Simulador de voo

Leonardo Holtz de Oliveira e Mario Jose Kunz Filho

.inf
INSTITUTO
DE INFORMÁTICA
UFRGS

Lógica do programa

Um simulador de controle do voo, com a movimentação inspirada em simuladores de voo e nos jogos Star Fox, Grand Theft Auto e Battlefield.

Interação de tempo real baseado em tempo

No loop do jogo foi colocado:

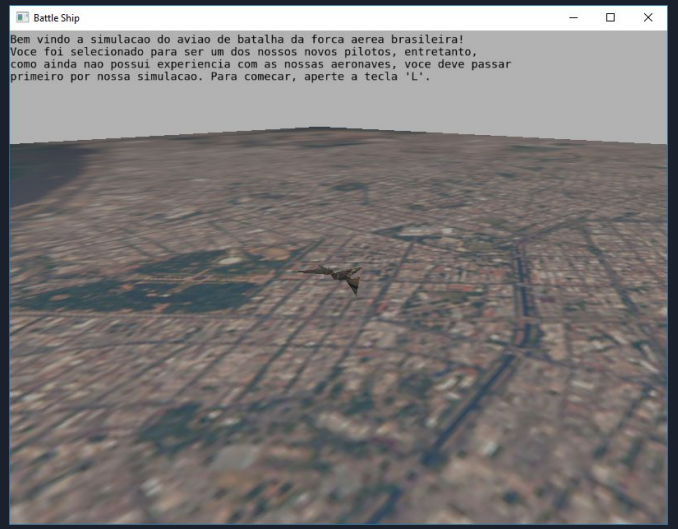
```
double tnow=glfwGetTime();

deltat = tnow- tprev;

tprev=tnow;
```

Assim possibilita movimentação em relação ao tempo:

```
rotation += 100*(float)deltat;
```



Transformações Geométricas

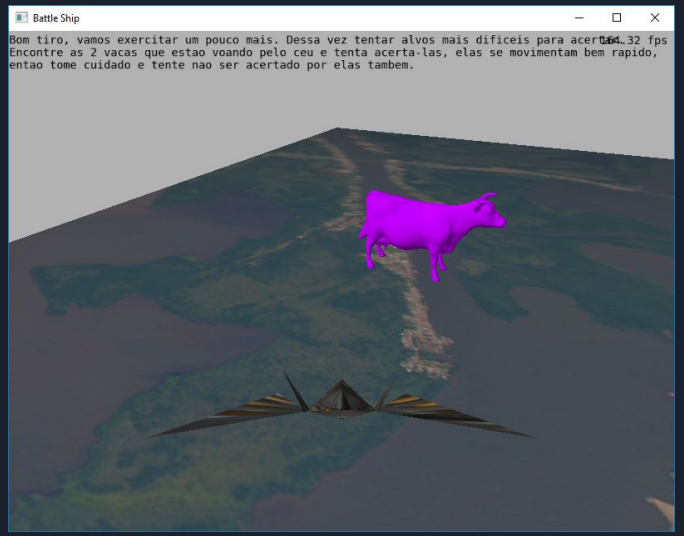
Com Mouse, temos a movimentação da nave para cima e para baixo e movimentação da câmera look_at.

Com o teclado temos a movimentação da nave para a direita e para a esquerda, que interferem também na rotação da nave.

Modelo Geométrico Complexo e Iluminação Lambert por vértice

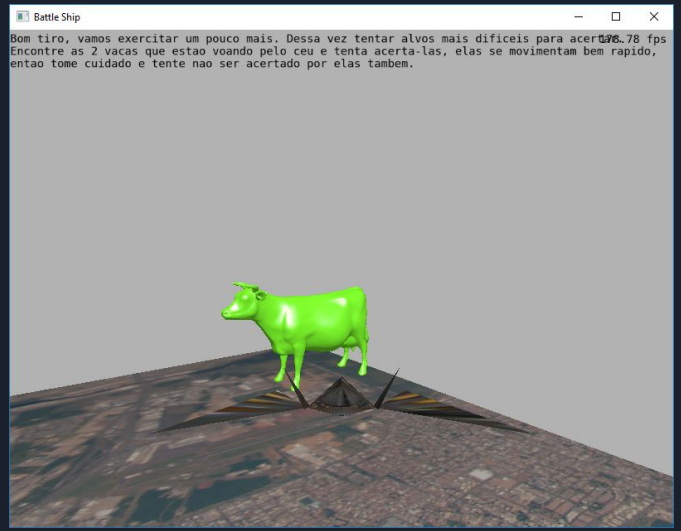
A vaca roxa se utiliza de um modelo geométrico complexo, além disso, ela utiliza o modelo de iluminação difusa e o Modelo de Gourad.

Veremos que existem duas instâncias do modelo complexo da vaca no próximo Slide.



Iluminação Phong por pixel

A vaca verde tem modelo de iluminação Blinn-Phong e cálculo da iluminação por pixel.



Curva Bézier

A curva é feita a partir de vértices de um triângulo equilátero, que formam uma curva Bézier quadrática circular

```
// Construção do triangulo a partir do apotema
glm::vec4 ponto1    = glm::vec4(apothem_x, apothem_y - (height/3), apothem_z, 1.0f);
glm::vec4 ponto2    = glm::vec4(apothem_x + (lado/2), apothem_y - (height/3), apothem_z, 1.0f);
glm::vec4 ponto3    = glm::vec4(apothem_x, apothem_y + (2 * height/3), apothem_z, 1.0f);
glm::vec4 ponto4    = glm::vec4(apothem_x - (lado/2), apothem_y - (height/3), apothem_z, 1.0f);
```

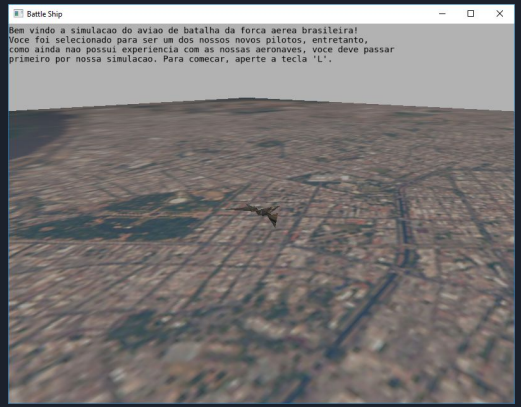
```
glm::vec4 C12 = ponto1 + valor_param_vacal*(ponto2 - ponto1);
glm::vec4 C23 = ponto2 + valor_param_vacal*(ponto3 - ponto2);
glm::vec4 C34 = ponto3 + valor_param_vacal*(ponto4 - ponto3);
glm::vec4 C41 = ponto4 + valor_param_vacal*(ponto1 - ponto4);
glm::vec4 C123 = C12 + valor_param_vacal*(C23 - C12);
glm::vec4 C234 = C23 + valor_param_vacal*(C34 - C23);
glm::vec4 C341 = C34 + valor_param_vacal*(C41 - C34);
glm::vec4 C1234 = C123 + valor_param_vacal*(C234 - C123);
glm::vec4 C2341 = C234 + valor_param_vacal*(C341 - C234);
Ct = C1234 + valor_param_vacal*(C2341 - C1234);
```

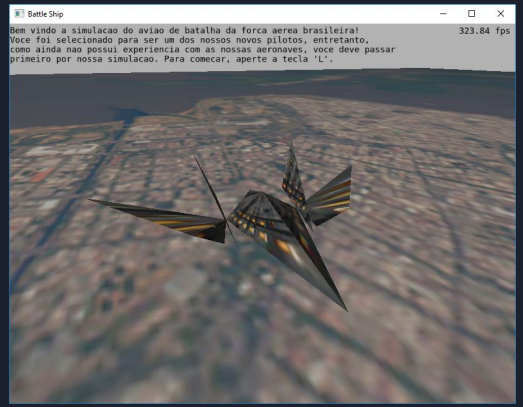
```
// Atualizamos o parametro da curva
if(valor_param_vacal < 1.0f)
    valor_param_vacal = valor_param_vacal + deslocamento * deltat;
else
    valor_param_vacal = 0.0f;
```

Câmera Livre e Camera Look at

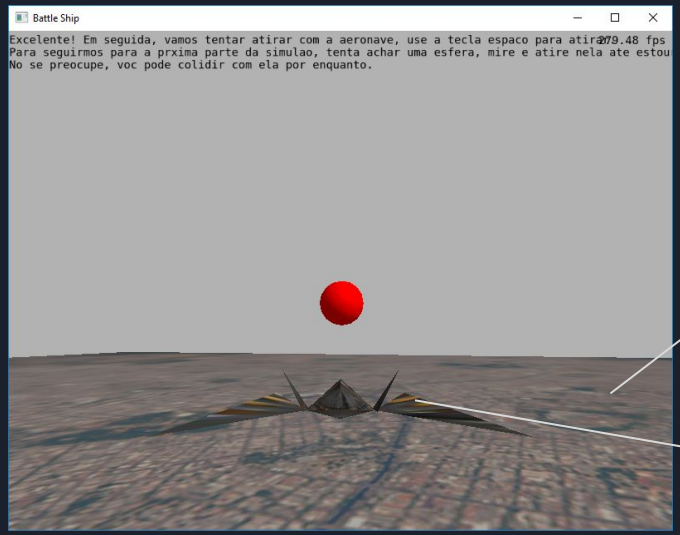
A movimentação da câmera livre é em conjunto com a nave e ela se movimenta para frente e faz rotação no próprio eixo.

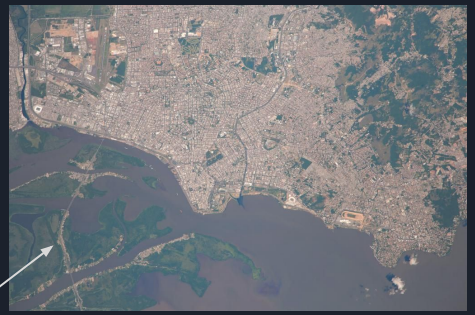
A câmara look_at é notória no início do programa, o usuário deve pressionar a tecla L para começar a simulação com a câmera livre.

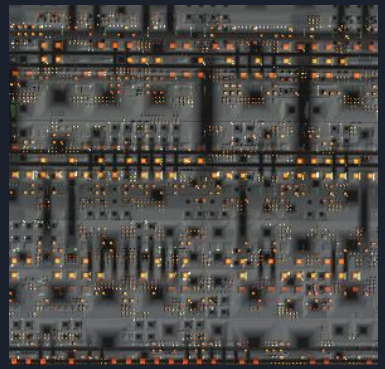




Mapeamento de textura: Chão e nave







3 tipos de Colisão

Colisão Cubo-Cubo: Vaca e Nave

Colisão Ponto-Esfera: Tiro da Nave e objetos 3-D

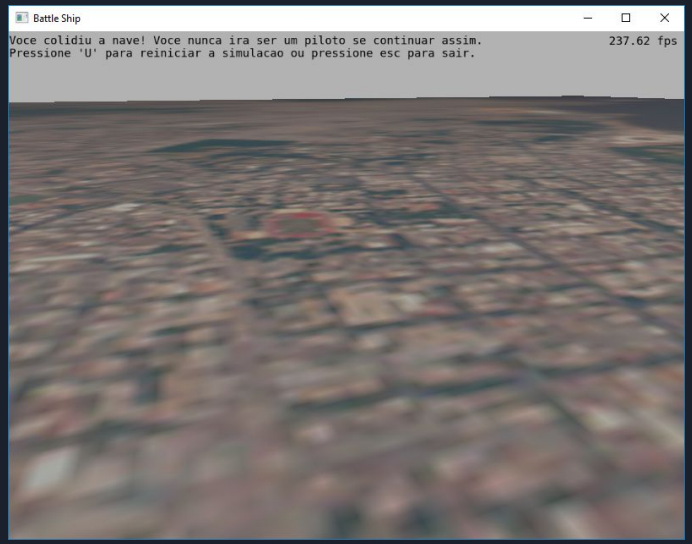
Colisão Cubo-Plano: Nave e Plano do chão

Colisão Cubo-Plano

Feita a partir do ponto mínimo do modelo.

Quando o ponto cruza o plano do chão, é considerado que a nave colidiu.

```
// Intersecção cubo-plano
bool isPlaneBox(glm::vec4 boxmin,glm::vec4 boxmax)
{
  return (boxmin.y <= -1.0f);
}
```



Colisão Cubo-Cubo

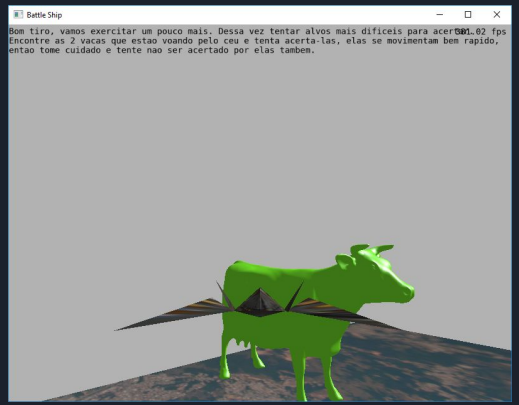
Feita a partir dos pontos mínimos e máximos dos modelos da nave e das vacas.

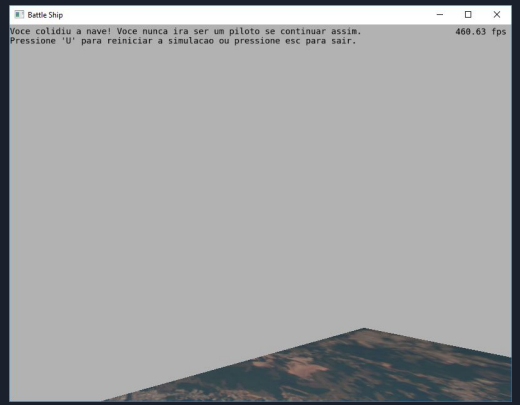
Quando um dos pontos da nave entra no intervalo dos pontos de uma das vacas, a nave colide.

```
// Intersecção cubo-cubo
bool boxintersect(glm::vec4 boxmin1,glm::vec4 boxmax1,glm::vec4 boxmin2,glm::vec4 boxmax2)
{
    return (((((boxmin1.x>=boxmin2.x) && (boxmin1.x<=boxmax2.x)) ||
              ((boxmax1.x>=boxmin2.x) && (boxmax1.x<=boxmax2.x))) ||
             (((boxmin1.x<=boxmin2.x) && (boxmin1.x>=boxmax2.x)) ||
              ((boxmax1.x<=boxmin2.x) && (boxmax1.x>=boxmax2.x)))) &&

            ((((boxmin1.y>=boxmin2.y) && (boxmin1.y<=boxmax2.y)) ||
              ((boxmax1.y>=boxmin2.y) && (boxmax1.y<=boxmax2.y))) ||
             (((boxmin1.y<=boxmin2.y) && (boxmin1.y>=boxmax2.y)) ||
              ((boxmax1.y<=boxmin2.y) && (boxmax1.y>=boxmax2.y)))) &&

            ((((boxmin1.z>=boxmin2.z) && (boxmin1.z<=boxmax2.z)) ||
              ((boxmax1.z>=boxmin2.z) && (boxmax1.z<=boxmax2.z))) ||
             (((boxmin1.z<=boxmin2.z) && (boxmin1.z>=boxmax2.z)) ||
              ((boxmax1.z<=boxmin2.z) && (boxmax1.z>=boxmax2.z)))));
}
```



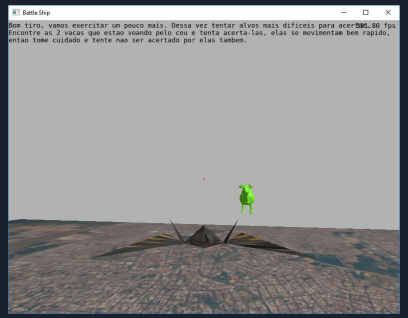


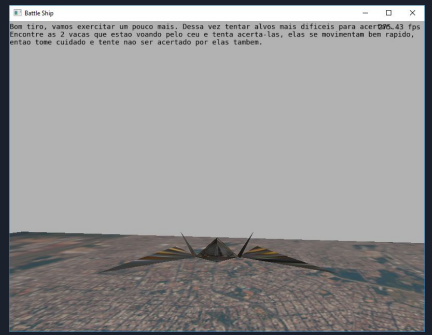
Colisão Ponto-Esfera

Feita a partir dos pontos mínimos e máximos dos modelos dos objetos, onde é calculado o ponto central de um objeto e o seu raio.

Quando a distância entre um ponto e o ponto central do objeto for menor ou igual ao raio do objeto, a colisão acontece

```
// Intersecção ponto-esfera
bool isPointCircle(glm::vec4 point,glm::vec4 circle,float raio)
{
  float distance =norm((point-circle));
  return distance<raio;
}
```





Perguntas?


MISSION
COMPLETE
quickmeme.com