**Hearthstone – Duel Status Recognition**

André Cardoso, 65069, [marquescardoso@ua.pt](mailto:marquescardoso@ua.pt)

Jorge Faustino, 64441, [jorgebalseiro@ua.pt](mailto:jorgebalseiro@ua.pt)

***Resumo* – Este artigo apresenta uma possível solução para encontrar o vitorioso e um derrotado num jogo de cartas de coleção online. Começa por explicar o jogo em questão e a necessidade de haver um método onde terceiros possam saber qual foi o jogador que ganhou e o que perdeu assim como alguns detalhes relevantes durante o duelo de cartas.**

**Por último é abordada a arquitectura do projecto relativamente à interface e implementação WebGL e as várias técnicas do mesmo utilizadas no projecto.**

I. Introdução

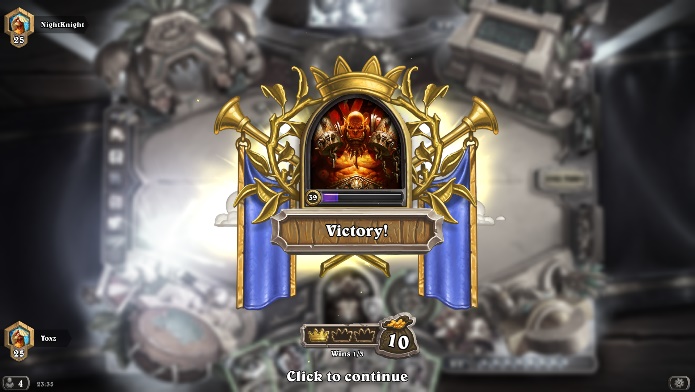
No âmbito da Unidade Curricular Computação Visual do 4º ano do curso Mestrado Integrado em Engenharia de Computadores e Telemática escolhemos a realização de um projeto usando as bibliotecas de OpenCV[1], o editor Visual Studio 2017[x] utilizando a linguagem de programação c++[].

O contexto do nosso trabalho está inserido um jogo online, Hearthstone [], que conta com mais de 70 milhões de jogadores até meio de 2017 [https://www.pcgamesn.com/hearthstone/hearthstone-player-count]. No jogo existe uma coleção de 2407 cartas, entre elas, cartas que representam feitiços e monstros com um determinado poder de ataque e defesa que o jogador vai acumulando. Existem também e 8 classes, isto é, Mágico, Guerreiro, Caçador e outros que o jogador escolhe como a sua identidade durante o duelo.[ <https://hearthstone.gamepedia.com/Card#Collecting_cards>]. Os procedimentos para executar um duelo são simples:

1. Cada jogador escolhe uma classe;
2. Cada jogador escolhe um baralho previamente feito com as cartas que possuiu na sua coleção;
3. É feito um duelo até a classe de um jogador ficar sem vida.

Mais pormenores são desnecessários para o contexto deste trabalho. Agora problema encontra-se em saber com um grau de certeza elevado quem foi o vencedor do duelo. O jogo ocorre numa plataforma chegada desenvolvida pela Blizzard [] onde não existe qualquer tipo de API para sabemos quem ganhou. E como os jogo ocorrem online, cada jogadores em espaços físicos completamente distintos, não é viável um deslocação física ao local para apurar o vencedor.

Uma possível solução é pedir uma capture de ecrã no final do duelo de cada jogador, serem submetidos num serviço e assim calcular com auxilio as ferramentas de analise de imagem do OpenCV quem é o vencedor, o derrotado e que classes cada jogador usou no seu duelo. A figuras 1 e 2 abaixo representação dois exemplos de uma possível captura de tela feita no final do duelo.



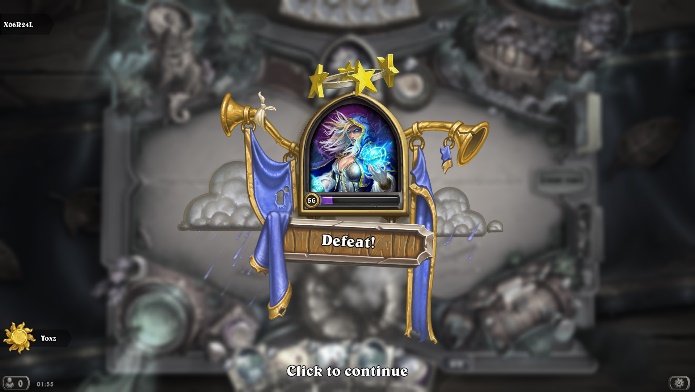


Figura 1 representa a classe Guerreiro que venceu o duelo e a figura 2 representa a classe Magico que perdeu o duelo.

II. Implementação

Nesta secção vamos apresentar todos os fractais implementados e podem ser visualizados no conteúdo dentro do ficheiro do tipo Winrar[7] que é acompanhado com este artigo.

Como esta solução usa extensivamente o GPU devido á recursividade, para os cálculos da iluminação das figuras, decidiu-se usar os fragment shader. Pois utiliza iluminação para cada pixel em vez de ser por vértices. Para tal foi criada uma matriz nova que contém a transposição da matriz inversa da matriz.

Durante toda a implementação foi usada os recursos disponíveis no site da unidade curricular dedicados às aulas práticas.[8]

As próximas subsecções contêm os vários modelos explorados: Sierpinki Gasket[9], Koch Snowflake[10], Menger Sponge[11], Mosely Snowflake[12]. Assim como um fractal desenvolvido por nós.

*A. Sierpinski Gasket*

Este fractal é baseado na subdivisão recursiva feita por tetraedros, como demonstra a Figura 2, que contém o estado original, com uma e com duas incrementações sucessivamente.

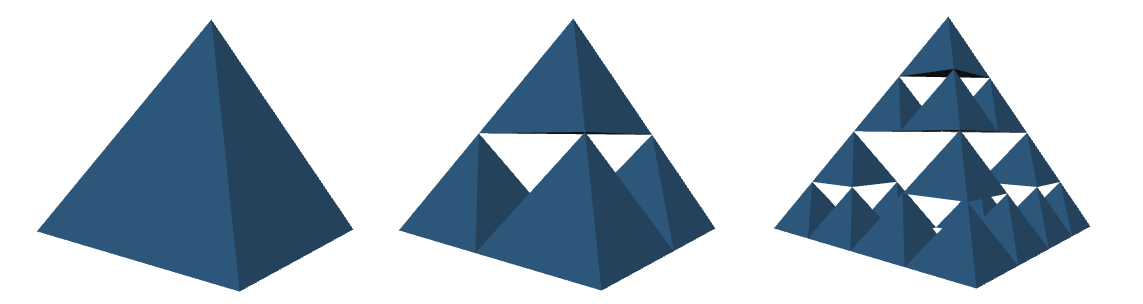


Figure 2- Representação de Sierpinski Gasket

Para obter o resultado pretendido são selecionados os vértices originais da figura, e é calculado um ponto intermédio cada par. Como é apresentado na Figura 3.

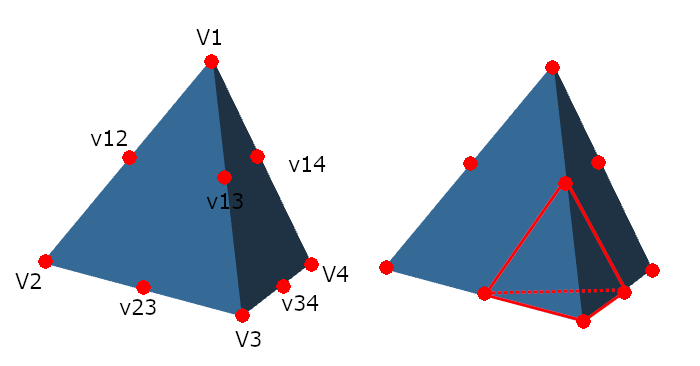


Figure 3- Representação do método de cálculo.

Por fim a nova lista de vértices com os pontos intermédios é introduzida, formando assim diferentes arestas que dão forma a quatro tetraedros no lugar de um.

*B. Koch Snowflake*

Com a mesma estrutura inicial do exemplo anterior, é possível criar um outro tipo de fractal. consiste em acrescentar um novo tetraedro mais pequeno, perpendicularmente, cada uma das faces como mostra a Figura 4.



Figure 4 - Representação de Kock Snowflake.

Na Figura 5, para cada face, são selecionados os vértices e é calculado o ponto intermédio entre cada par.

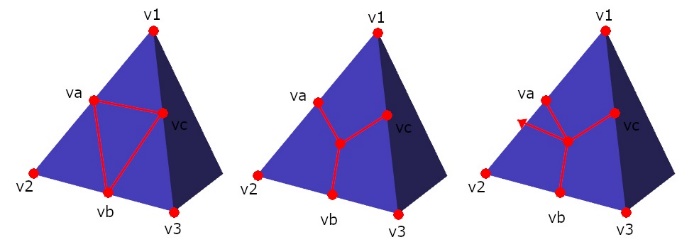


Figure 5 - Representação do método de cálculo.

É calculado o ponto central entre esses 3 novos vértices, e nesse ponto central é usado o cálculo do vector normal através dos vértices originais (v1,v2 e v3), que como estão no sentido anti-horário vão gerar um vector normal a “apontar para fora”. As coordenadas da extremidade desse vector irão formar o 4º vértice do novo tetraedro.

*C. Menger Sponge*

A *sponge* é formada através da divisão recursiva de um cubo em 27 cubos, removendo os cubos no centro de cada face, e no seu interior. Figura 6, representa o estado do fractal nas várias incrementações.

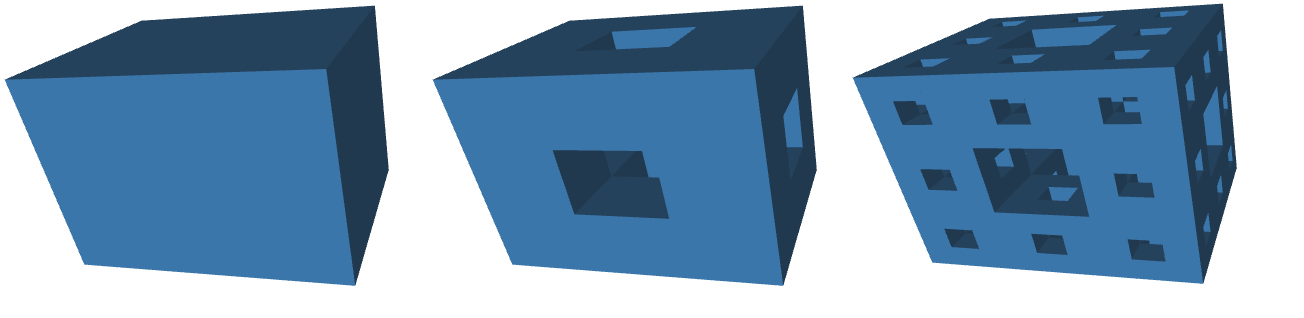


Figure 6 - Representação de Menger Sponge.

Para tal são selecionados os seus 8 vértices, é calculada a distância entre 2 vértices da mesma face, por exemplo o v3 e o v4, e é usado o terço dessa distância para calcular a posição dos novos vérticies.

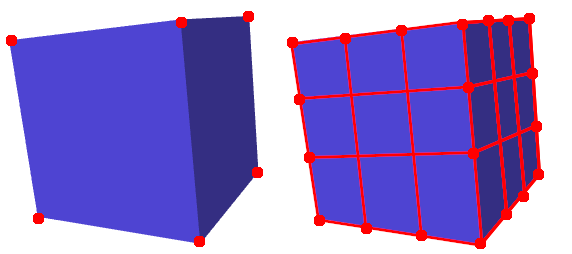


Figure 7 - Representação do método de cálculo.

*D. Mosely Snowflake*

Neste fractal, divide-se um cubo em 27 cubos mais pequenos e remove-se os cubos dos cantos. Figura 8, representa o estado do fractal nas várias incrementações.

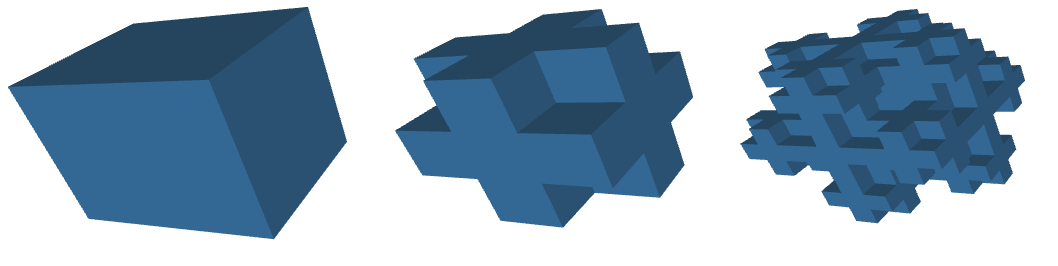


Figure 8 - Representação de Mosely Snowflake.

São selecionados os vértices originais, é calculado o ponto que fica a um terço da distância até a todos os outros vértices, como demostra a Figura 9.

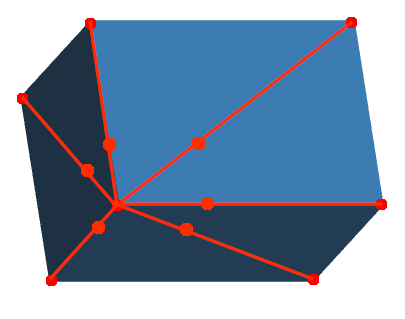
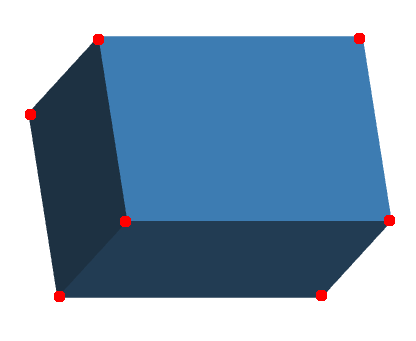


Figure 9 - Representação do método de cálculo.

Depois é calculado o ponto que fica a dois terços de distância até a todos os outros vértices.

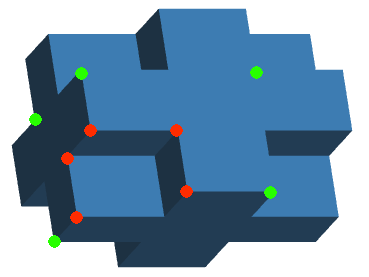
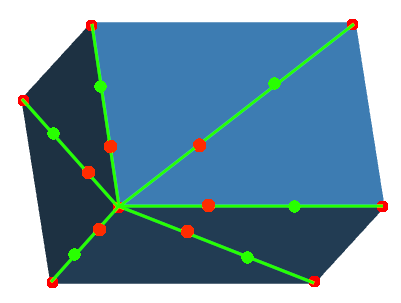


Figure 10 - Continuação da representação do método de cálculo.

Os vértices originais são descartados e a renderização é feita com os novos vértices.

*E. Custom Design*

Foi um fractal que inventámos, baseado no *Menger Sponge*,

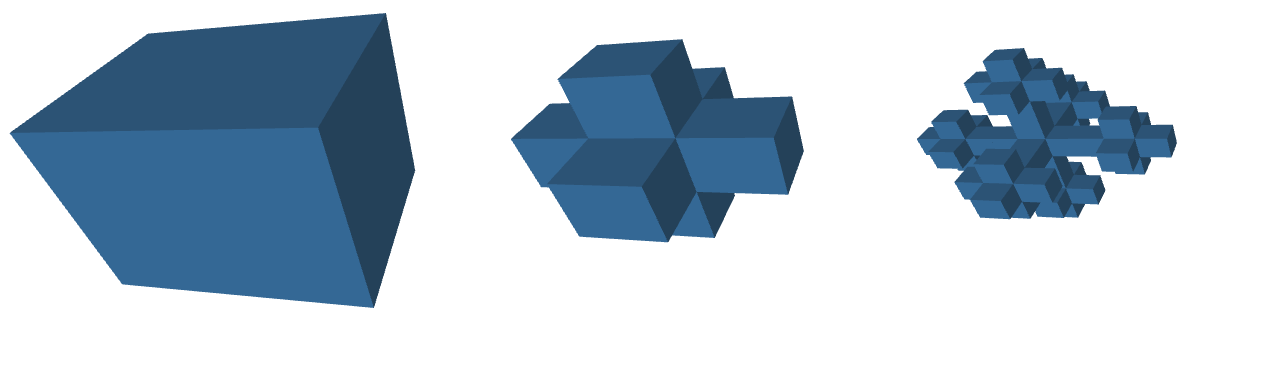


Figure 11 - Representação de Custom Design.

O seu cálculo é também semelhante ao do fractal *Menger Sponge*, o cubo original é dividido em 27, e são escolhidos os cubos centrais.

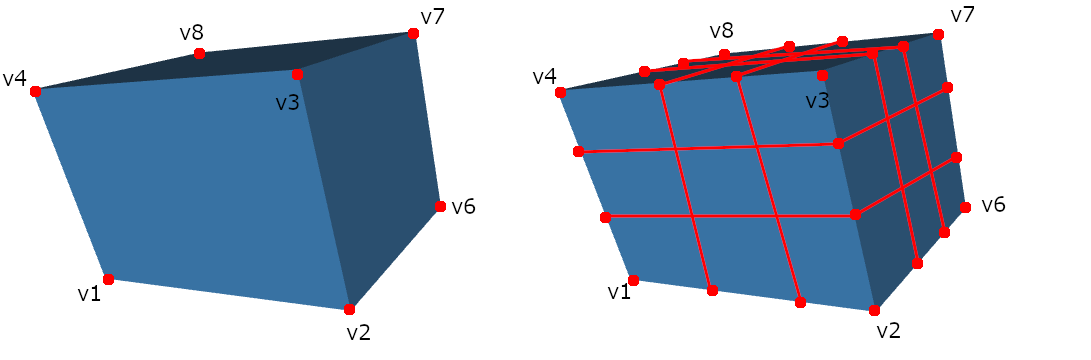


Figure 12 - Representação do método de cálculo.

III. Resultado Final

O nosso projecto baseia-se em tecnologias web já mencionadas, como tal para o executar apenas é necessário um browser. Para obter uma melhor experiência de utilização foi usado o Bootstrap para customização de estilos.

A Figure 13 apresenta interface gráfica é composta no lado direito pelo canvas que renderiza o Fractal escolhido com o devido número de incrementações. No lado esquerdo é possível controlar a interação com esse mesmo Fractal escolhido. É possível controlar a rotação e a respectiva velocidade de todos os eixos. Abaixo temos uma tabela de controlo onde é possível escolher o fractal, o tipo de projeção e o modo de renderização. Assim como um botão para voltar ao estado inicial.

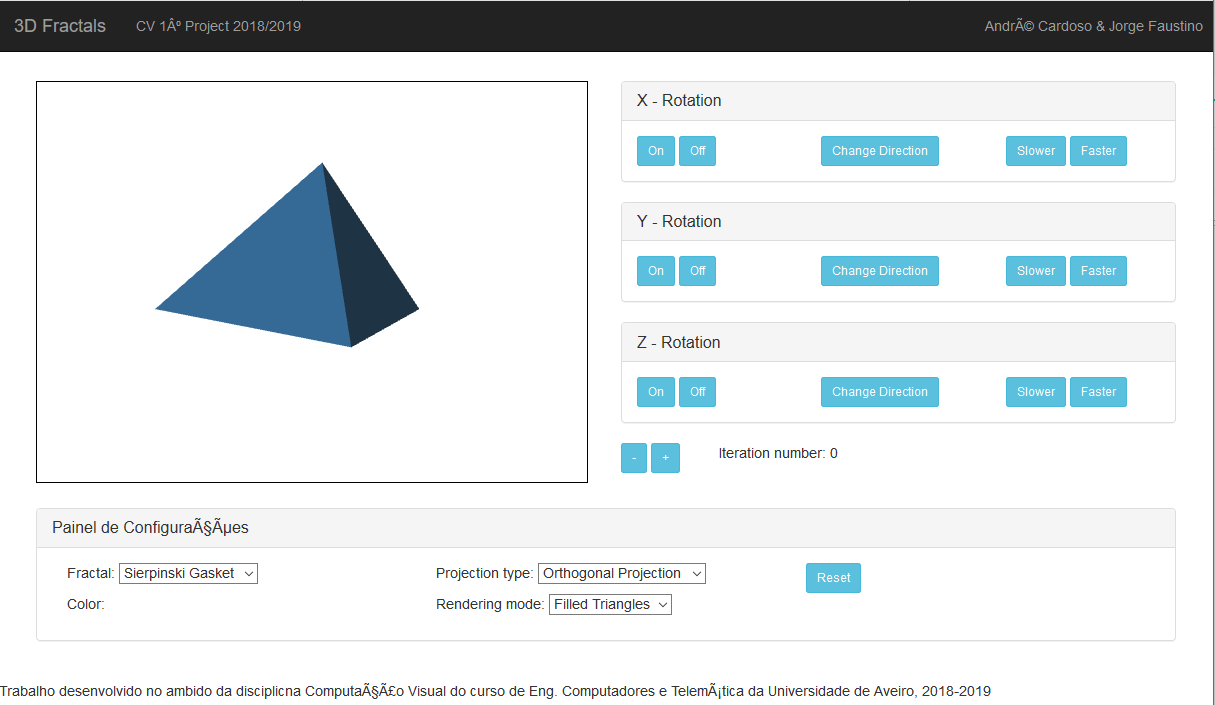


Figure 13 - Interface da aplicação final.

Os tipos de projecções possíveis são Orthogonal Projection[13] e Perspective Projection.[14]

O modos de renderização são Filled Triangles[15], Wireframe e Vertices.

IV. Conclusão

Os fractais são em si objetos complexos, mas com utilidade em vários campos, para compreender certos objetos e padrões encontrados na natureza também são usadas em nas áreas cinemáticas e jogos.

Este trabalho foi muito importante para o nosso conhecimento, o aprofundamento deste tema fez-nos aprender e compreender o método utilizado na manipulação de objetos 3D utilizando algoritmos recursivos. Cumprimos todos os objetivos que nos foram propostos neste projeto em que o principal era a projeção de um Fractal 3D.

V. References

[1] “WebGL.” [Online]. Available: https://www.khronos.org/webgl/. [Accessed: 20-Nov-2018].