Universidade Estadual de Campinas
UNICAMP
Faculdade de Engenharia Elétrica e da Computação

IA725A

Projeto Final OpenGL
Cena Foto Realística

Coordenadora:

Profa. Wu Shin Ting

Alunos:

Daniel Scalioni Carvalho 070548

Sumário

1 Introdução

1.1 Motivo do projeto

Com o intuito de fixar e finalizar o aprendizado sobre a maioria dos métodos ensinados e utilizados em OpenGL foi definido o desenvolvimento de projetos, e este é um deles.

Como premissas, o projeto deveria levar em consideração o maior número de métodos ensinados inserindo-os em uma cena foto realística, que de acordo com (1), se resume a uma pintura que transporta, com bastante fidelidade, imagens originalmente obtidas com uma câmera fotográfica.

Deste modo, foi escolhida uma imagem de um filme conhecido, conforme Figura 1, e o projeto aqui apresentado, tenta retratar com certa propriedade esta imagem.

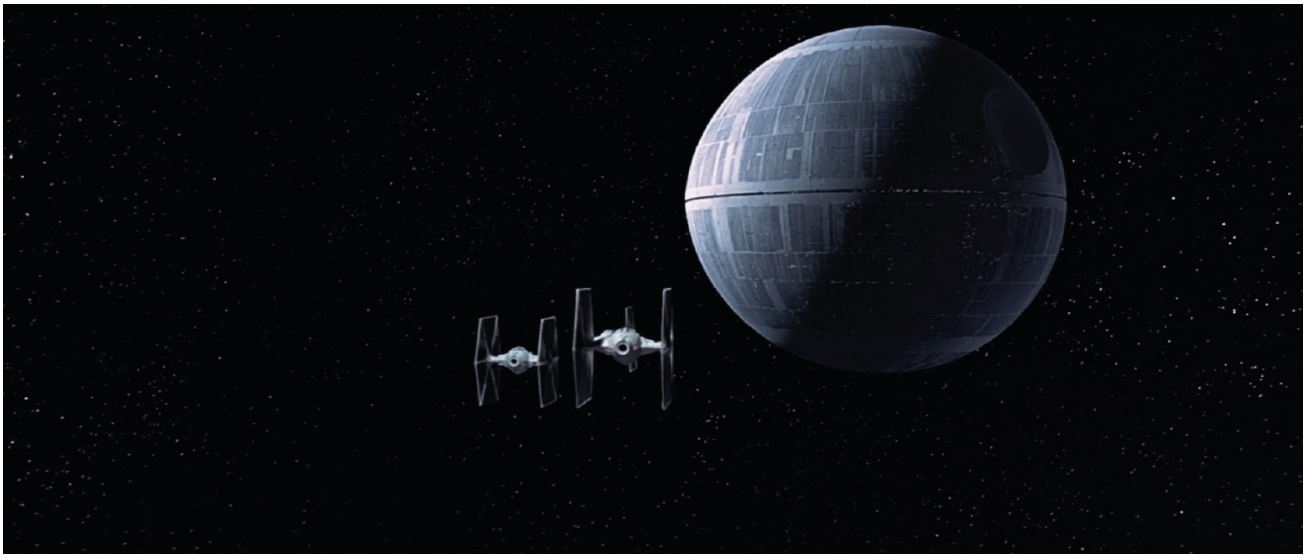

Figura 1 - Figura utilizada para o projeto, obtida no site (2).

2 OpenGL

2.1 O OpenGL

Como base para a disciplina cursada, foi escolhida a plataforma OpenGL para aplicar as metodologias aprendidas, sendo umas das referência utilizadas o livro (3) apontado na bibliografia.

O OpenGL é uma biblioteca gráfica muito conceituada utilizada para desenhar objetos em três dimensões utilizando o hardware GPU (placa de vídeo). Este hardware foi desenvolvido para processar massivamente e paralelamente os cálculos necessários neste processo de ilustração, já que a quantidade de dados é muito grande, se comparado a softwares normais.

Visando maior velocidade no processamento, este hardware utiliza-se das dezenas de processadores que possui para o processamento paralelo dessa grande gama de dados.

Para que este processo tenha o maior rendimento, a biblioteca possui passos que são executados internamente na GPU para que o usuário possa visualizar em sua tela em 2D, o objeto ou a cena de objetos, que são dimensionados, coloridos, projetados, iluminados, rasterizados, texturizados entre outros métodos aplicáveis.

2.2 PipeLine OpenGL

Este passo-a-passo é usualmente chamado de pipeline do OpenGL. Este pipeline é realizado, conforme a Figura 2, dentro da GPU, que em cada passo, é possível informar à GPU o modo com que será processada cada informação de entrada (vertex data). Esta forma é definida em arquivos chamados Shaders que são compilados em run-time, que são incluídos no pipeline nos pontos em laranja.

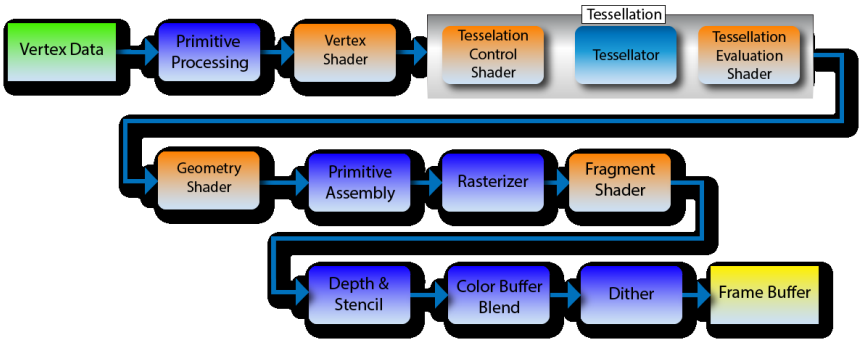

Figura 2 - Pipeline do OpenGL processado na GPU

Na CPU é necessário realizar todas as configurações para que este pipeline seja processado conforme a forma prevista. Para isso, deve-se projetar todos os Shaders, realizar as configurações na biblioteca OpenGL e enviar os dados dos objetos e métodos utilizados.

2.3 Configuração no Visual Studio 2015

Para o desenvolvimento do projeto, foi necessária a utilização das bibliotecas glut e glm. Sendo o primeiro a biblioteca padrão OpenGL e o segundo uma biblioteca matemática para facilitar a utilização de modelos matriciais e suas interações, como multiplicação vetorial e inversas.

Para tanto, foi encontrado no site definido em (4) uma facilidade para o emprego desta biblioteca, que é a utilização da ferramenta NuGet do Visual Studio. Desta forma foi possível utilizar sem muita dificuldade a biblioteca OpenGL com a facilidade do Visual Studio com seu debug prático e visualização do hardware utilizado durante a execução do programa, Figura 3.

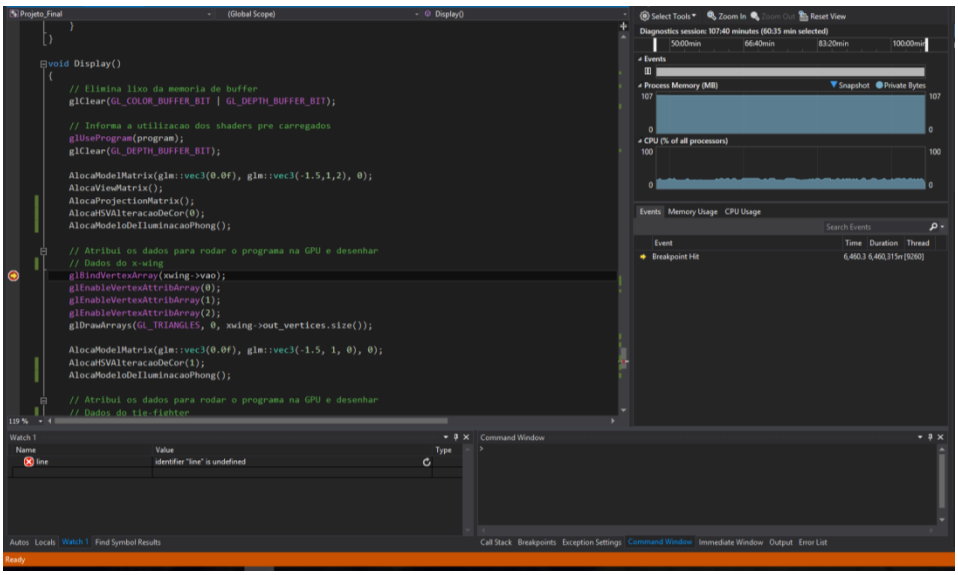

Figura 3 - Debug (seta amarela) e visualização da utilização de hardware no VS2015.

3 Modelagem Geométrica

3.1 Introdução

A modelagem geométrica se resume a obter vértices que representem o objeto desejado. Há várias metodologias para tal, já que há um equilíbrio entre quantidade de vértices, que quanto maior o objeto será mais bem definido, e tempo de processamento.

Neste projeto, como a necessidade era de uma imagem foto realística e devido ao pouco tempo de desenvolvimento do projeto, foi utilizado a abordagem de importação de modelos já prontos.

3.2 Importando Modelo Pronto

Uma das formas de exportação desses modelos é em OBJ, este tipo de arquivo, além de outros dados, guarda os diferentes vértices, normais das faces e os vértices de mapeamento da textura.

No entanto, não estão na forma necessária para utilização pelo OpenGL, sendo que o OpenGL utiliza-se de vértices alinhados de três em três para formação de triângulos. Desta forma pode ser possível utilizar o mesmo vértice mais de uma vez. Este alinhamento é definido no arquivo OBJ pelos elementos de face, que, por índices aos vértices já definidos, alinha de três em três os pontos que definem uma face triangular.

3.3 Ajustando Dados

De forma a ajustar os dados para a importação ao OpenGL, utilizou-se a referência (5) como base para a definição e importação dos arquivos OBJs utilizados no projeto.

Para tanto foi necessário utilizar o software Blender para transformar o modelo baixado da internet para o tipo de arquivo OBJ.

No projeto, este processo de importação foi encapsulado na classe ObjLoader, que abre o arquivo OBJ do modelo, realiza a leitura dos vértices, das normais e do mapeamento de textura. Ao fim, cria o alinhamento dos vértices em faces triangulares conforme o arquivo.

A seguir, os modelos que foram importados no projeto.

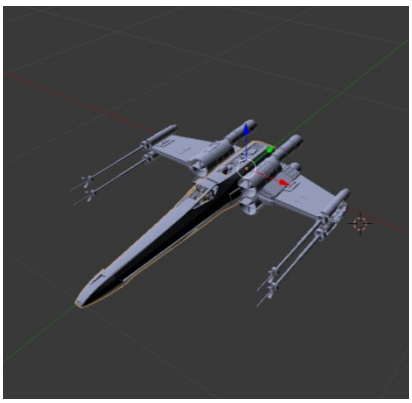

Figura 4 – Modelo do X-wing

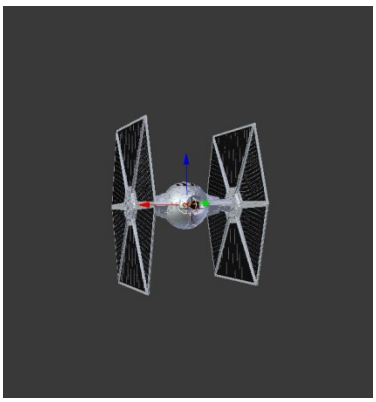

Figura 5 - Modelo do Tie-Fighter

Figura 6 - Modelo da Estrela da Morte

4 Transformações Geométricas

4.1 Intrudução

Após a configuração para utilização do OpenGL e modelagem do objeto inicia-se o processo de criação da cena que se exprime em alguns passos, indicados na Figura 7, presente no site (6).

Figura 7 - Matrizes de Transformação

Estes passos são realizados no Vertex Shader, no qual a partir dos vértices que definem os objetos a serem exibidos, são calculadas as posições finais dos objetos dentro da cena.

À esquerda estão os vértices que formam as faces, normalmente, triangulares do objeto. A matriz do modelo é responsável por ajustar as transformações pertinentes somente ao objeto, sejam elas mudança de orientação, escala e distorção. A matriz do mundo é responsável por posicionar o objeto no mundo, normalmente esta etapa é realizada conjuntamente à matriz do modelo. A matriz de visualização é responsável por alterar a cena conforme a posição do observador. E a última matriz é a de projeção, que altera a perspectiva de visualização da cena, considerando o observador próximo ou distante do objeto.

4.2 Matriz do Modelo

No projeto, foi programada uma função que a partir dos parâmetros, baricentro, posição final e rotacionando ou não, do objeto uma matriz diferente é construída.

Estas matrizes são então enviadas à memória da GPU por referência para serem utilizadas em cada objeto desenhado.

A seguir uma figura do posicionamento dos objetos utilizados no projeto. A vista está superior para uma melhor compreensão, mas o posicionamento é dado no referencial em 3D. O objeto “A” foi reduzido e aproximado para redução do custo computacional do projeto.

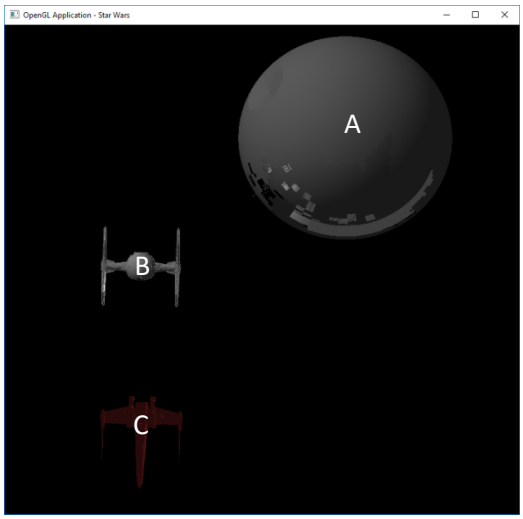

Figura 8 - Vista superior da cena com o posicionamento dos objetos

Visando maior realismo, foi incluída uma rotação no objeto “A” para que se parecesse presente no espaço como um astro rotacionando. A seguir, imagens mostrando esta interação.

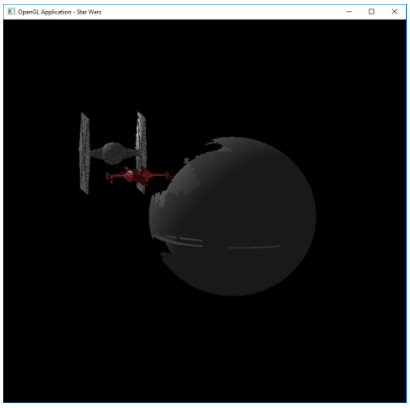

Figura 9 - Estrela da morte girando no estado A

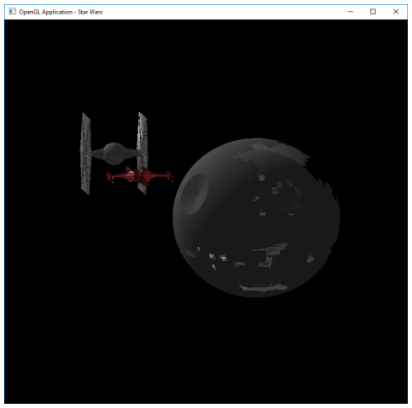

Figura 10 - Estrela da morte girando no estado B

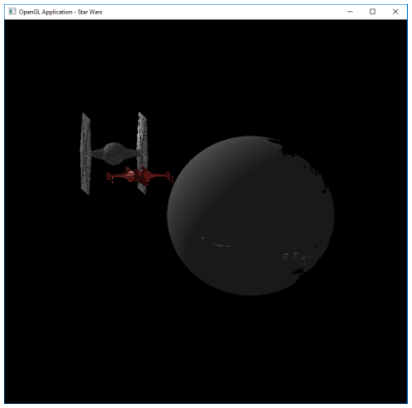

Figura 11 - Estrela da morte girando no estado C

A matriz do modelo é calculada na função `AlocaModelMatrix(glm::vec3 centro, glm::vec3 posfinal, int enable)`, o qual o primeiro argumento é o baricentro do objeto, para que o software saiba o ponto invariante uma vez que a modelagem não necessariamente adota o referencial do modelo em seu baricentro, o segundo argumento é a posição no espaço em que o objeto será posicionado e o último argumento é se o objeto rotacionará ou não.

4.3 Matriz de Visualização

A matriz de visualização foi determinada arbitrariamente para que contribuísse para uma melhor visualização da cena pensando na proximidade dos objetos menores.

Nesta matriz também foi incluída um modelo de rotação da cena pelo usuário chamada ArcBall. Neste modelo cria-se uma esfera virtual ao redor da cena e ao ser arrastada pelo clique do mouse, a esfera gira entorno de seu centro, girando toda a cena no mesmo vetor de rotação e ângulo em que a esfera gira.

Este método foi baseado no método descrito no site (7) e melhor representado pela Figura 12 - Representação do método ArcBall.Figura 12, na qual A e B são os pontos da tela em que o mouse foi arrastado gerando o ângulo alfa e o vetor de rotação V.

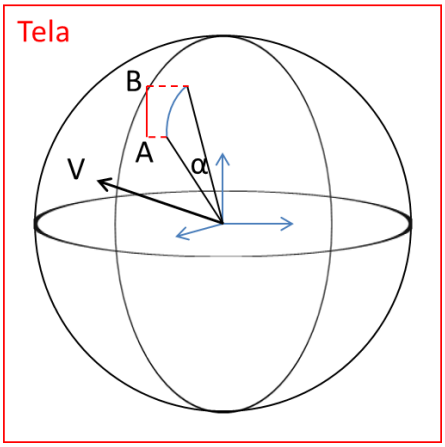

Figura 12 - Representação do método ArcBall.

A seguir figuras da utilização do método ArcBall no projeto, sendo possível visualizar a esfera virtual ao redor da cena. As setas presentes nas imagens representam o ponto inicial e final em que o mouse foi arrastado na tela. Para retornar à posição inicial, basta digitar “r”.

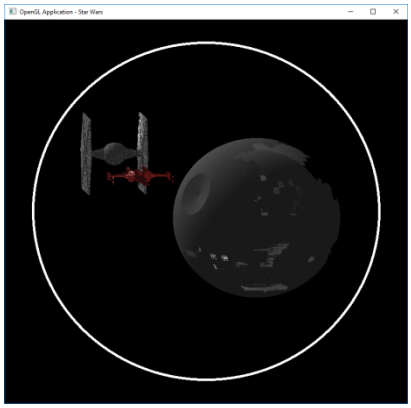

Figura 13 - Cena com a esfera virtual.

Figura 14 - Cena com esfera virtual rotacionada mediante arrasto para cima gerado pelo mouse.

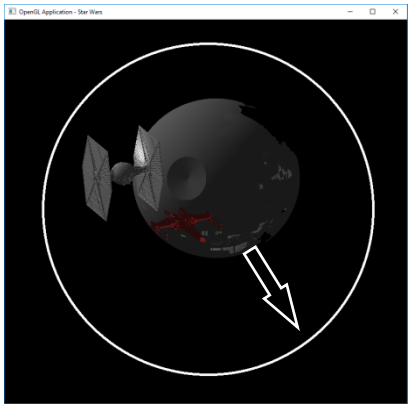

Figura 15 - Cena com esfera virtual rotacionada mediante arrasto em diagonal.

O ArcBall foi encapsulado na classe `Arcball` do projeto.

4.4 Matriz de Projeção

A matriz de projeção define a forma de visualização do observador em relação à cena.

Inicialmente a cena do projeto é definida com projeção ortogonal, a qual o observador está em um ponto muito longe da cena. Desta forma, não é representada a perspectiva de volume do objeto e nem entre um objeto e outro.

Ao clicar com o botão direito do mouse, um menu é adicionado a cena, onde é possível selecionar o seu modo de projeção. Ao mudar para um modo de projeção em perspectiva, a cena é alterada para representar a visão de um observador em um ponto próximo aos objetos.

A seguir, a representação de cada tipo de projeção.

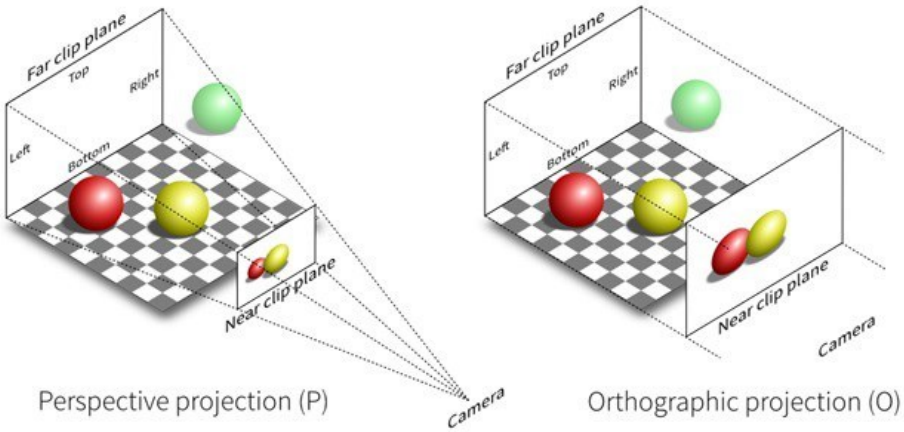

Figura 16 - Tipos de projeções.

A seguir a representação da alteração de projeção definida no projeto. Esta representação é definida na função `void AlocaProjectionMatrix()`, a qual é responsável pela alteração conforme seleção do menu. Sendo a primeira opção a projeção ortogonal, a segunda em perspectiva com um ponto de fuga, a terceira com dois pontos de fuga e a quarta com três pontos de fuga.

Figura 17 - Menu de projeções.

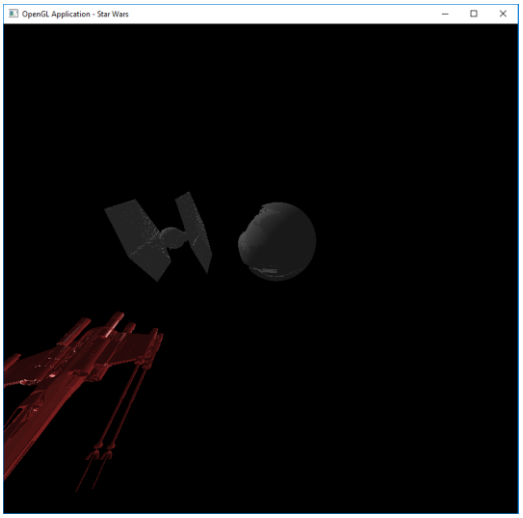

Figura 18 - Projeção em perspectiva.

5 Modelo de Cor

5.1 Coloração dos Vértices

Para coloração básica do objeto, utiliza-se, igualmente a definição dos vértices do objeto, vetores de três posições definindo as cores em RGB de cada vértice. Portanto, para cada vértice descrito, é possível atribuir uma cor a ele.

No projeto, para facilitar, todos os vértices do mesmo objeto possuem a mesma coloração. A coloração atribuída para o objeto A é `vec3(0.5f, 0.5f, 0.5f)` (cinza escuro), para o objeto B é `vec3(0.7f, 0.7f, 0.7f)` (cinza claro) e para o objeto C é `vec3(0.8f, 0.2f, 0.2f)` (vermelho escuro).

5.2 Alteração HSV da Iluminação

Como padrão a iluminação inicia-se na cor branca. O modelo de iluminação será retratado a seguir, no entanto, sua alteração será explicada aqui.

De forma a alterar a coloração iterativamente, foi adicionada uma função que altera a cor da iluminação conforme o modelo HSV de coloração. Este modelo utiliza-se da seguinte ideia para obter-se uma cor RGB, que é a utilizada em OpenGL.

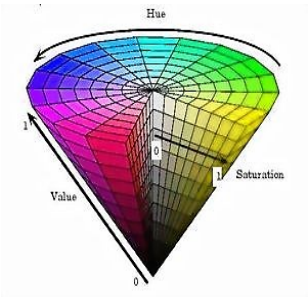

Figura 19 - Modelo HSV de coloração.

No projeto este efeito foi utilizado na iluminação do objeto B. Ao digitar “H” o valor de hue é aumentado e ao se digitar “h” seu valor é reduzido. O mesmo foi implementado para o valor de S (saturation) e o valor de V (value).

A seguir a visualização desta implantação no projeto.

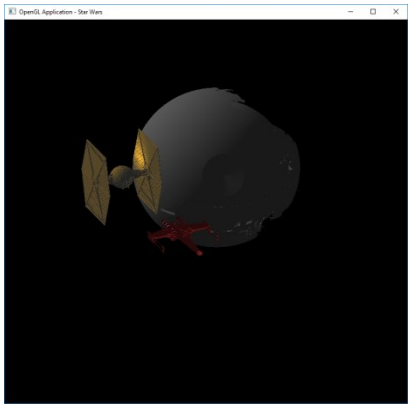

Figura 20 - Coloração do objeto B alterado pela redução na saturação e alteração de coloração (H) da iluminação.

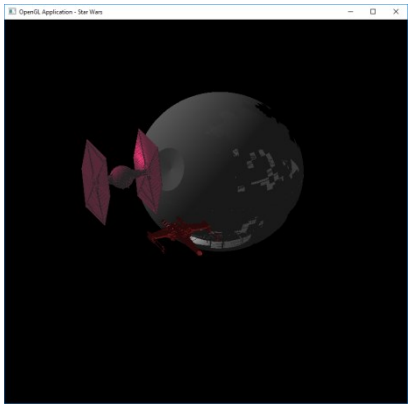

Figura 21 - Coloração do objeto B alterado novamente pela alteração de coloração (H) da iluminação.

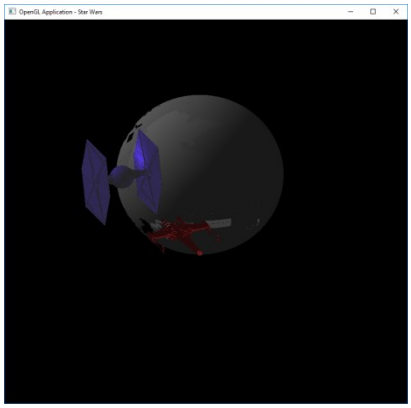

Figura 22 - Coloração do objeto B alterado pela alteração de coloração (H) da iluminação.

No teste, a redução da saturação é necessária pois a iluminação padrão está em branco, sendo a saturação máxima possível.

6 Modelo de Iluminação

6.1 Modelo de Phong e Blinn

O modelo de Phong é um modelo de iluminação local, o qual inclui uma variação na coloração do objeto considerando a posição relativa entre a fonte de iluminação, o objeto e o observador. Em sua implementação mais básica alguns efeitos são perdidos, por exemplo, a sombra dos objetos “B” e “C” em “A”.

Para a implementação deste método foram utilizadas as normais obtidas pela importação dos objetos. Essas normais influenciam na variação da coloração conforme o esquema abaixo.

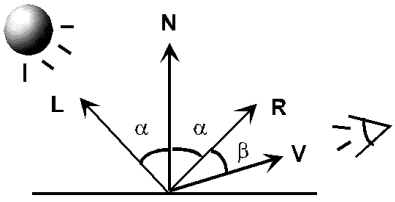

Figura 23 - Ideia do modelo de iluminação de Phong.

No projeto este método foi implementado, com base no método descrito em (8), no Fragment Shader, mas poderia ser utilizado no Vertex Shader. A diferença é que a interpolação dos dados seriam realizados na coloração calculada dos vértices ao invés da interpolação ser realizada entre as normais dos vértices.

O modelo de Blinn diferencia-se do modelo de Phong somente pela não utilização do raio refletido, mas sim de um raio médio entre o observador e o raio de luz. Desta forma o cálculo necessário é reduzido, já que para se calcular o raio refletido é muito mais custoso do que o cálculo do raio médio.

Como comparação, foi implementado no projeto estes dois métodos, sendo possível alternar entre eles somente apertando-se “B” para ativar o modelo de Blinn e “b” para retornar ao modelo de Phong. A seguir, duas imagens para comparação entre os dois modelos.

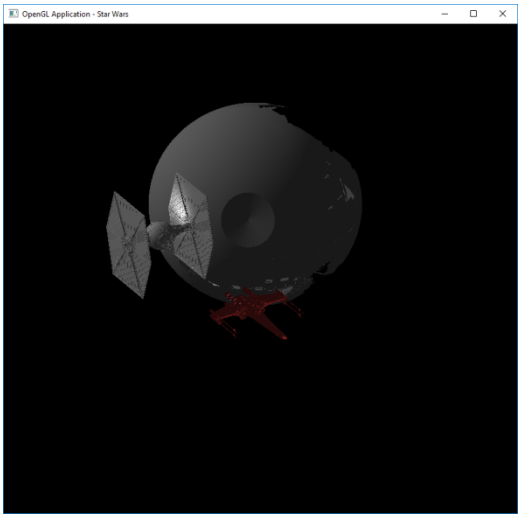

Figura 24 - Modelo de iluminação de Phong .

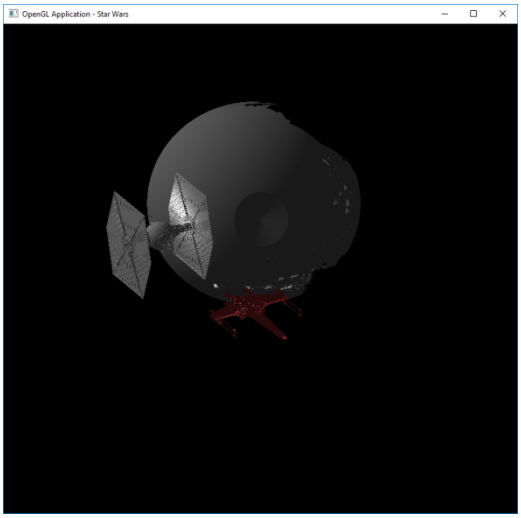

Figura 25 - Modelo de iluminação de Blinn.

A diferença pode ser mais bem notada na aba do objeto B, que no caso de Blinn ela fica levemente mais iluminada.

7 Conclusão

Como a proposta foi o desenvolvimento de um projeto foto realístico, comparando-se a imagem base com a imagem obtida pelo projeto, pode-se observar que ficaram razoavelmente parecidas.

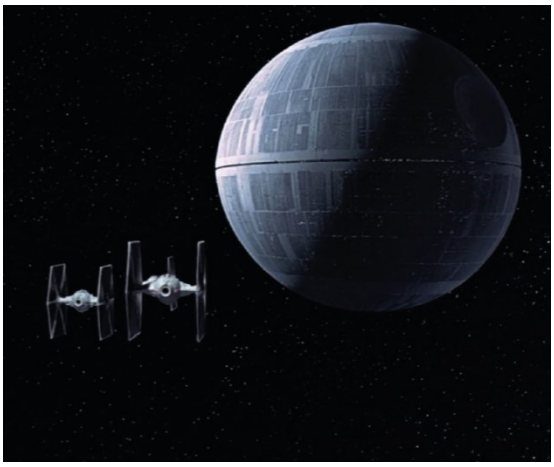

Figura 26 - Imagem básica.

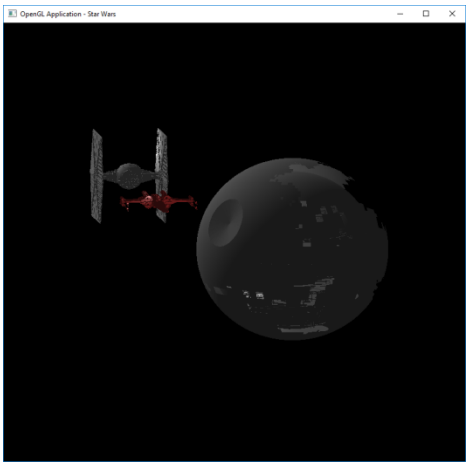

Figura 27 - Imagem gerada pelo projeto.

A inclusão de mais opções foi utilizada, pois o projeto teve como base as atividades desenvolvidas em sala de aula.

Ao final, o projeto pode ser melhorado incluindo-se textura e processamento pós-processamento.

8 Bibliografia

1. Fotorealismo. *Wikipédia.* [Online] [Citado em: 01 de 07 de 2016.] https://pt.wikipedia.org/wiki/Fotorrealismo.

2. Tie Fighter Approaching the Death Star. *The Disney Wiki.* [Online] http://disney.wikia.com/wiki/File:TIE-Fighters-approaching-the-Death-Star.png.

3. **Hughes, John F., et al., et al.** *Computer Graphics: Principles and Practice in C, 3rd edition.* s.l. : Addison-Wesley Professional, 2013. ISBN: 0321399528.

4. **Craitoiu, Sergiu.** SETTING UP OPENGL WITH VISUAL STUDIO USING NUGET. *IN2GPU.* [Online] 29 de 11 de 2014. [Citado em: 01 de 06 de 2016.] http://in2gpu.com/2014/11/29/setting-opengl-visual-studio-using-nuget/.

5. OpenGL-Tutorial. *Tutorial 7 : Model loading.* [Online] [Citado em: 25 de 06 de 2016.] http://www.opengl-tutorial.org/beginners-tutorials/tutorial-7-model-loading/.

6. Modern OpenGL. *IN2GPU.* [Online] [Citado em: 15 de 06 de 2016.] http://in2gpu.com/opengl-3/.

7. **LINELLO.** Tutorial of Arcball without Quaternions. *BrainTrekking.* [Online] 21 de 08 de 2012. [Citado em: 18 de 06 de 2016.] https://braintrekking.wordpress.com/2012/08/21/tutorial-of-arcball-without-quaternions/.

8. **Akenine-Moller, Tomas, Haines, Eric e Hoffman, Naty.** *Real-Time Rendering, 3rd ed.* s.l. : A K Peters/CRC Press, 2008. ISBN: 1-56881-424-0.