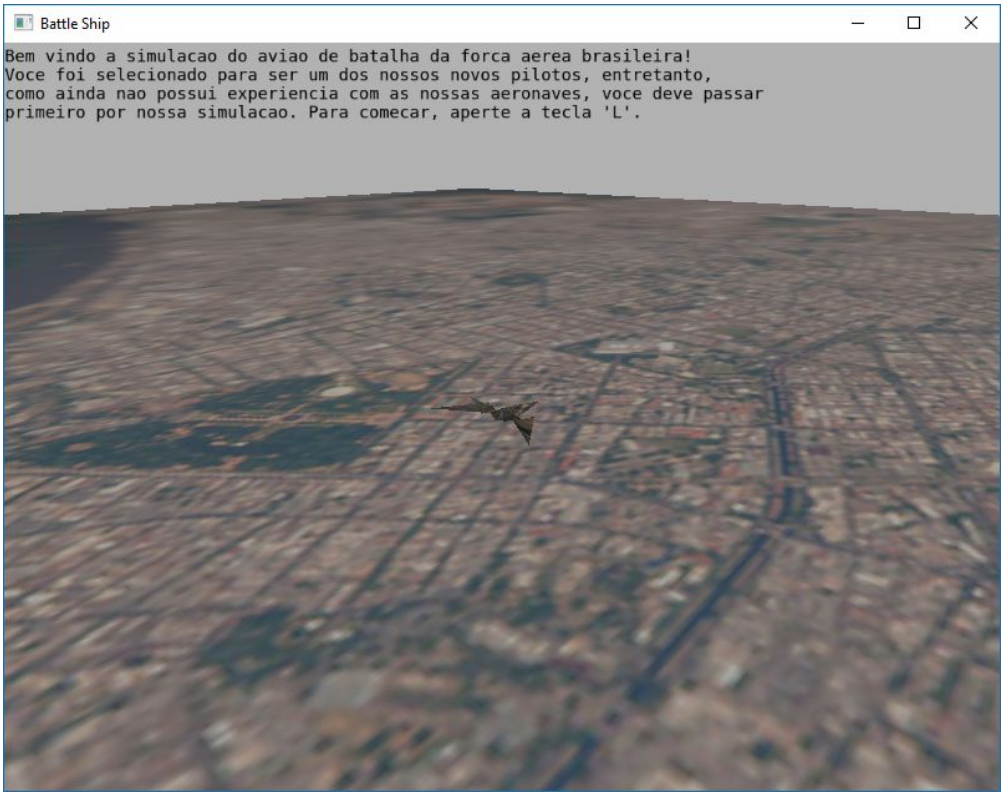
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF01047 - Fundamentos de Computação Gráfica
Professor: Eduardo Gastal

RELATÓRIO TRABALHO FINAL - SIMULADOR DE VOO

INTEGRANTES:
Leonardo Holtz de Oliveira
Mario Jose Kunz Filho

1. Introdução

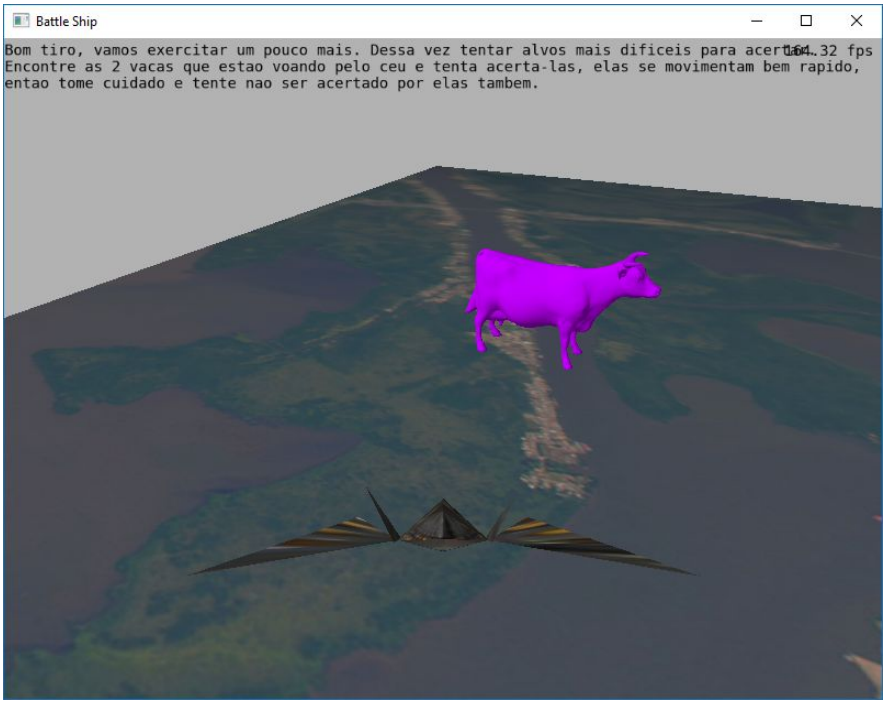
A aplicação desenvolvida se baseia em uma simulação de uma aeronave de batalha, que pode se movimentar pela tela e atirar em objetos pela cena. O jogador controla a movimentação e o tiro da aeronave por meio do teclado e mouse e tem por objetivo completar a simulação, através de objetivos que a mesma propõe.



2. Implementação da aplicação

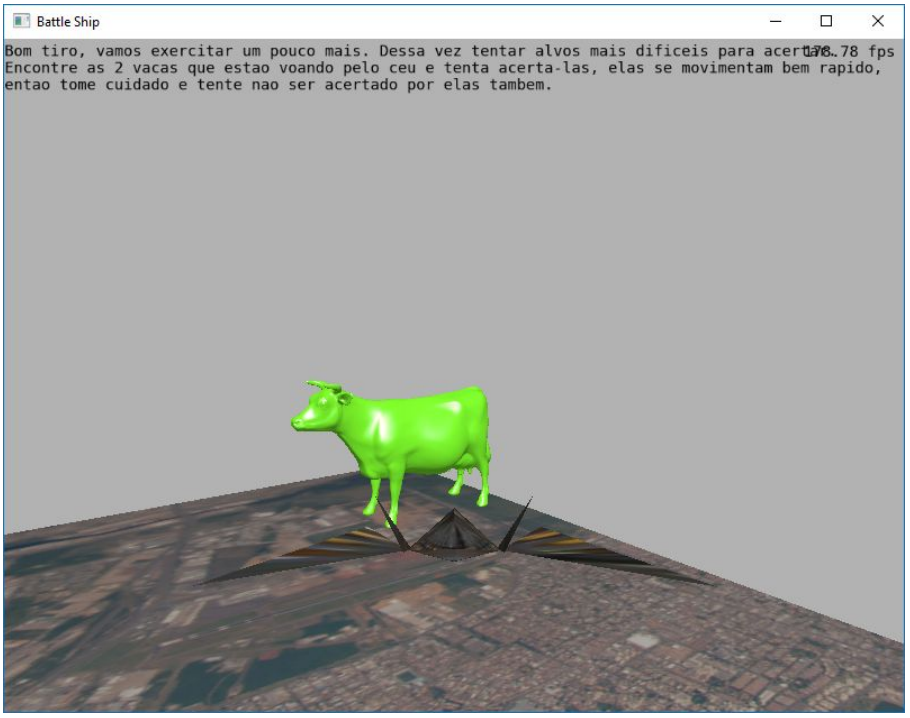
Utilizamos os códigos dos laboratórios 4 e 5 como base para nosso programa, pois eles já apresentam bastante das funções e definições que são necessárias para o trabalho final. A partir disso, implementamos todos os conceitos que estudamos em computação gráfica e que eram os requisitos mínimos para o trabalho:

- Inclusão de modelos poligonais complexos: Foi usado o modelo complexo cow.obj para a aplicação. Entretanto, outros modelos menos complexos também foram usados.
- Transformações geométricas de objetos virtuais: O usuário pode controlar as transformações geométricas da aeronave ao controlar ela por meio de rotações e translações, e da esfera, aumentando o fator de escalamento ao atirar nela.
- Controle de câmeras virtuais: Foi implementada uma câmera ‘look at’ no começo do programa para se ter visão completa da aeronave e do cenário onde ela irá voar. O usuário pode movimentar a câmera com o mouse e dar zoom com a “rodinha” dele. A free câmera funciona como a visão em terceira pessoa da aeronave, conforme o usuário controla a nave, a câmera a acompanha. A movimentação para a esquerda e para a direita fica por conta do teclado e a movimentação para cima e para baixo fica por conta do mouse.
- Duas ou mais instâncias de um objeto virtual: Os modelos cow.obj e sphere.obj são duplicados na aplicação. Os modelos da vaca funcionam como alvos que se mexem pela cena e os modelos da esfera servem como um objeto parado na cena e como o tiro da aeronave.
- Intersecção entre objetos virtuais: Foi implementado três tipos de intersecções. Uma cubo-plano, para a colisão da aeronave com o “chão” da cena. Uma ponto-esfera para a colisão do tiro com um objeto da cena. E uma colisão cubo-cubo, para a intersecção da nave com as vacas que voam.
- Modelos de iluminação: Um dos modelos da vaca (vaca roxa) teve sua iluminação por modelo difuso (Lambertiano) e avaliado para cada vértice (Gourad)



Vaca roxa feita com iluminação difusa avaliada para cada vértice

Para a outra vaca, foi feita a iluminação de Blinn-Phong, avaliada para cada pixel da imagem (modelo de Phong).



Vaca verde feita com iluminação de Blinn-Phong avaliada para cada pixel

- Mapeamento de texturas: Foi usadas duas texturas, uma para a aeronave, com detalhes em metal para simular a estrutura da aeronave, usando mapeamento esférico. Para o plano do chão, foi usada uma textura de uma imagem em satélite de Porto Alegre que utiliza mapeamento planar.
- Curvas Bézier: Os dois modelos das vacas se movimentam por curvas Bézier quadráticas, onde o 1° ponto também é o último ponto, o que faz com que as vacas façam uma movimentação circular.
- Animação de movimento baseada no tempo: As movimentações por curvas bézier e a movimentação da nave são feitas por meio do tempo, onde toda a movimentação é multiplicada por um ‘delta t’ que é a diferença de tempo entre a execução anterior e a execução atual do programa.

```
double tnow = glfwGetTime();
deltat = tnow - tprev;
tprev = tnow;
```

```
acelera_frente += 0.01*deltat;
```

```
if(rotateR == 1){
    rotationX+=0.1f*deltat;
}
if(rotateL == 1 ){
    rotationX-=0.1f*deltat;
}
```

```
// Atualizamos o parametro da curva
if(valor_param_vacal < 1.0f)
    valor_param_vacal = valor_param_vacal + deslocamento * deltat;
else
    valor_param_vacal = 0.0f;
```

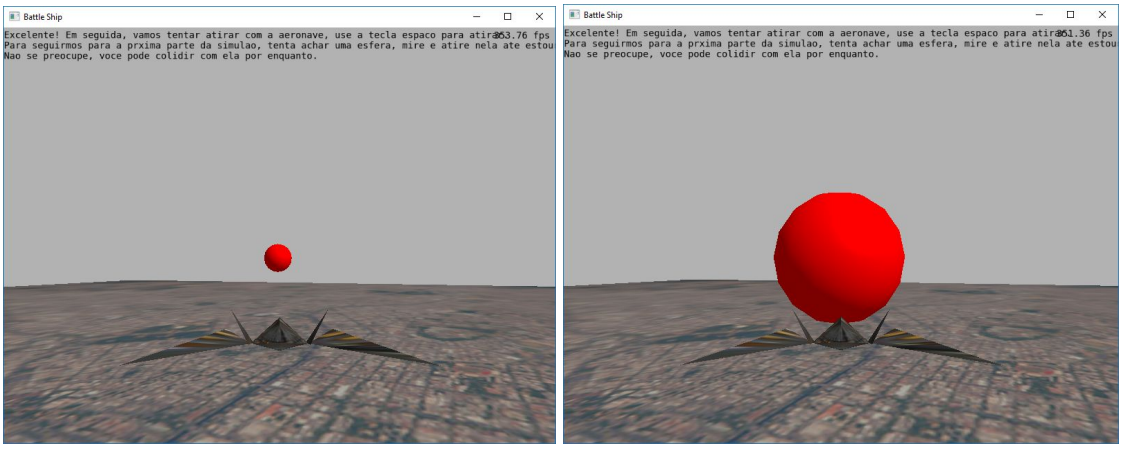
3. Manual

É recomendado o usuário executar o programa em tela cheia, pois aparecem textos na parte superior esquerda da tela na qual o usuário deverá ler para suceder com a simulação e completá-la.

Primeiramente, uma tela de boas vindas é introduzida, com uma câmera look at focando na nave. Para começar a simulação, o usuário deve apertar ‘L’.

A simulação pede para o usuário testar os controles da aeronave. A aeronave funciona por meio de ganho e perda de velocidade, a tecla ‘W’ aumenta a velocidade da aeronave e a tecla ‘S’ diminui a velocidade. Quanto mais o usuário segura essas teclas, mais aumenta a velocidade até uma velocidade máxima e diminui a velocidade até a aeronave parar.

Logo após aprender os controles de movimentação da nave, o usuário deve testar o tiro da nave, que ocorre ao pressionar a tecla espaço. O usuário deve encontrar uma esfera na cena, onde ele deve atirar com a nave na esfera para fazer ela aumentar de tamanho até estourar. Depois, o usuário deve atirar em alvos mais difíceis de acertar, as vacas que se movimentam por curva de Bézier. Ao conseguir acertar as duas, sem colidir com elas, a simulação acaba e o usuário pode repetí-la ao pressionar a tecla ‘U’ ou sair da aplicação, pressionando a tecla esc.



Esfera antes de atirar nela e depois de atirar nela, quando está prestes a estourar

4. Compilação e execução

Para compilar e executar o programa, é necessário fazer os mesmos procedimentos dos laboratórios da disciplina. Para compilar, usa-se o CodeBlocks, basta carregar o projeto e selecionar a opção ‘Build’ na barra de tarefas. Ocorreu alguns problemas na transição de arquivos desse tipo de projeto com o CodeBlocks onde não é possível compilar devido ao um erro de referência em um dos arquivos (geralmente o textrendering.cpp ou tiny_obj_loader.cpp) durante o semestre e com este trabalho. Se este erro ocorrer, basta reeditar o arquivo que está dando problema, isso salvará o diretório do arquivo no projeto, e o erro não acontecerá mais. O executável está disponível em bin/Debug.