Projeto Final CIC270 - Universe Search

Leonardo V. Ferreira[1], **Lucas P. de Souza**[1], **Thiago S. Pereira**[1]

[1]Instituto de Matemática e Computação – Universidade Federal de Itajubá (UNIFEI)
CEP – 37.500-903 – Itajubá – MG – Brasil

{leovieiraferreira, lucaspraca80, thiago.itajuba}@unifei.edu.br

Resumo. *Esse é um relatório sobre o projeto "Universe_Search" para a disciplina de Computação Gráfica. Aplicando todo o conteúdo aprendido em sala de aula, a proposta criada é a construção de um universo de formas geométricas 3D, simulado e interativo, com o OpenGL moderno, aplicando conceitos de luz, translação, rotação, escala e projeção. A aplicação final é totalmente configurável e possui grande potencial de desenvolvimento futuro à medida que novos conceitos são aprendidos.*

1. Introdução

Esse relatório reporta a construção do projeto final da disciplina "Computação Gráfica" ministrada pela docente Elisa de Cassia Silva Rodrigues. O projeto propõe a construção de uma aplicação que integre todos os conceitos apresentados em sala de aula sobre a biblioteca moderna da OpenGL: modelos de objetos 3D e imagens, translação, rotação, escala, projeção e iluminação.

Para cumprir a proposta do trabalho foi desenvolvido uma aplicação 3D que simula um universo e formas geométricas. O objetivo é a aplicação apresentar diferentes modelos 3D que interajam com o espaço, cumprindo com todos os requisitos da proposta. O programa é totalmente configurável e permite que o usuário explore o espaço gerado com uma nave. Um grande aliado na construção do projeto foi o site da LearnOpenGL[1], que forneceu dicas e tutoriais bem explicativos para problemas que tivemos no caminho.

O ambiente de desenvolvimento foi construído com a utilização de uma máquina virtual Linux Mint, um dos requisitos do projeto. A biblioteca OpenGL possui recursos com mais fácil acesso a esse tipo de sistema operacional. Também foi utilizado a IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado, em português) VSCode.

Este relatório está dividido da seguinte maneira:

- A Seção 2 detalha como foi feito a implementação do projeto e a aplicação dos requisitos.
- Na Seção 3 é apresentado quais são as funcionalidades da aplicação e seus comandos.
- Por fim, a Seção 4 traz as conclusões dos autores a respeito do projeto realizado.

2. Implementações

Nessa seção serão apresentadas as principais técnicas e características do programa. Na tabela 1, podemos ver um resumo das técnicas utilizadas e onde são aplicadas.

[1]https://learnopengl.com/

Table 1. Tabela de técnicas e aplicações

Técnica	*Aplicações*
Transação	Movimentação da nave e posicionamento dos objetos em tela
Rotação	Efeito de rotação dos planetas e da nave
Escala	Definição do tamanho das esferas e da nave
Iluminação	Acompanha a nave como ponto de luz para os planetas
Modelos 3D	Naves e planetas
Projeção	Toda a cena usa a projeção perspectiva

O projeto é construído com base em objetos 3D, assim a projeção perspectiva é aplicada para a melhor visualização dos objetos pelos usuários. Essa é definida no início de toda cena através dos vetores e matrizes do *model* (modelo do objeto), da *view* (posição da câmera), vetor também utilizado para a exploração do espaço, e a *projection* (área de projeção), um paralelepípedo baseado no tamanho da tela, sendo os 3 combinados dentro do *vertex shader* para renderização na tela.

A matriz modelo é formada pelas matrizes de translação, rotação e escala, e muda de objeto para objeto, já a visualização e projeção se mantêm a mesma durante todo o *display*.

A luz dos objetos é criada através da junção dos modelos de luz ambiente, difusa e especular, uma técnica conhecida como Iluminação de Phong. Para isso é fornecido um vetor de cor para a luz, definido em (1.0, 1.0, 1.0) para luz branca, posição da luz, baseada na posição da nave, e posição da câmera seguindo o que já é definido na *view* da projeção.

Das principais modelagens dentro do programa, podemos citar a cena do menu de escolha da nave, a criação da nave, a criação dos planetas e a criação do fundo estrelado.

2.1. Menu de Escolha

A primeira cena que é apresentada para o usuário é a de escolha da nave, o programa utiliza da função de geração de naves, explicado na seção 2.3, para gerar naves de diferentes cores a qual podem ser selecionadas pelo usuário. Usando as teclas A e D, ele é capaz de navegar pelos índices do vetor de naves até achar a que mais lhe agrada. E ainda tem a opção de definir uma escala para nave utilizando as teclas W e S, variando em 3 níveis de tamanho da nave.

Uma vez que a customização da nave é feita, o usuário pode aperta a tecla “enter” para iniciar a cena principal, que usará o índice e a escala escolhida para reproduzir a nave.

2.2. Estrelas

As estrelas que podem ser observadas no mapa são controladas por um VAO (*Vertex Array Object*) próprio para cada estrela. Isso permite que controlemos as cores e posições das estrelas individualmente, levando ao efeito de brilho que é visto no programa (Figura 1).

Todas as estrelas são espalhadas em uma área 38x38 da tela, com valores de x e y entre -19 e 19 definidos pela função 1. Dessa maneira, mesmo que nem todas as estrelas apareçam na visão do usuário inicialmente, podemos controlar a câmera e a nave, como é mostrado na Seção 2.3, para navegar pelo espaço.

Figure 1. Estrelas definidas no programa

$$P_x = (((float)rand()/(float)(RAND\_MAX)) * 38) - 19 \quad (1)$$

As cores das estrelas são mantidas em um vetor separado dos objetos estrelas em si. Ele é composto de valores entre 0 e 1 e representa a intensidade do branco de cada estrela. A função "changeStarsColor" tem uma chance de 0 a 100 de ser chamada sempre que a função de estado "idle", programa sem mudança de estado, está sendo executada. Quando isso acontece, cada estrela tem uma chance de 0 a 1000 de mudar de cor, o que dá o efeito de estarem piscando na tela. Ambos esses valores podem ser alterados nas configurações do programa (Seção 3).

As estrelas também têm tamanho variado entre 2pts a 4pts, utilizando a função "glPointSize". Apesar de ser aleatório, esse valor não muda no decorrer do programa.

2.3. Nave

A nave é construída com base em uma pirâmide de base triangular, assim sendo feita pela junção de 4 primitivas triangulares. A função que realiza a construção dos vértices também recebe um inteiro que representa a coloração da nave, que pode ser entre:

0 - Magenta RGB(1.0, 0.5, 1.0)
1 - Ciano RGB(0.5, 1.0, 1.0)
2 - Verde Limão RGB(0.5, 1.0, 0.5)
3 - Amarelo RGB(1.0, 1.0, 0.5)
4 - Vermelho RGB(1.0, 0.5, 0.5)

Cada face recebe uma versão do RGB 0.1x menor para a melhor visualização da figura 3D sem interferência da luz (Figura 2). O usuário consegue controlar a câmera, *view* da projeção, para caminhar pelo mapa, a posição da nave é transladada para o inverso da posição dessa câmera, assim mantendo o objeto centralizado na janela e dando a impressão de movimentação pelo espaço.

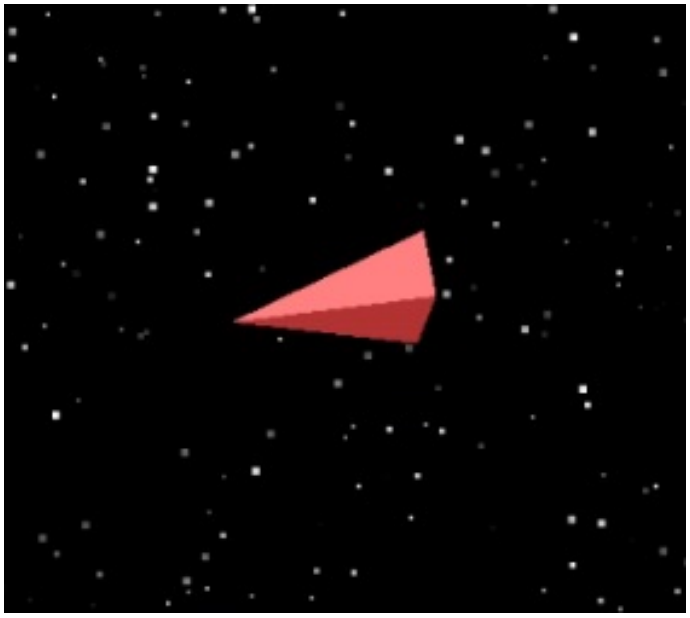

Figure 2. Renderização de uma nave vermelha

A nave também possui uma rotação fixa, que acompanha a direção da qual o usuário está a movimentando. Isso permite uma melhor impressão de navegabilidade da nave e um belo efeito visual.

2.4. Planetas

A construção dos planetas é um pouco mais complexa, já que possuem mais configurações e utilizam do efeito da luz em sua projeção. Existem três tipos de planetas: cúbicos, piramidais e esféricos.

Em qualquer uma das formas de planetas, a função de construção recebe 9 parâmetros, 10 no caso das esferas:

- unsigned int VAO - Vertex Array Object
- unsigned int VBO - Vertex Buffer Object
- float positions[] - guarda a posição retornada pela função
- float X - Posição do objeto em x
- float Y - Posição do objeto em y
- float R - Tonalidade de vermelho
- float G - Tonalidade de verde
- float B - Tonalidade de azul
- float size - Tamanho do objeto (recomendado entre 2.0 e 0.5)
- int vertices (Apenas Esfera) - Retorna o número de vértices do .obj renderizado.

Por padrão esses valores são aleatorizados para cada planeta definido nas configurações(Seção 3), Assim gerando planetas em diferentes posições, cores e tamanhos.

Para a criação dos planetas cúbicos, são gerados 2 triângulos de vértices que representam cada um de suas 6 faces. $Size$ é usado como medida na criação dos vértices,

Figure 3. Renderização de um planeta cúbico

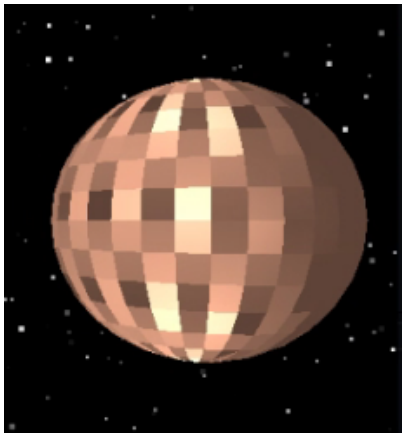

Figure 4. Renderização de um planeta esférico

assim os pontos (-1, 1) e (1, 1) seriam definidos como $(-size, size)$ e $(size, size)$ respectivamente no exemplo. Isso é o que permite o controle do tamanho dos objetos e a formação equilátera de um cubo.

Para a criação dos planetas piramidais, usamos o mesmo princípio, porém agora precisamos trabalhar com 3 valores diferentes de coordenadas x,y e z , ficando defino com $(-size, 0, size)$.

A criação de esferas em OpenGL é mais complexa, já que também precisamos definir sua forma usando primitivas triangulares. No programa, a geração dos vértices é definida utilizando da ajuda de um .obj que já contém as informações de posição (x,y,z), texturas, que não foram utilizadas, e as normais para o cálculo da iluminação. O código utilizado para a leitura dos objetos é baseado no artigo da Opengl-Tutorials [2]. Para a definição de tamanho utiliza-se da escala de objetos como ferramenta de controle.

Todos os planetas, junto com a posição de seus vértices recebem atributos para cores, definidas pelos parâmetros R, G e B da função, e retornam pelo vetor $positions$ os valores para translação e rotação do objeto em cena. Nas Figuras 3, 4 e 5 podem ser vistos exemplo dos planetas criados.

[2] http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/

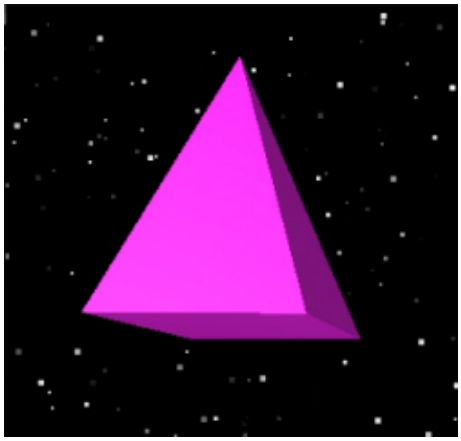

Figure 5. Renderização de um planeta piramidal

Além disso, todos os planetas possuem uma normal, que é utilizada no *fragment shader* para definição de iluminação nos objetos. A iluminação utiliza a fórmula de Phong, com posição da luz baseada na posição da nave. Um a cada dois planetas, também tem a capacidade de uma rotação fixa sobre o eixo y.

3. Manual de Funcionalidade

Nessa Seção é abordado as principais funcionalidades que o programa fornece aos usuários. O objetivo do projeto é o desenvolvimento de um universo interativo totalmente configurável. Assim, no começo do arquivo código do programa, existe a separação completa de todas as principais configurações globais que podem ser alteradas pelo usuário, como número de planetas de cada tipo, velocidade de rotação, e outros parâmetros visualizados na tabela 2.

Table 2. Tabela de técnicas e aplicações

Parâmetro	*Descrição*
win_width	Largura da janela
win_height	Altura da janela
STARNUMBER	Número de estrelas em cena
colorChangeRate	Velocidade a qual as estrelas piscam
rotationInc	Velocidade de rotação da nave
CUBEPLANETSNUMBER	Número de planetas cúbicos
rc_inc	Velocidade de rotação dos planetas cúbicos
SPHEREPLANETSNUMBER	Número de planetas esféricos
rs_inc	Velocidade de rotação dos planetas esféricos
PIRAMIDPLANETSNUMBER	Número de planetas piramidais
rp_inc	Velocidade de rotação dos planetas piramidais

Ao iniciar a aplicação, será aberto uma tela somente com a nave, o menu de escolhas (Seção 2.1). Nessa cena o usuário utiliza das teclas ‘A’ e ‘D’ para alternar entre as 5 cores possíveis de nave, com as teclas ‘W’ e ‘S’ também pode variar o tamanho da nave entre os três valores pré definidos. Ao clicar na tecla “Enter”, as escolhas do usuário serão confirmadas e a cena principal será carregada.

A partir dessa cena, os controles tomam uma nova função e agora as teclas ‘W’, ‘A’, ‘S’ e ‘D’ podem ser utilizadas para mover a nave para cima, para esquerda, para baixo e para a direita respectivamente. Na cena principal o usuário consegue interage com a cena navegando pelo universo criado, visualizando as formações criadas.

4. Conclusão

Uma das maiores limitações foi o sistema operacional; a instalação do OpenGL no Windows não funcionou, o que obrigou a usar o sistema Linux para programar. Além disso, apenas dois dos membros podem utilizar o Linux em uma maquina virtual e ainda assim houveram problemas na instalação do OpenGL e suas bibliotecas.

Outra dificuldade foi as funções do OpenGL não possuírem documentação intuitiva sobre suas aplicações. A função “glutSolidSphere” da biblioteca GLUT, por exemplo, deveria desenhar uma esfera na tela, mas não soubemos aplicá-la corretamente e tivemos que importar uma esfera em arquivo .obj para ser renderizada no programa.

Dentre os problemas não resolvidos, destacam-se a falta de um texto no menu de escolha da nave, que tentamos colocar usado a biblioteca “freeType” que deu falha na leitura, falhas na aplicação da iluminação na esfera importada, gerando *bugs* visuais na superfície da esfera, como pode ser visto na Figura 4. E ainda, o foco de luz altera parcialmente a iluminação do objeto, gerando conflitos entre a luz especular e difusa.

No final, a construção do projeto foi muito divertida, já que interage com um assunto tão interessante como a modelagem gráfica e o universo. A escolha desse tipo de projeto ainda permitiu a exploração de todas as primitivas e conceitos iniciais do OpenGL. A aplicação final ainda fica como um modelo que pode ser cada vez mais desenvolvido e aprimorado à medida que novos conceitos do OpenGL são explorados, o que permite um aprendizado mais interativo e completo.

ANEXO: `https://drive.google.com/file/d/1NhGcfy_fY9fn9RTvaipP0S1cgNZxxRyR/view?usp=sharing`