Instituto Superior e Politécnico de Tecnologias e Ciências

Departamento de Engenharias e Tecnologias

COMPUTAÇÃO GRÁFICA

STARBLAST

Grupo nº 2

Andreia Vanessa Graça de Brito – 20180296

Helder Lucheses Gonçalves da Costa – 20181555

Miguel Gamboa Francisco Domingos – 20170757

INSTITUTO SUPERIOR POLITÉCNICO DE TECNOLOGIAS E CIÊNCIAS

DEPARTAMENTO DE ENGENHARIAS E TECNOLOGIAS

COMPUTAÇÃO GRÁFICA

Trabalho prático como requisito parcial que visa cumprir as exigências da disciplina para obtenção da aprovação em Computação Gráfica pelo Instituto Superior Politécnico de Tecnologias e Ciências (ISPTEC).

Docente: Lírio Ramalheira

LUANDA – TALATONA

2022

Introdução

O presente relatório aborda o processo de desenvolvimento do jogo StarBlast. Um jogo desenvolvido pelos integrantes como requisito para aprovação na cadeira de Computação Gráfica. Neste explicam-se as etapas pelas quais o jogo passou até chegar a versão final.

Teve-se como guia os enunciados disponibilizados ao longo do semestre, seguindo-se passo a passo as recomendações dadas pelo orientador, professor Lírio Ramalheira, e em 4 entregas chegou-se ao objectivo final.

O StarBlast é um jogo de combate entre naves totalmente desenvolvido em JavaScript fazendo uso da biblioteca Three.js que é uma biblioteca JavaScript/API cross-browser usada para criar e mostrar gráficos 3D animados em um navegador browser.

Objetivo geral

- Desenvolver um jogo de batalha entre naves aplicando os conceitos de câmera ortogonal, câmara perspectiva, iluminação, colisões e texturas.

Índice

1. Protótipo 2
2. Resumo 3
3. **1ª Entrega** 4
4. **2ª Entrega** 5
   - **4.1. Problemas** 6
   - **4.2. Dificuldades** 6
5. **3ª Entrega** 7
   - **5.1. Correcções das entregas anteriores** 7
   - **5.2. Dado em enunciado** 10
   - **5.3. Melhorias** 11
   - **5.4. Problemas** 11
   - **5.5. Dificuldades** 11
6. **4ª Entrega** 12
   - **6.1. Dificuldades** 12
7. Conclusões e propostas de trabalho futuros 13

Links 13

Referências Bibliográficas 13

1. Protótipo

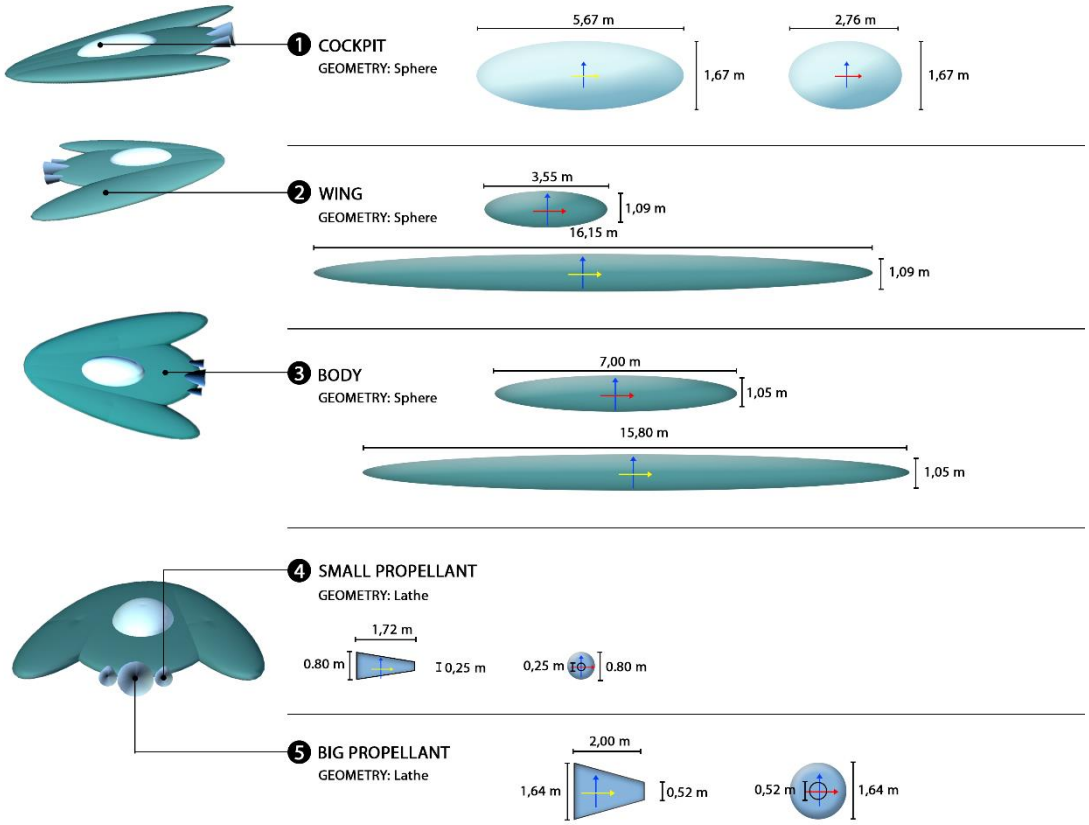


Figura 1 - Nave Heroína

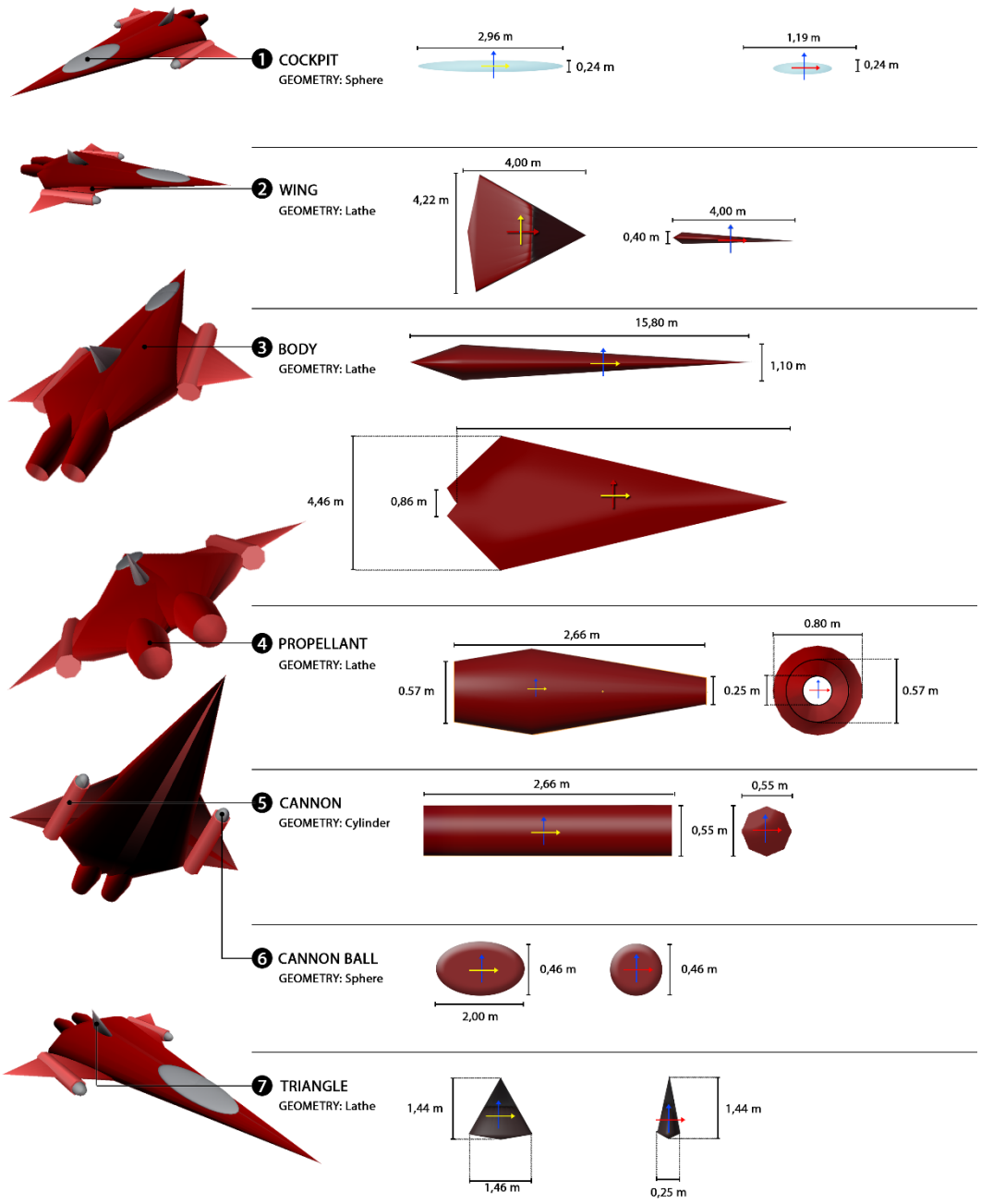


Figura 2 - Nave Inimiga

2. Resumo

O objectivo do trabalho foi a criação de um jogo de batalha entre uma nave heroína e naves inimigas. A nossa versão do jogo foi dividida em níveis começando do nível 1 até ao nível 18.

O desenvolvimento foi divido em entregas incrementais (4 no total) onde a cada entrega novas funcionalidades foram adicionadas até se obter a versão final que se verifica na 4ª entrega.

3. 1ª Entrega

Na primeira entrega montou-se uma cena simples com câmaras fixas. Para a cena usaram-se primeiramente 6 planos (esquerdo, direito, frente, trás, topo, baixo) para montar uma caixa. Para renderizar ambas faces dos planos se usou o THREE.DoubleSide. Também se adicionaram estrelas aos planos recorrendo ao uso de octaedros.

Para a confecção das naves usaram-se primitivas como esferas, tornos e cilindros (ver figuras 1 e 2).

Quanto às câmaras foram definidas:

- uma câmara fixa com vista de topo accionada pela tecla '1';
- uma câmara fixa com vista lateral accionada pela tecla '2';
- uma câmara fixa com vista frontal accionada pela tecla '3'.

Utilizou-se projecção perspectiva em todas as câmaras da primeira entrega. Este tipo de projecção simula a maneira como o olho humano vê e é a mais comumente usada para renderizar cenas em 3D.

Premindo a tecla '4' é possível ver todas as primitivas da cena em modelo de arame.

Finalmente também foi programado uma primeira instância do movimento das naves. A nave do herói move-se usando as setas do teclado a do vilão move-se de uma ponta à outra da cena de forma aleatória.

Achar um bom movimento aleatório para o vilão foi um desafio. Na primeira entrega, simplesmente se fez a velocidade no eixo X variar conforme a função cosseno que ia aumentando conforme um step.

Ao se movimentarem pelo eixo X, pôde-se notar que as naves se inclinavam. Pois bem, isso foi feito com o uso de percentagens, sendo $\frac{velocidade\ actual}{velocidade\ máxima}$ a fracção que determina ângulo. O ângulo vai de $^{-\pi}/_{8}$ a $^{\pi}/_{8}$ $(-22{,}5° \ a\ 22{,}5°)$.

Nesta entrega o foco foi o movimento do herói. Neste, além da inclinação, dá para reparar que há um movimento harmónico no eixo Y. Isto foi feito para parecer que a nave está *hovering* (a pairar).

Finalmente pode-se reparar que foi implementado o "atrito" para dar aquela sensação de física ao movimento.

3.1. Problemas

Um dos problemas desta entrega é que o movimento da nave inimiga não era aleatório, ela simplesmente se movimentava de um extremo ao outro da cena.

Um segundo problema detectado é que a nave do herói podia se mover para fora da câmara e nunca mais ser vista, perdendo-se nos confins do universo.

Havia também um *glitch* na inclinação da nave do herói, porque ao contrário da nave inimiga, a inclinação não era relativa à velocidade e era determinada por que tecla

estava apertada. Pressionando as setas direita e esquerda de forma ininterrupta a inclinação não era natural.

Ao movimentar o herói o jogo procurava saber para onde o herói está virado em relação à cena. Isso fazia para com que cada câmara as teclas de movimento fossem diferentes. Por exemplo, na câmara frontal para ir pra a frente utilizava-se a seta para cima, mas na lateral usava-se a seta para a direita.

3.2. Dificuldades

Para organizar melhor o código seria útil poder se separar o programa em módulos. Infelizmente o JavaScript não permite o uso de módulos a menos que seja usado um servidor. A solução usada para trabalhar localmente foi criar ficheiros e ir importando no HTML, criando uma sensação, ainda que falsa, de modularização.

Ao desenhar as naves, devido às restrições do trabalho foi impossível usar programas como o *Blender* para modelação. Houve várias naves desenhadas usando extrudes de esferas e cubos, mas momentos antes da entrega foi necessário arranjar uma solução diferente. Para modelação, portanto, usou-se o threejs.org/editor. Isto foi feito colocando as primitivas no editor e copiando manualmente (que dá um trabalhão), cada medida, cor, rotação e posição de cada primitiva. Primitivas como o torno deram um especial trabalho porque para deformação é utilizado um *array* de pontos.

4. 2ª Entrega

Na segunda entrega tiveram-se como objectivos explorar o conceito de câmara virtual, as diferenças entre câmara fixa e câmara móvel, a diferença entre a projecção ortogonal e a projecção perspectiva além da compreensão das técnicas básicas de animação e detecção de colisões.

Criaram-se 7 naves inimigas com o mesmo movimento sinusoidal, tendo como mudança que ao colidirem, as velocidades são trocadas.

À nave heroína foram adicionados limites ao movimento, para resolver o problema de a nave sair da vista da câmara e nunca mais voltar.

Adicionaram-se canhões à nave heroína, um debaixo de cada asa da nave e um no centro (embaixo também). Para a nave inimiga adicionaram-se 2 canhões, indo de acordo ao protótipo da primeira entrega. É possível seleccionar um canhão premindo as teclas: ‘Q’, ‘W’ e ‘E’ (esquerda, centro e direita respectivamente). O canhão seleccionado muda de cor para rosa-choque.

Implementaram-se os disparos usando esferas. Estas esferas giram sobre o seu próprio eixo. Apesar de girarem é impossível de se perceber já que as esferas são de cor sólida e são “esferas perfeitas”.

Os disparos são efectuados pressionando-se a barra de espaço. As balas seguem o caminho longitudinal do eixo Z quando disparadas.

Quanto às câmaras foram definidas:

- uma câmara fixa com vista de topo accionada pela tecla ‘1’ (ortogonal);
- uma câmara fixa com vista frontal accionada pela tecla ‘2’ (perspectiva);

- uma câmara que segue a bala accionada pela tecla ‘3’ (perspectiva);
- uma câmara que gira em torno da cena, accionada pela tecla ‘5’ (perspectiva).

As colisões foram feitas todas pelo método AABB (Axis Aligned Bounding Boxes). As colisões desta fase foram relativamente “preguiçosas”. A bala ao ricochetear simplesmente muda o sinal da velocidade no eixo Z e das naves ao coliderem são trocados o módulo e o sentido da velocidade, já que as naves inimigas apenas se movem no eixo Z.

Após 3 colisões de uma bala com um inimigo o inimigo desaparece.

Nesta entrega foi resolvido o problema de as naves se moverem de acordo à câmara. O movimento agora é absoluto (independe da câmara seleccionada).

4.1. Problemas

As colisões são demasiado preguiçosas. As colisões bola-bola não foram feitas nesta entrega e a troca de sentido e módulo não representa corretamente uma colisão, isso faz com que as naves se sobreponham em certos casos, e noutros simplesmente copiem as velocidades umas das outras, fazendo com que grudem em vez de se moverem livremente. Isto também se deve ao nível de aleatoriedade do movimento e a liberdade (sendo apenas num dos eixos).

Apesar da cena ter sim paredes, essas paredes são o espaço sideral, sendo que a altura e a largura são enormes, tornando fútil a colisão com as paredes, em vez disso foram usados limites mesmo (limites estes que estão relativamente distantes das paredes). Outro problema vindo disto é que o limite de movimento das naves inimigas (em X) é maior que a nave dos heróis.

Devido à textura das bolas ser apenas uma cor sólida é impossível visualizar a rotação no seu próprio eixo.

Devido a uma câmara ser perspectiva e outra ortogonal (uma câmara frontal e outra de topo respectivamente), é impossível estabelecer os limites de forma consistente entre as duas câmaras, então levou-se em consideração aquilo que foi tomada como câmara principal do jogo (mesmo para futuras entregas): a câmara frontal.

Os canhões da nave do herói foram modelados com *bugs*, devido a um erro que corrompeu o ficheiro criado em threejs.org/editor, tornando os números mostrados numa representação imprecisa do desenho modelado.

A câmara que gira em torno da cena tem o problema de cada vez que se aperta na tecla girar mais rápido. Além disso ela pode ser ativada na camara de topo, algo fora dos planos da equipa de desenvolvimento.

4.2. Dificuldades

Esta entrega foi tribulada e a pior de todas feitas pelo grupo (de longe), por um desentendimento quanto à data de entrega do trabalho. Apesar de não estar relacionado ao trabalho de Computação Gráfica, parte do grupo estava a se preparar para uma competição internacional de programação e julgou-se haver mais uma semana para a

entrega. No entanto no dia da entrega foi esclarecido que era para ser entregue no mesmo dia. Apesar do professor ter sido generoso o suficiente para dar mais um dia, simplesmente não foi o suficiente para manter o padrão de trabalho, tornando esta entrega um pouco mais "*sloppy*".

Realizar colisões não é tarefa fácil, e pelas razões mencionadas no parágrafo anterior, não foi possível dedicar muito tempo a fazê-las, sem contar com os inúmeros *bugs* encontrados pelo caminho.

5. 3ª Entrega

Nesta entrega, basicamente o trabalho foi feito desde o início, já que o código estava a se tornar numa autêntica salada russa, mas mantendo a lógica das entregas anteriores. O que mudou é que se passaram a usar os conceitos de orientação a objectos de forma mais estrita. A entrega anterior (2ª) foi refeita devido aos inúmeros problemas.

5.1. Correcções das entregas anteriores

A nave do herói foi corrigida com os canhões certos, além disso foram criadas mais 2 naves inimigas diferentes com propriedades diferentes.

O movimento aleatório das naves inimigas foi drasticamente melhorado. As naves já não seguem simplesmente um movimento harmónico pelo eixo X. Existe uma função moveRandomly(), que após um número determinado de *frames* computa um novo destino para a nave. Ou seja, as naves inimigas têm sempre como objectivo chegar a um ponto aleatório do mapa, podendo assim usar a mesma física de movimento que o herói (aceleração, atrito, travagem, hovering, etc.).

O *hovering* teve de ser aumentado já que as naves, devido aos canhões cresceram, no eixo Y.

A inclinação da nave do herói agora também é relativa à $\frac{velocidade\ actual}{velocidade\ máxima}$. Foi adicionada também uma inclinação em Z para quando a nave se move para a frente. Não existe inclinação ao se mover para trás porque os desenvolvedores não simpatizaram com o resultado.

Infelizmente foi um pouco sacrificada a lógica de "espaço sideral", porque além das paredes do espaço foram criadas caixas para as paredes delimitadoras do movimento. Dando um toque de personalização as paredes são transparentes e THREE.DoubleSide. Foi então criado um delimitador de movimento para o herói e outro para os inimigos.

A cena foi melhorada, agora as estrelas (octaedros) só ocupam 60% do raio da cena, ou seja, elas são aleatoriamente posicionadas desde o extremo até 60% do tamanho da aresta da caixa que representa o espaço sideral, tornando o centro mais realista, já que agora é impossível ter uma estrela mesmo no centro da cena. As naves seriam dilaceradas!

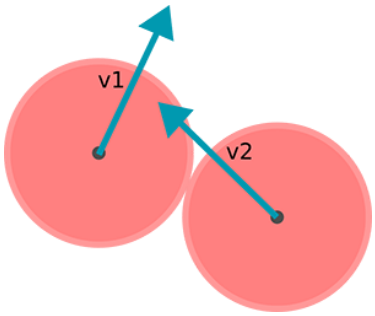
Os canhões agora quando seleccionados em vez de rosa-choque (toque feminino) tornam-se brancos, melhorando a estética e harmonia de cores. Implementaram-se os disparos de duas formas, o canhão central dispara balas redondas e os laterais usam mísseis. As bolas valem uma vida e os mísseis duas. Os inimigos também usam bolas e mísseis dependendo do canhão a ser usado por eles.

Quanto às colisões, foram completamente reestruturadas. Todas as colisões foram calculadas usando AABB, com excepção das colisões entre as bolas que foram calculadas usando esferas envolventes, cujo raio é calculado pelo tamanho da aresta da bounding box (todas as arestas têm o mesmo tamanho já que a bounding box de uma esfera é um cubo). As bolas desaparecem quando saem do espaço sideral, limitado pelos planos que anteriormente eram as paredes da cena.

O algoritmo usado para determinar os novos vectores-velocidade foi o mesmo tanto para as bounding boxes quanto para as esferas envolventes. Após detectar a colisão usando AABB ou esferas envolventes, a nova velocidade foi calculada seguindo os seguintes passos:

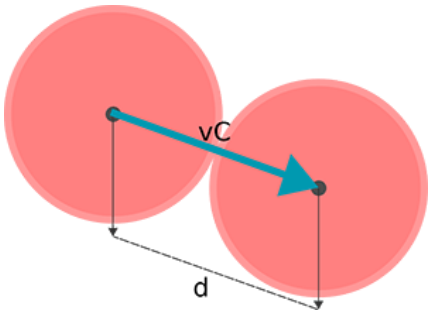
Nota: os exemplos estão em duas dimensões por simplicidade, para funcionar em três dimensões é só adicionar a componente Z em cada um dos passos.

1. Achar o vector de colisão.



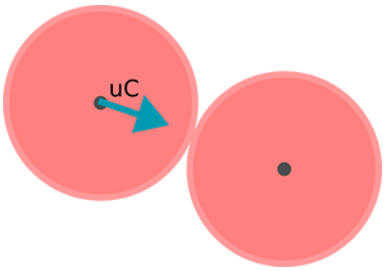
$$\vec{v}_{colisão} = (objecto_{2_x} - objecto_{1_x}, objecto_{2_y} - objecto_{1_y})$$

2. Achar a distância entre os vectores de colisão.



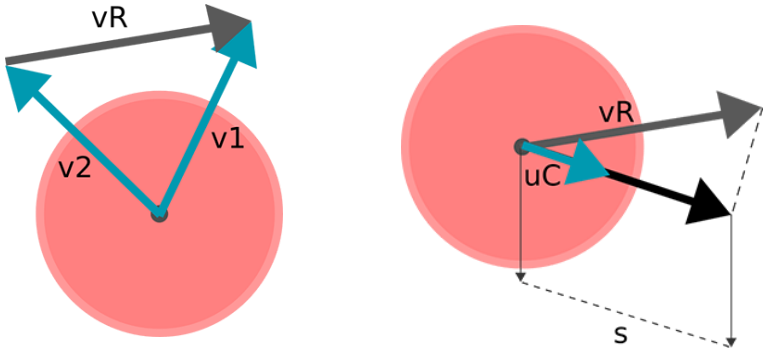
$$d = \sqrt{(objecto_{2_x} - objecto_{1_x})^2 - (objecto_{2_y} - objecto_{1_y})^2}$$

3. Normalizar o vector de velocidade da colisão, o que nos permite ter apenas a direção da colisão.



$$\hat{v}_{colisão} = (\frac{v_{colisão_x}}{d}, \frac{v_{colisão_y}}{d})$$

4. A nova velocidade é o produto escalar entre a velocidade relativa e o vector da colisão normalizado.



$$\vec{v}_{relativa} = (v_{1_x} - v_{2_x}, v_{1_y} - v_{2_y})$$

$$|v| = v_{relativa_x} \cdot \frac{v_{colisão_x}}{d} + v_{relativa_y} \cdot \frac{v_{colisão_y}}{d}$$

Só para reforçar, para replicar os cálculos no espaço é só adicionar a componente Z.

Além do cálculo natural da colisão foi usado o impulso para os corpos mais pesados não serem afectados pelos menores, seguindo a fórmula do impulso em física:

$v = \frac{2 \times velocidade \times massa_2}{massa_1 + massa_2}$, para um objecto com massa $massa_1$

O movimento das balas também foi melhorado, elas são disparadas com uma v_y positiva, mas são afectadas pela aceleração gravítica. Elas são removidas da cena quando caem para o abismo (abaixo do nosso "universo observável").

Um detalhe adicionado ao movimento dos mísseis é que após colidirem giram em torno do eixo Y, simulando um movimento em espiral.

Após o número necessário de colisões (os inimigos têm vidas variáveis dependendo do tipo de nave) o inimigo se desintegra. Isso é feito atribuindo velocidades aleatórias não nulas às primitivas. Depois de um certo número de frames as primitivas são removidas da cena.

5.2. Dado em enunciado

Nesta entrega tiveram-se como objectivos principais compreender as noções básicas e os conceitos de material, fonte de luz direcional e fonte de luz spotlight, e compreender a modelação geométrica por instanciação de primitivas, criação de malhas de polígonos e uso de materiais.

Nas entregas passadas todos os objectos da cena foram renderizados usando a MeshBasicMaterial nesta entrega definidos um vetor de 3 posições com 3 tipos de material para renderizar as naves inimigas e as paredes, sendo estes materiais MeshBasicMaterial, MeshLambertMaterial e MeshPhongMaterial. Já para a nave heroína definimos um vector de materiais de 2 posições sendo estes MeshBasicMaterial e MeshLambertMaterial.

As principais diferenças entre estes tipos de material são as seguintes:

- o MeshBasicMaterial não é afectado pela iluminação;
- o meshLambertMaterial é usado para superfícies não brilhantes e sem realces especulares. Este material não é baseado fisicamente para calcular o factor de reflexão, por isso simula bem superfícies como madeira não tratada ou pedras, mas não pode simular superfícies brilhantes com realces especulares como madeira envernizada. O sombreamento é calculado usando um modelo de sombreamento Goraud que calcula o sombreamento por vértices e interpola intensidades nos vértices.
- o MeshPhongMaterial é usado em superfícies brilhantes com realces especulares. Este material usa um modelo de Blinn-Phong. Ao contrário do modelo Lambertiano, este pode simular superfícies brilhantes como madeira envernizada. O sombreamento é calculado por pixel que fornece resultados mais precisos do que o modelo de Goraud ao custo de algum desempenho.

A seguir criou-se a iluminação global da cena fazendo uso de uma luz direcional que é uma luz que emitida numa direcção específica. Essa luz se comporta como se estivesse infinitamente distante e os raios produzidos por ela fossem paralelos. Esta luz é acionada pela tecla ‘Q’.

Para alternar o tipo de sombreamento de Goraud para Phong e vice-versa simplesmente mudamos o tipo de material dos objectos de acordo ao sombreamento pretendido (MeshLambertMaterial para o sombreamento de Goraud e MeshPhongMaterial para o sombreamento de Phong). Está alteração é feita ao acionar a tecla ‘E’.

O outro tipo de luzes utilizadas foram as spotlights ou luzes de holofotes que são luzes que são emitidas de um único ponto em uma direcção ao longo de um cone que aumenta de tamanho à medida que se afasta da luz.

As teclas dos holofotes são:

- ‘Y’ – para o superior esquerdo;
- ‘U’ – para o superior direito;

- ‘H’ – para o inferior esquerdo;
- ‘J’ – para o inferior direito.

Nesta entrega também se mudou a câmara de topo (anteriormente ortogonal) para uma câmara perspectiva e se adicionou uma câmara ortogonal centrada na nave heroína acionada pela tecla ‘6’.

Mudou-se a câmara de topo para uma câmara ortogonal como solicitado no enunciado e adicionou-se uma câmara móvel perspectiva para seguir a última bala disparada pela nave heroína e uma câmara perspectiva móvel que percorre a cena toda em 360 graus.

5.3. Melhorias

Para manter os pontos de avaliação foi criado o “Teacher’s mode”, tendo apenas o que foi pedido em enunciado. O leitor pode se perguntar, “mas porquê?”, pois bem, nesta entrega o grupo tentou adicionar mais lógica ao jogo.

Primeiro foi desenhado um menu com opções para instruções, jogar, entre outras.

No modo de jogo normal foram introduzidos os níveis. Cada nível tem uma dificuldade diferente, sendo que algumas naves são mais difíceis de abater que outras. O jogador recebe pontos ao acertar num inimigo e quanto mais tempo restante tem, mais pontos recebe. Por cada nível passado há um bónus de tempo, se o tempo ou as vidas acabam o jogador perde. Até ao momento foram implementados 18 níveis, sendo que o boss não foi completamente desenhado nesta entrega.

5.4. Problemas

Ao reiniciar o jogo os elementos do jogo passado continuam carregados na cena, fazendo com que, após um certo número de reinícios o navegador *crash*. Este problema foi resolvido em entregas posteriores.

Por algum motivo as bounding boxes das paredes não funcionavam correctamente, então foi programada uma lógica análoga manualmente.

A rotação espiral dos mísseis ao cair não é perfeita, funciona, mas visualmente não é tão agradável quanto devia ser.

5.5. Dificuldades

Programar *spotlights* foi um pesadelo, foi uma eternidade até descobrir que para apontar a *spotlight* para um dado lugar requer apenas o target e updateWorldMatrix().

Apesar de ser possível colocar mais de um material num objecto em simultâneo usando um vector, o enunciado pede para mostrar os vários tipos de sombreamento. A forma lógica de fazê-lo é usar um material de cada vez.

Ao implementar o sombreamento houve uma confusãozinha com mapeamento de sombras. O mapeamento de sombras ainda se encontra nos confins do código, mas está desactivado já que requer um pouco mais de processamento e não melhora de forma considerável a cena em termos estéticos.

6. 4ª Entrega

Os objectivos principais dessa entrega foram aprofundar os conhecimentos de iluminação e compreender os princípios básicos da aplicação de texturas e mensagens.

Para a aplicação das texturas de imagens utilizamos duas primitivas do tipo BoxGeometry colocadas nas paredes. De seguida carregamos a textura de imagem usando a classe THREE.TextureLoader(). Esta classe usa a classe ImageLoader internamente para carregar os ficheiros, no caso, as imagens. Em seguida criamos um material MeshBasucMaterial e aplicamos essa textura a propriedade map do material. Assim a BoxGeometry é renderizada com a imagem. Este processo foi feito para o quadro 'Monalisa' e o quadro 'Grito'.

Para o chão, primeiramente criamos o modelo usando uma BoxGeometry. Em seguida escolhemos a textura e atribuímos às propriedades wrapS e wrapT o modo THREE.RepeatWrapping, dessa forma a textura foi repetida na vertical e na horizontal cobrindo toda BoxGeometry. Usamos a propriedade bumpMap para criar um mapa de relevo onde valores de preto e branco mapeiam a profundidade percebida em relação às luzes e a propriedade bumpScale para determinar quanto o mapa de relevo afeta o material, o valor atribuído foi 5.

Para a concepção das luzes usaram-se 3 tipos de primitivas, BoxGeometry para o corpo renderizado na cor 0x111111, CylinderGeometry para os sticks renderizados na cor 0x333333 e SphereGeometry para o bulbo renderizado na cor 0xeeeeee. Em seguida criaram-se luzes pontuais que são luzes que emitem a luz de um ponto para todas direcções. Posicionou-se as luzes exatamente nas posições em que se encontram os bulbos. Para apagar/acender as luzes pontuais accionam-se as teclas 'D(d)' para a parede esquerda e 'P(p)' para a parede direita.

Quanto a mensagem de início, como mencionado anteriormente, o jogo possui um menu inicial com uma mensagem com o nome do jogo e uma imagem de fundo ilustrativa do jogo. Neste menu o utilizador pode escolher o modo de jogo.

Durante o jogo, o utilizador tem a opção de pausar o jogo accionando a tecla 'S(s)' e fazer reset accionando a tecla 'R(r)'.

A lógica de jogo foi adicionada na terceira entrega, onde implementamos os níveis. A cada nível o grau de dificuldade aumenta, pois, o número de naves aumenta aparecendo até mesmo naves mais difíceis de abater, o tempo de jogo vai diminuindo bem como as vidas do jogador caso a nave seja atingida.

6.1. Dificuldades

A grande dificuldade neste nível foi na escolha das texturas. Foi muito difícil encontrar texturas agradáveis visivelmente que condissessem com a paleta de cores usada no jogo. Outra dificuldade foi na modelação das luzes de cartaz, de longe a tarefa mais odiada pelos membros do grupo.

7. Conclusões e propostas de trabalho futuros

Com a elaboração deste trabalho ganhou-se uma melhor noção de como é feita a geração de imagens no computador, a forma de representação dos dados e informação. Conceitos como pixel, cor, frame, animação, renderização, recorte, iluminação, texturas, sombreamento, colisões, materiais, projecção, câmeras, etc. foram aprimorados pois foram os conceitos que serviram de base para a elaboração do jogo, percebeu-se como cada pixel representa uma cor na cena, a cada mudança na cena gera-se um novo frame e a sucessão rápida dos frames da ilusão da animação, percebeu-se também como as luzes podem ou não afectar certos materiais, os tipos de luzes, o sombreamento que provocam, a diferença visível da utilização de uma câmera ortogonal e uma perspectiva, como a aplicação das texturas dá uma aparência mais agradável e realista dos objectos na cena, entre outros.

Pôde-se perceber, da unidade mais pequena da cena (pixel) à cena completa, como cada conceito supracitado influenciou no resultado que se obeteve no final.

A criação de um jogo está longe de ser uma tarefa fácil de se concretizar, mas o facto de ter sido um trabalho em equipa tornou tudo mais leve porque cada um pôde contribuir da sua maneira para atingir o objectivo traçado.

Pra trabalhos futuros propõem-se que se explore mais as texturas, a aplicação de texturas ao trabalho ajuda a tornar a cena mais real e mais atrativa visualmente podendo simular vários ambientes e objectos do mundo real. Também se propõe adicionar mais componentes à mecânica do jogo para retirar alguma da monotonia. Muitos dos aspectos que se encontram em desenvolvimento podem ser acabados em versões futuras como: colisão com o chão, mapeamento de sombras, fogo nas balas e muito mais, só o destino sabe no que este jogo se tornará.

Links

O jogo pode ser acedido pelos seguintes links:

- https://starblast.lucheses.com/
- https://starblast.indiouz.com/
- https://starblast.andreiabrito.xyz/

O código encontra-se disponível num repositório no github que pode ser acedido pelo link:

- https://github.com/hlucheses/starblast

Referências Bibliográficas

https://threejs.org/