simular o espaço do Bar, fizemos com que toda a cena fosse desenhada dentro de um cubo com as seis paredes visíveis.

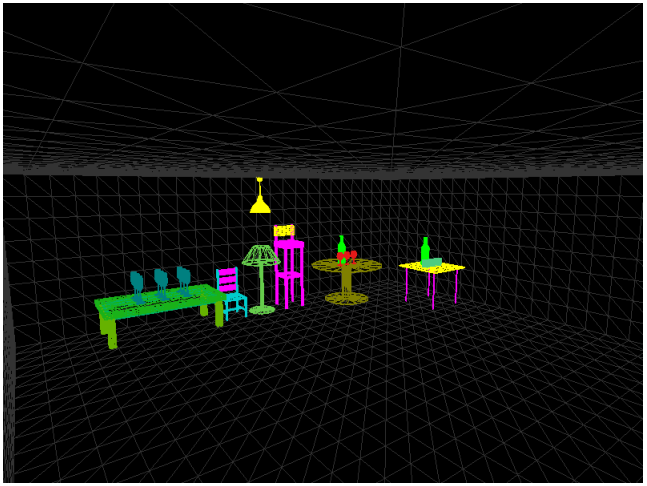


Figura 3: As seis paredes visíveis em modo linha

Objectos

Para além do espaço limite do bar, foi também pedido que fossem construídos alguns elementos que se podem encontrar num bar. Estes elementos são desenhados recorrendo aos métodos da classe **Figuras**.

Para esta fase desenhámos três **mesas**, sendo que uma delas tem a base e o tampo redondo, outra mesa **quadrada** e por fim uma mesa **rectângular** baixa.

A mesa **quadrada** é a mais simples das três, contendo quatro cilindros que constituem as pernas da mesa e um rectângulo para a superfície.

Temos uma mesa rectangular mais baixa que a quadrada. A mesa é formada por 9 cubos: 4 para as pernas, 4 para o reforço e 1 para a superfície da mesa.

Por fim temos a mesa **redonda**, que é composta por uma base de suporte inferior e pela superfície superior. A unir estes itens está um cilindro de raio inferior à base e ao tampo. Esta mesa é desenhada com recurso a Sólidos de Revolução (ver secção 3.4.8).

Outro elemento do Bar pedido eram as **cadeiras**, além de cadeiras optámos por também desenhar um **banco** alto. As pernas, assento e recosto da cadeira são cubos. O banco tem pernas cilíndricas e ambos o recosto e o assento são cubos.

Apresentamos agora os **copos** que criámos usando Sólidos de Revolução (ver secção 3.4.8 para detalhes sobre Sólidos de Revolução). Recorrendo à mesma técnica desenhámos também uma garrafa.

Finalmente o último item pedido eram candeeiros, aqui optámos por fazer dois estilos diferentes, um de tecto e outro de pé. Tanto o candeeiro de tecto e o *abajour* do candeeiro de pé são Sólidos de Revolução. O candeeiro de pé também é formado por uma composição de vários cilindros.

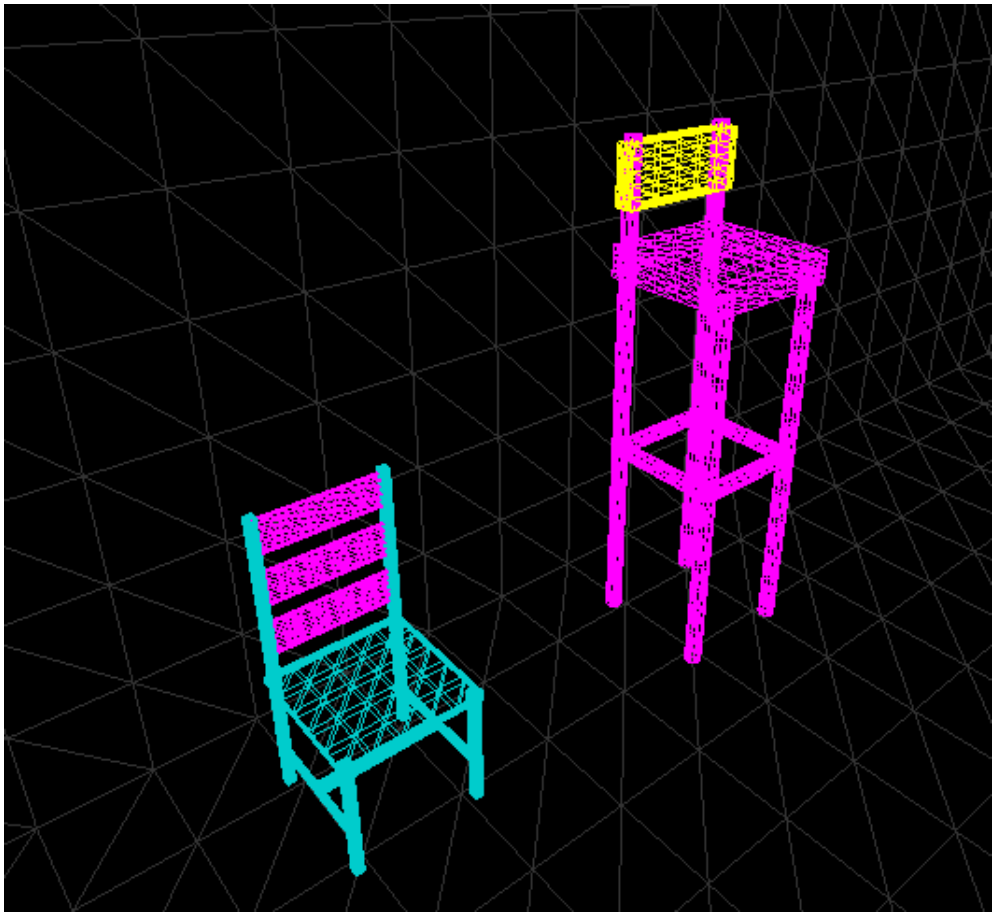


Figura 4: Cadeira normal e banco de balcão

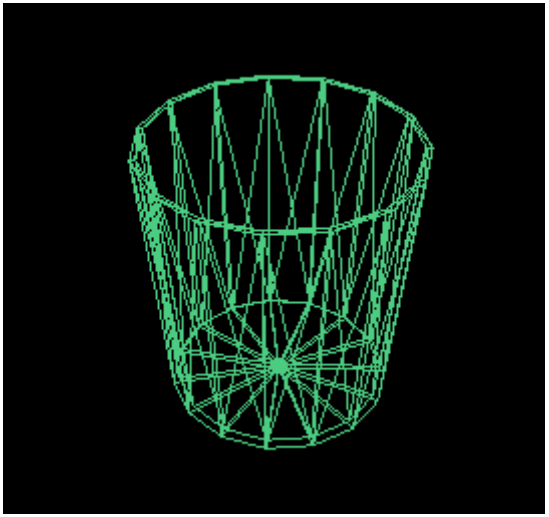


Figura 5: Copo normal

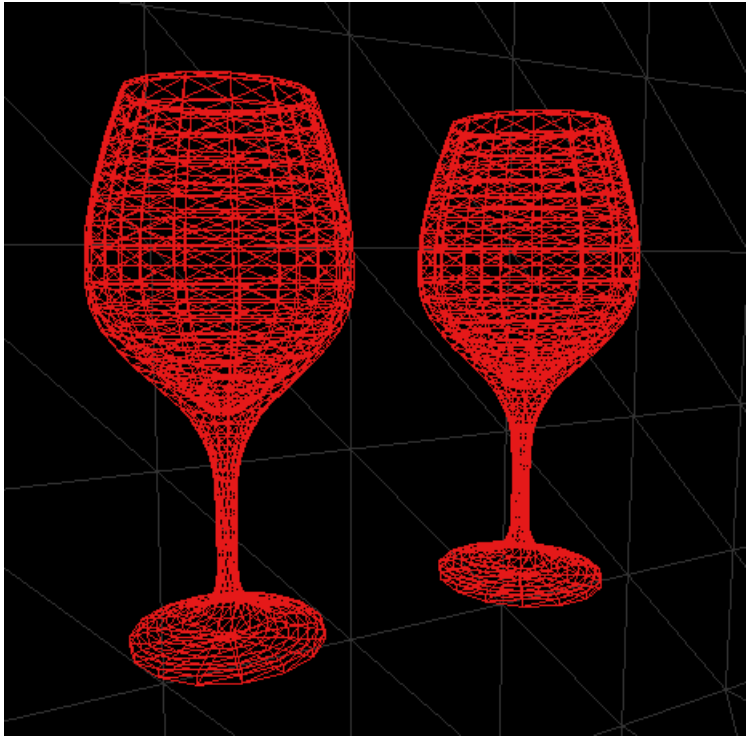


Figura 6: Copo de vinho

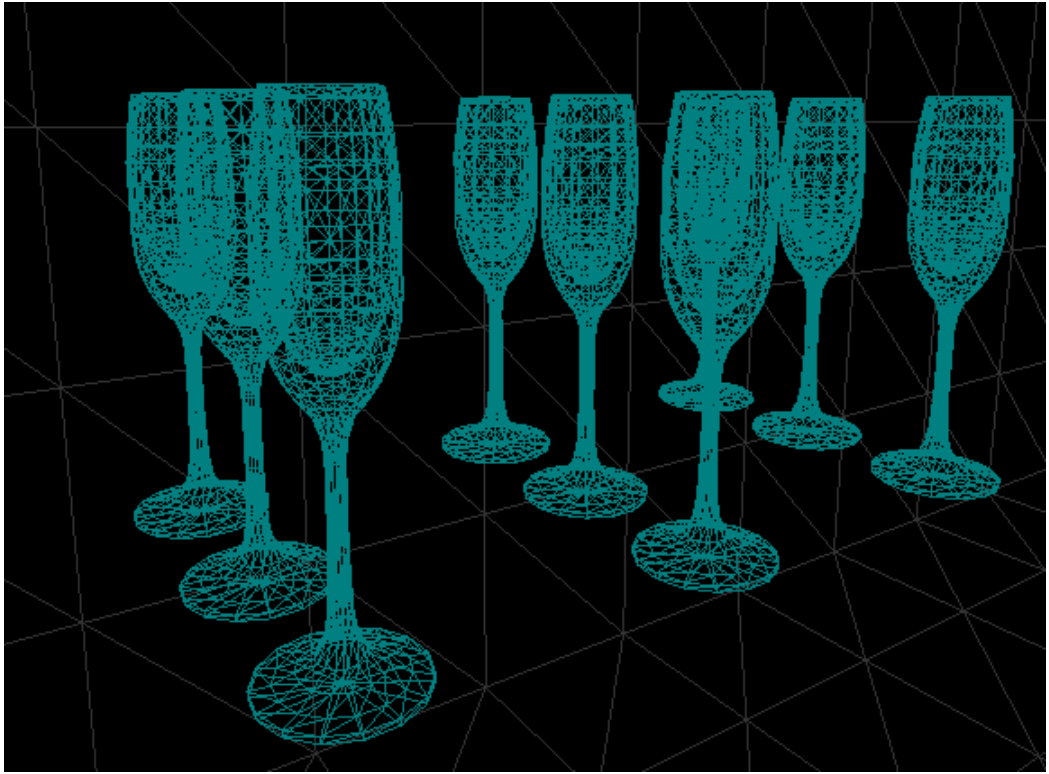


Figura 7: Copos de champanhe

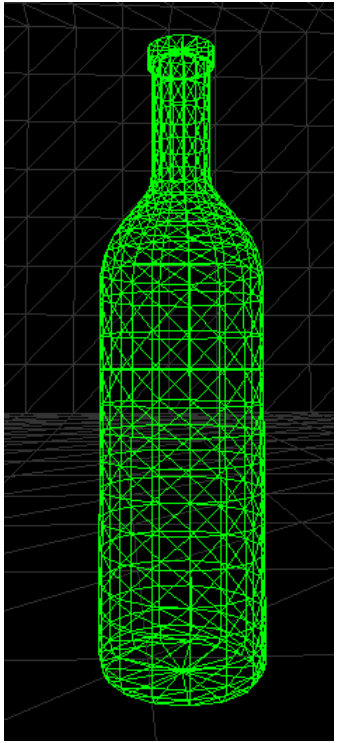


Figura 8: Garrafa

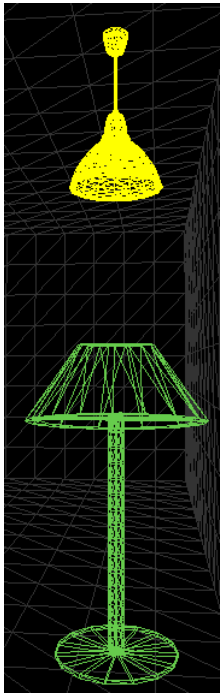


Figura 9: Candeeiro de tecto e de chão

3.2.3 Terceira Fase

Tal como já tinha sido referenciado anteriormente e no enunciado, um dos objectivos para esta etapa do projecto seria a utilização de *Vertex Buffer Objects* no desenho do Bar. Este método é uma optimização implementada nesta terceira fase do projecto, embora nós na segunda fase já tinhamos implementado os VBO's mas sem índices. Desenhando o Bar com este método, pretendemos acelerar o processo de desenho dos elementos gráficos. Os *Vertex Buffer Objects* são descritos em pormenor na secção **??** Como já foi dito na segunda fase, criamos uma classe **CG OBJ** onde são definidos os *Vertex Buffer Object*, nesta fase apenas adicionamos os índices que ainda não existiam.

VBO's nas Primitivas

Como para esta fase tínhamos de começar a preparar as nossas primitivas de forma a usarem texturas e iluminação. Desta forma, cada primitiva tem uma função *preencherVertices()* que preenche os *arrays* com vértices, normais e as coordenadas de textura e depois guarda os vértives no buffer da placa.

Esta função é chamada pela *guardarOBJ*, que cria o VBO, alocando espaço para os vértices, normais e coordenadas de textura. Um exerto de código do Cilindro.cpp:

```
void Cilindro::preencherVertices(){
...
for(int fatia=0; fatia < fatias; fatia++){
alpha = delta * fatia;
alphaDelta = delta * (fatia+1);

this->addVertex(&vi,0,altura2,0);
this->addVertex(&vi,raio * sin(alpha), altura2, raio * cos(alpha));
this->addVertex(&vi,raio * sin(alphaDelta), altura2, raio * cos(alphaDelta));

this->addNormal(&ni, 0, 1, 0);
this->addNormal(&ni, 0, 1, 0);
this->addNormal(&ni, 0, 1, 0);

this->addTextureCoord(&ti, 0.5, 0.5);
this->addTextureCoord(&ti, 0.5 + 0.5*cos(alpha), 0.5 + 0.5*sin(alpha));
this->addTextureCoord(&ti, 0.5 + 0.5*cos(alphaDelta), 0.5 + 0.5*sin(alphaDelta));
        ...
        }
```

```
Cilindro::Cilindro(float raio, float altura, int fatias, int seccoes){
this->raio = raio;
this->altura = altura;
this->fatias = fatias;
this->seccoes = seccoes;

this->guardarOBJ( 2 * this->fatias + this->fatias * this->seccoes * 2 );
```

Cada uma das Classes que antes foram primitivas (Plano, Cilindro, etc) têm um construtor, ao qual são passados os parâmetros necessários para ele calcular o número de triângulos que vai usar. Depois de calcular quantos triângulos vai usar, chama a função *guardarOBJ* com o número de triângulos como parâmetro para a *guardarOBJ* saber quanto espaço tem de alocar. Portanto, a *guardarOBJ* vai chamar a *preencherVertices* relativa à sub-classe em questão. No caso do Plano ia chamar a *preencherVertices* que está definida no plano.cpp.

Os objectos que constituiem o nosso Bar são vários VBO's juntos. A classe Figuras é quem cria os VBO's e depois tem métodos que fazem cadeiras, candeeiros, mesas e copos juntando os vários VBO's.

Iluminação e Texturas

Nesta fase como foi pedido no encunciado, implementamos as texturas e já conseguimos ter iluminação no nosso Bar. Para além das coordenadas de textura e das normais foram adicionados duas novas classes: **Texturas** e **Light**.

Figura 10: Objectos com primitivas já com as texturas e normais definidas

3.2.4 Quarta Fase

3.3 Organização do Projecto

-(DIAGRAMA DE CLASSES IMAGEM) -um pouco de texto

3.4 Optimização e Técnicas usadas

Aqui iremos falar das várias técnicas que usamos ao longo do nosso projecto e também de algumas optmizações que fizemos.

3.4.1 "Camadas"nas primitivas

3.4.2 Evitar mudanças de estado

3.4.3 Coordenadas Polares e Esféricas

3.4.4 Evitar erros de vírgula flutuante

3.4.5 Vertex Buffer Objects

] nao esquecer a geração dinamica de indices e fazer distinçao entre VBO de fase 2 e 3

3.4.6 Câmara FPS

3.4.7 Gestão de input

3.4.8 Sólidos de Revolução

3.4.9 Backface Culling

3.4.10 View Frustum Culling

A implementação do View Frustum Culling permite que não sejam desenhados objectos que não estão no campo de visão da cãmara.

3.4.11 Bouding Volumes Hierárquicos

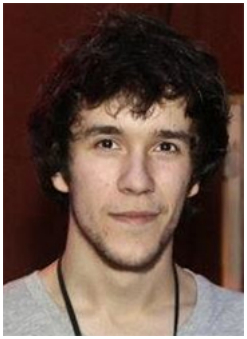
4 Navegação e Controlos

Um dos requisitos finais é a implementação de uma câmara de movimento livre. Nós inicialmente para visualizar as primitivas usavamos menus, em que o utilizador escolhia qual primitiva a visualizar separadamente, em conjunto com outras opções como colorir ou ver em modo de linhas ou todo preenchido. Na segunda fase já podiamos ver uma câmara FPS para ver os objectos construídos, e além disso ainda tinhamos um menu com as opções já atrás referidas. Na terceira fase já ouve alguma evolução na câmara e deixaram de existir os menus. Nesta etapa a câmara é de movimento livre através das teclas do teclado ('w','s','a' e 'd'), e recorrendo à tecla 'z' é possível alternar a visualização para modo preenchido ou só as linhas dos objectos. Com a ajuda da tecla '+' e '-' o utilizador pode aumentar e diminuir a velocidade da câmara. Portanto nesta fase final, a navegação efectua-se recorrendo às teclas 'w', 's' (respectivamente, movimento para a frente para trás) e 'a','d' (movimentação lateral para a esquerda e direita, respectivamente), equanto o rato serve para rodar e controlar o ângulo da câmara. Além disso, como foi feito anteriormente, é possível alternar de estados para preenchido e só linhas, e também aumentar ou diminuir a velocidade da câmara. No canto superior esquerdo, é possível visualizar a quantos FPS está a correr. Nesta fase acrescentamos, além do 'z', o 'x' que activa e desactiva o View Frustum Culling com o primir da tecla. Todas estas funcionalidades são implementadas nas funções exclusivas do OpenGL para a utilização do rato e teclado. As funções de implementação do teclado e do rato encontram-se no Camara.cpp e Input.cpp.

5 Conclusão

Com a chegada do fim deste relatório chegou a altura de se tecerem as conclusões finais.

6 Fotos



Bruno Ferreira
A61055

Serafim Pinto
A61056