Tiago Magalhães Fernandes

Licenciado em Engenharia Informática

Exploração Visual 3D das estrelas da Via Láctea na WEB

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática

Orientador: João Moura Pires, Prof. Dr., Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa

Co-orientador: Fernando Birra, Prof. Dr., Faculdade de Ciências e Tecnologia - Universidade Nova de Lisboa

Júri

Presidente: Prof. Doutor Carlos Viegas Damásio, Prof. Associado, DI/FCT/UNL
Arguente: Prof. Doutor João António Madeiras Pereira, Prof. Associado, DI/IST/UL
Vogal: Prof. Doutor João Moura Pires, Prof. Auxilar, DI/FCT/UNL

Novembro, 2016

Exploração Visual 3D das estrelas da Via Láctea na WEB

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "unlthesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2]. [1] https://github.com/joaomlourenco/unlthesis [2] http://www.di.fct.unl.pt

Agradecimentos

Em primeiro lugar, quero agradecer aos meus orientadores, Professor João Moura Pires e Professor Fernando Birra, pela orientação, disponibilidade e imenso apoio e ajuda para a concretização desta dissertação.

Agradeço à Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa e aos professores que contribuíram para a minha aprendizagem e desenvolvimento académico e pessoal.

Agradeço também ao António Falcão pela disponibilidade que apresentou para ajudar durante o desenvolvimento da dissertação.

Por último, agradeço a toda a minha família, amigos e colegas pelo apoio e encorajamento durante todo o meu percurso na faculdade.

Resumo

Ao longo de cinco anos previstos de análise da nossa galáxia, a missão GAIA, realizada pela Agência Espacial Europeia, irá disponibilizar a informação de mil milhões de estrelas da nossa galáxia, cerca de 1% de todas as estrelas da galáxia, fornecendo aos astrónomos uma representação rica e correta sobre como a nossa galáxia está formada.

Atualmente, existem ferramentas de visualização tridimensionais especializadas para *desktop* capazes de visualizar os dados disponibilizados pelo Gaia, contudo, estas funcionalidades não existem sob forma de aplicações para *browser*.

Nesta dissertação é descrita uma visualização 3D interativa para o publico em geral, onde a maioria dos utilizadores não tem conhecimentos sobre dados astronómicos, e de tal forma que seja compatível com a maioria dos dispositivos existentes com acesso à Internet via *browser*.

Neste trabalho propõe-se uma arquitetura que permite à aplicação adaptar-se às condições da visualização, sendo capaz de produzir visualizações tridimensionais de estrelas em *browsers*, utilizando WebGL para explorar o universo 3D e interagindo com um Servidor de Objetos remoto que transmite os dados de catálogos de estrelas de forma eficiente.

É realizada uma avaliação sobre o protótipo desenvolvido que implementa esta arquitetura, onde é analisado o comportamento e capacidade que este tem de visualizar eficientemente catálogos de estrelas de grandes dimensões.

Palavras-chave: Visualização 3D, Redução de dimensionalidade, otimização, interatividade

Abstract

After five years of analysing our galaxy, the GAIA mission by the European Space Agency is going to release information of 1 billion stars in our galaxy, around 1% of all stars, delivering a rich and correct representation to the astronomy community of how our galaxy is formed.

Currently, various specialised standalone 3D visualization tools exist for desktop able to visualise the Gaia data, however, these tools do not yet exist in form of browser applications.

This dissertation describes an interactive 3D visualization for the general public, where most users do not have extensive knowledge of astronomical data, and in a way that will be compatible with the majority of existing devices with Internet access through a browser.

In this work, an arquitecture is proposed that will allow the application to adapt to the visualization conditions, being capable to produce 3D visualization of large numbers of stars inside a browser, using WebGL to explore the 3D universe and utilizing a remote Object Server that will transmit the star catalog data in an efficient way.

An evaluation will be performed to test the developed prototype that implements this arquitecture, mesuring its performance and capabilities of efficiently visualizing large star catalogs.

Keywords: 3D Visualisation, Dimension reduction, optimization, interactivity

Índice

Capítulo 1

Introdução

1.1 Contexto

Após diversos anos de exploração e análise da nossa galáxia, a missão GAIA [13], realizada pela Agência Espacial Europeia[2], irá disponibilizar no mês de Setembro 2016 um novo catálogo de estrelas que será utilizado para análise e exploração por astrónomos e o público geral. Esta missão tem como objetivo responder a questões fundamentais sobre a formação e evolução da nossa galáxia.

Para isto, esta missão irá analisar e recolher informação posicional de cerca de mil milhões de estrelas pertencentes à galáxia, sendo aproximadamente 1% de todas as estrelas da população celestial. Combinado com outros atributos astrofísicos, que serão recolhidos para cada estrela, estes dados terão a informação e precisão necessárias para ser possível quantificar a formação inicial da galáxia e a subsequente evolução.

Para ser possível usufruir deste novo catálogo de estrelas, foram desenvolvidas diversas ferramentas no âmbito desta missão, com o objetivo de tornar o acesso desta grande quantidade de dados a qualquer utilizador de forma fácil e eficiente, oferecendo mecanismos de visualização e ferramentas especializadas na interação e exploração com este catálogo.

Recentemente, no âmbito da missão GAIA, foi desenvolvido um projeto com o objetivo de responder aos problemas que este novo catálogo introduziu. Este projeto, Interactive Visualization Environment for Large Arquives, que tal como o nome indica, visou criar um ambiente de visualização interativo tridimensional para catálogos de grandes quantidades de estrelas, sendo desenvolvido para plataformas em ambiente *desktop*.

Para satisfazer estes objetivos, o projeto teria de ser capaz de criar visualizações com base num número de estrelas na ordem de dezenas de milhões a mil milhões de estrelas, mantendo foco nas componentes de interação e exploração que podem ser realizadas

pelos utilizadores.

Estas componentes dão a liberdade ao utilizador de navegar nos dados de forma interativa, efetuar seleções de regiões de interesse nos dados, visualizar o conteúdo selecionado usando diferentes técnicas visuais, aplicar diferentes tipos de escalas nos eixos cartesianos, entre outros.

Um dos objetivos principais da missão GAIA será conseguir expor toda esta informação de uma forma compreensível e de fácil acesso para o público geral. A aplicação anteriormente envolvida não constitui uma solução viável para este objetivo de divulgação para o grande público, visto que necessita que o utilizador realize um descarregamento prévio da aplicação e que efetue uma instalação deste no seu computador.

Assim, qualquer solução passará por utilizar a tecnologia web, permitindo que qualquer utilizador em geral consiga aceder a toda esta informação a partir do seu *browser*, não requerendo instalações prévias. Do mesmo modo, as funcionalidades a disponibilizar, assim como a informação que será fornecida, estarão a par do que foi oferecida pela aplicação *desktop*.

1.2 Enquadramento e Motivação

Durante o projeto anterior, foi concluído que o hardware existente hoje em dia não tem capacidade de processamento e armazenamento suficiente para lidar com a dimensão e complexidade do catálogo de estrelas que será fornecido pela missão GAIA. A ferramenta teria de ser capaz de utilizar os dados das estrelas para criar visualizações tridimensionais e interativas, o que seria impossível de realizar se tivesse de utilizar todo o catalogo em simultâneo. Para colmatar esta limitação, foi introduzido um Servidor de Objetos, o qual tem acesso aos dados destas estrelas, e permite efetuar entrega desta informação a um cliente *desktop*.

Deste modo, o servidor efetua um pré-processamento destes dados, de maneira a que não seja necessário transmitir toda a informação do catálogo para o cliente. Este processamento possibilita ao servidor enviar informação simplificada dos dados que o cliente está a visualizar em cada instante, com mínimas perdas visuais da visualização.

O cliente utiliza esta informação simplificada do catálogo para criar visualizações tridimensionais, onde oferece funcionalidades de interação e exploração no espaço da visualização. Como o cliente não possui todos os dados do catálogo, e o detalhe destes dados depende do posicionamento do utilizador no ambiente de visualização, o cliente necessita de comunicar regularmente com o servidor, de modo a requisitar informação sobre novas áreas da visualização. Assim, o cliente terá de atualizar a sua visualização localmente de acordo com a informação que recebe do servidor.

Apesar de o cliente estar ligado ao servidor de objetos, e dependente deste para criação de visualizações de catálogos desta dimensão, o cliente assenta fortemente na tecnologia OpenGL para permitir uma visualização tridimensional de grandes quantidades de estrelas.

Atualmente, com o surgimento de WebGL [11], é introduzida uma oportunidade de utilizar as capacidades que esta API oferece para criação de ambientes tridimensionais complexos em *browsers*. Esta API foi construída com base em OpenGL, conseguindo interagir com as placas gráficas para processamento gráfico. WebGL é atualmente suportado pelos *browsers* principais que são usados pelo público, e suporta um grande volume de plataformas, desde *desktop*, portáteis, *smartphones*, *tablets* e consolas. Acrescentando que WebGL não necessita de *plug-ins* de terceiros para ser utilizado no *browser*, faz com que esta API seja ideal para abranger o publico geral com grande facilidade.

1.3 Problema

Encontrar uma arquitetura que, integrando com o servidor de objetos, permita a visualização tridimensional e interativa da quantidade massiva de estrelas disponíveis no catálogo GAIA, através de *browsers* web compatíveis com WebGL.

A visualização e a interação destinam-se ao público em geral, permitindo uma perceção 3D dos conjuntos de estrelas e algumas características das estrelas, nomeadamente as suas posições, cores, luminâncias, massas e velocidades orbitais.

A arquitetura deve atender a diferentes características de latência, largura de banda, memória e velocidade de *browser*, e diferentes princípios de interação.

1.4 Objetivos e Contribuições

É efetuada uma proposta de uma arquitetura que faz uso do servidor de objetos, onde terá acesso ao catálogo do GAIA, que irá possibilitar a criação de visualizações tridimensionais inteligentes de catálogos de grandes dimensões. Esta arquitetura utilizará as capacidades de WebGL para permitir a realização de visualizações complexas em ambientes 3D com grandes quantidades de estrelas em *browsers*.

Esta arquitetura deverá adaptar-se a diferentes volumes de dados, diferentes entidades de dados e características dos dispositivos com base na capacidade de processamento e memória. A arquitetura também deverá permitir ao utilizador interagir com a visualização, ao oferecer funcionalidades de exploração e navegação nos dados, de maneira a que a experiência do utilizador seja fluida.

É realizada uma implementação de um protótipo que implementa esta arquitetura, efetuando uma avaliação do seu comportamento e capacidade de produzir visualizações 3D interativas no *browser* com catálogos de estrelas de grandes dimensões.

Capítulo 2

Estado de Arte

Neste capítulo será efetuado o estudo do estado de arte no contexto deste projeto.

Em primeiro lugar será efetuada uma descrição da missão Gaia, dos seus objetivos e conteúdo que será disponibilizado.

De seguida será realizada uma descrição do Servidor de Objetos e das funcionalidades que este oferece para possibilitar a visualização de catálogos de estrelas de grandes dimensões.

Uma análise pormenorizada será efetuada sobre o projeto realizado anteriormente no contexto da missão Gaia, o qual tem o objetivo de criar visualizações interativas de grandes quantidades de estrelas, com a possibilidade de realizar seleções complexas destes dados para análise de regiões de interesse. Este projeto foi desenvolvido para um ambiente computacional do tipo *desktop*.

Depois, será realizada uma descrição sobre o projeto atual GAVIDAV, realizado no âmbito da missão Gaia, com o objetivo de realizar visualizações interativas dos dados do catálogo fornecido pelo Gaia, de modo a abranger o maior número possível de utilizadores através de uma aplicação Web.

Sendo o WebGL a API para o desenvolvimento de aplicações Web 3D, serão analisadas as suas capacidades para visualização deste tipo de dados no *browser*, bem como possíveis bibliotecas que implementem técnicas de processamento de dados e processamento gráfico eficientes e que possam ser adaptados à visualização complexa de grandes quantidades de estrelas.

No contexto deste tema, serão analisadas aplicações que são utilizadas atualmente pelos astrónomos para visualização e análise destes dados, observando como os astrónomos interagem e que tipo de visualizações são usadas para análise e exploração dos dados.

2.1 Gaia

Lançado em 2013, o satélite, chamado Gaia Spacecraft, iniciou uma missão com duração de 5 anos com o objetivo de observar e analisar uma vasta quantidade de estrelas da nossa galáxia [13].

Este satélite analisa dezenas de atributos de cada estrela, mas o seu foco principal é revelar a composição, formação e evolução da nossa galáxia. Para isto, o instrumento analisará 1% de todas as estrelas da nossa galáxia, que contém cerca de 200 mil milhões de estrelas.

Atualmente, a melhor representação existente sobre a nossa galáxia resulta da missão Hipparcos [14], que produziu um primeiro catálogo de 118000 estrelas, chamado Hipparcos, e um segundo catálogo de cerca de 2 milhões de estrelas, chamado Tycho, cujas posições teriam menos precisão em comparação com os dados recolhidos no catálogo Hipparcos.

Estes catálogos são usados pela comunidade de astrónomos profissionais, sendo que o novo catálogo Gaia irá trazer uma grande quantidade de nova e mais detalhada informação sobre um maior número de estrelas, tendo um grau de precisão 100 vezes superior ao catálogo Hipparcos.

2.1.1 Atributos

Para cada estrela que será analisada pelo satélite Gaia, serão recolhidas informações de alta precisão sobre dezenas de atributos, desde informação posicional, temperatura, luminosidade, raio, velocidade orbital, composição, entre outros atributos usados pelos astrónomos para análise de estrelas.

Entre todos estes atributos, a informação posicional das estrelas é o foco principal desta missão. A informação posicional é definida em coordenadas equatoriais [1], como pode ser observado na Figura 2.1, onde as coordenadas são definidas com base no equador da Terra, onde Right Ascencion define longitude, Declination define latitude e Parsec define a distância entre a Terra e a estrela. Ambos Right Ascencion e Declination são medidos em ângulos, enquanto que Parsec é medido utilizando a técnica de Parallax, que consiste em medir a distância que a estrela se move no céu quando a Terra está em duas posições diferentes da sua orbita com o Sol.

Estas medidas são então usadas para posicionar cada estrela no sistema de coordenadas, podendo converter para qualquer outro sistema de coordenadas, como por exemplo um sistema de coordenadas cartesianas, para fácil integração num ambiente 3D.

2.1.2 Casos de uso

Durante o decorrer da missão Gaia, foram definidos diversos casos de uso submetidos por cientistas, engenheiros e publico geral, com o objetivo de descrever para que fins

[1] http://astronomy.swin.edu.au/cosmos/E/Equatorial+Coordinate+System

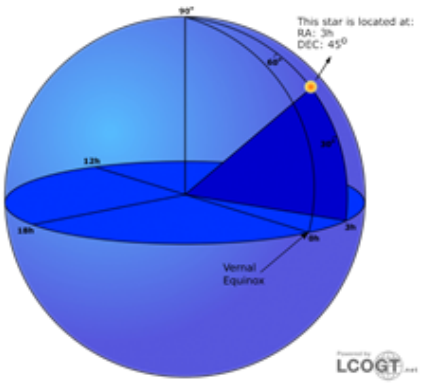

Figura 2.1: Representação de sistema de coordenadas equatorial

utilizariam os dados que serão fornecidos pela missão Gaia [2] . Estes casos de uso cobrem uma vasta gama de objetivos, desde diversos tipos de visualização dos dados, seleções de estrelas, disponibilização de informação especifica de estrelas, entre outros.

Para efeitos deste projeto de tese de mestrado, apenas se irão focar os casos de uso que sejam mais apropriados para o tema do projeto, apresentando 4 casos de uso que foram submetidos no portal do Gaia. Todos estes cenários são publicamente acessíveis a partir do portal Gaia [3].

2.1.2.1 Cenário 1

GDAS-BR-09 [4]

Cenário: "I want to manipulate a 3D cube of the Milky-Way."

Comentário: "Very useful to understand the structure of the Galaxy. By 3D I mean three SPATIAL dimensions."

- Neste cenário, o utilizador indica que quer manipular um cubo 3D na Via Láctea. Não existe muita informação sobre este caso de uso, mas para satisfazer este caso de uso, terá de ser criada uma visualização 3D da nossa galáxia e oferecer ferramentas de interação e exploração desta visualização.

2.1.2.2 Cenário 2

GDAS-BR-05 [5]

Cenário: "I want a pretty colour picture for my power-point presentation, created using Gaia data. For example (first that comes in mind): I would like to select all stars belonging to the halo of the Milky Way and colour them according to their 6D phase space information, assigning to each group its own colour. Then I want to project this on the sky either in ecliptic coordinates or whatever I choose, so that I can illustrate how

[2] http://great.ast.cam.ac.uk/Greatwiki/GaiaDataAccess
[3] http://great.ast.cam.ac.uk/Greatwiki/GaiaDataAccess
[4] http://great.ast.cam.ac.uk/Greatwiki/GaiaDataAccess/GdaUseBrowsing
[5] http://great.ast.cam.ac.uk/Greatwiki/GaiaDataAccess/GdaUseBrowsing

the halo is structured and where the streams are, to make a picture like the one from the SDSS one, but with Gaia data"

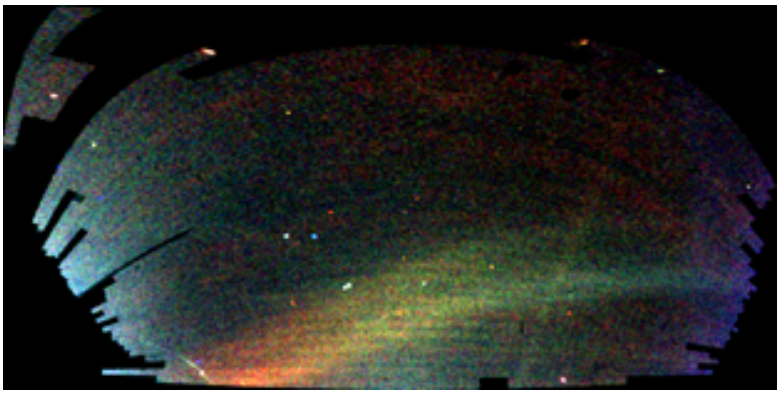

Figura 2.2: Demonstração de uma visualização do catálogo SDSS

Comentário: "This is one type of service that would make astronomers life much better, and their presentations much nicer."

- Este cenário descreve uma visualização colorida do catálogo Gaia. Para isto, o utilizador quer utilizar a informação posicional das estrelas, e atribuir à cor destas estrelas um atributo em particular, com o objetivo de visualizar e mostrar como o centro da galáxia está estruturado. Em conjunto com este cenário, é mostrada uma imagem do catálogo SDSS [7], como pode ser visto na figura 2.2 , como exemplo de uma visualização possível do catálogo Gaia.

2.1.2.3 Cenário 3

GDAS-PR-03 [6]

Cenário: "Make a movie of a flight through the Milky Way disc, respecting the distances and apparent luminosity of objects."

- Neste cenário, o utilizador quer realizar um filme onde simula uma viagem através do disco central da galáxia, respeitando as distancias e luminância dos objetos. Semelhante ao cenário 1, poderá ser criada uma visualização do Gaia, onde o utilizador poderá interagir usando um outro tipo de interação com a visualização, e que lhe permita navegar através dos dados.

2.1.2.4 Cenário 4

GDAS-GA-11 [7]

Cenário: "I want a tool that provides a face-on view of the Milky Way, as inferred from Gaia data."

[6] http://great.ast.cam.ac.uk/Greatwiki/GaiaDataAccess/GdaUseOutreach
[7] http://great.ast.cam.ac.uk/Greatwiki/GaiaDataAccess/GdaUseGal

Comentário: "Image centered on the Galactic Centre. Alternative images can be done centered on the Sun, or any other star or object. The image could be shown online as well as saved in JPG, GIF, PNG, EPS formats."

Neste cenário, o utilizador quer visualizar a galáxia posicionando-se no centro da galáxia, no sol, ou outra estrela ou objeto. Tal como os cenários anteriores, este requer a criação de uma visualização da galáxia, e posicionamento da câmara em locais definidos pelo utilizador, de modo a observar a galáxia.

2.2 Servidor de Objetos

Acesso a dados de catálogos de estrelas é usualmente um processo longo e dispendioso, no qual é necessário ter acesso a todos estes dados localmente de forma a produzir visualizações interativas que os representam no dispositivo do utilizador.

Estes catálogos de estrelas são acessíveis pelo público através de portais online, como o portal VizieR [8], que disponibilizam dezenas de catálogos, onde qualquer utilizador os pode transferir para o seu dispositivo. Estes catálogos podem conter entre milhares e milhões de estrelas, onde cada estrela poderá conter dezenas de atributos.

Catálogos de grandes quantidades de estrelas podem por vezes ocupar centenas de *megabytes*, ou mesmo *gigabytes* de espaço, tornando difícil a utilização destes por parte de um utilizador comum, necessitando de transferir esta grande quantidade de dados pela Internet, e utiliza-los localmente.

No âmbito do projeto IVELA, no qual tem como objetivo disponibilizar visualizações 3D de grandes quantidades de estrelas de forma interativa, seria necessário disponibilizar estes dados de uma forma inteligente e rápida, fazendo com que o utilizador não precise de aceder a todos estes dados simultaneamente, e apenas aceder a uma fração destes dados suficiente para representar o catálogo de forma simplificada.

Desenvolvido pelo grupo de investigação Fork Research [9] , no âmbito da missão Gaia e do projeto IVELA, foi construído um servidor de objetos que realiza um pré-processamento de catálogos de estrelas e dispõe de uma API no qual aplicações podem comunicar e requisitar estes dados.

2.2.1 Estrutura de dados

O servidor de objetos utiliza a estrutura de dados Octree, onde subdivide o catálogo de estrelas por octantes, utilizando a informação posicional de cada estrela.

Uma *octree* [6] [8] é uma estrutura de dados em árvore, onde cada nó contém 8 filhos, como pode ser observado na Figura 2.3. Esta estrutura de dados particiona o espaço tridimensional ao subdividi-lo, recursivamente, em octantes, com o objetivo de separar objetos para facilitar a sua pesquisa com base na sua posição. Numa *octree* convencional,

[8] http://vizier.u-strasbg.fr/viz-bin/VizieR
[9] http://www.forkresearch.com

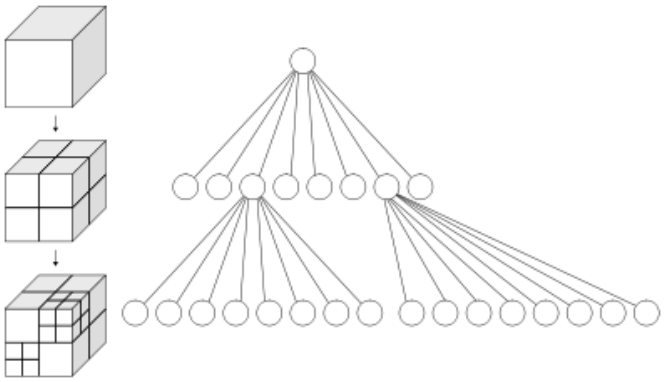

Figura 2.3: Representação visual de uma Octree

a *octree* é dividida recursivamente até que todos os objetos estejam separados por octantes diferentes nas folhas da árvore.

Assim, foi utilizada esta estrutura de dados, onde as estrelas do *dataset* são armazenadas numa lista em cada nó da árvore. Nesta *octree*, cada nó contém um *subset* dos dados, onde cada nível de profundidade a árvore representa um nível de detalhe.

Os dados são construídos de forma aditiva, significando que ao escolher um conjunto de octantes para serem visualizados, todos os octantes pais até à raiz da *octree* deverão também ser incluídos no resultado final.

2.2.2 API para visualização 3D

O servidor de objetos comunica com o cliente através de mensagens. Para requisitar um octante de um catálogo, o cliente adiciona parâmetros à mensagem de forma a identificar esse octante, indicando explicitamente o identificador do catálogo de estrelas da visualização e o identificador do octante.

Para além destes parâmetros, o cliente poderá indicar se o resultado deverá ser comprimido (indicando um tipo de compressão suportado pelo servidor) e em que formato a resposta deverá ser transmitida (texto, JSON, XML).

Ao receber esta mensagem, o servidor responde ao cliente com o octante. Devido ao espaço que o octante poderá ocupar, este é dividido em blocos de menor dimensão, chamados 'Payload Block', que são enviados para o cliente separadamente, cabendo ao cliente de os juntar quando estes chegam ao destino.

Esta informação é devolvida através de mensagens que incluem um parâmetro chamado *payload*, no qual irá conter blocos de dados que representam o octante. No fim do cliente receber todos os blocos de dados, este combina-os para extrair o octante requisitado.

2.3 Interactive Visualization Environment for Large Archives (IVELA)

IVELA, ou Interactive Visualization Environment for Large Archives [10] , é um projeto desenvolvido no âmbito da missão Gaia. Este projeto teve como objetivo criar uma ferramenta de visualização e interação de grandes quantidades de dados em 3 dimensões. Nessa ferramenta é possível visualizar desde 1 milhão de estrelas até mil milhões de estrelas, quer na plataforma Windows, quer na plataforma Mac.

Esta aplicação lida com esta quantidade de informação de modo inteligente e eficiente, criando visualizações onde o utilizador pode explorar, interativamente, e sem tempos de espera, suportando uma grande variedade de dispositivos com diferentes capacidades de hardware, desde portáteis de baixa gama até *desktops* melhor equipados.

Este projeto realizou-se durante o estágio de terceiro ano do curso de Mestrado em Engenharia Informática, criando uma aplicação no espaço de 1 semestre no âmbito do estágio, sendo que após a finalização do estagio, o desenvolvimento da aplicação continuou a ser realizado durante mais um ano, implementando novas funcionalidades, realizando otimizações e criando uma versão final de entrega para a ESA.

2.3.1 Plataforma e Frameworks

Sabendo que IVELA seria uma aplicação interativa num ambiente 3D, foi importante escolher uma *framework* de desenvolvimento apropriada para este tipo de aplicações.

Esta aplicação teria de ter a capacidade de criar ambientes 3D capazes de processar milhões de objetos em tempo real, ter uma API que disponibilizasse acesso a uma grande variedade de funcionalidades e estruturas de dados apropriadas para o projeto, oferecer otimizações significativas para ser possível visualizar esta grande quantidade de dados, e ter um ambiente de desenvolvimento de fácil e rápida prototipagem.

Tendo experiência anterior com o motor de jogos Unity [11], rapidamente se chegou à conclusão que esta *framework* era apropriada para o desenvolvimento da aplicação. Unity é uma *framework* de desenvolvimento, poderosa e flexível, de aplicações interativas e jogos para múltiplas plataformas e dispositivos.

Unity foca-se em oferecer um ambiente de desenvolvimento fácil e eficaz para desenvolvimento de aplicações, diminuindo o tempo de aprendizagem e desenvolvimento da aplicação e, ao mesmo tempo, oferecendo funcionalidades complexas e de alta qualidade.

2.3.2 Técnicas de Visualização de Estrelas

No projeto IVELA, os principais focos foram a criação de visualizações de grandes quantidades de estrelas, ferramentas de seleção para análise de conjuntos de estrelas, e navegação interativa com estas visualizações.

[10] http://www.ca3-uninova.org/project_ivela
[11] https://unity3d.com

Sendo o tema desta dissertação focado no aspeto de visualização e interação de estrelas, as próximas secções irão focar-se nos aspetos do IVELA que estão diretamente relacionados com este tema.

2.3.2.1 Tipos de Visualização

IVELA suporta dois tipos de visualizações: visualização local e visualização dinâmica.

A visualização local dá a possibilidade ao utilizador de criar visualizações usando *datasets* locais no seu computador, carregando todos os dados em simultâneo em memória para uso na visualização. Ao ter todo o *dataset* carregado em memória introduzem-se limitações na quantidade de informação possível de utilizar na aplicação.

Após testes da aplicação IVELA em hardware de gama média, verificou-se que a visualização local de *datasets* é apenas apropriada para *datasets* que têm uma dimensão menor de 4 milhões de estrelas.

Para ser possível a visualização de mais estrelas na aplicação, neste caso, mil milhões de estrelas, optou-se pela utilização e comunicação com um servidor remoto. Este tipo de visualização veio-se a chamar Visualização Dinâmica. Com esta abordagem, os dados são pré-processados no servidor, onde são subdivididos em pequenos conjuntos que o cliente pode requisitar em tempo-real.

2.3.2.2 Estrutura de dados

Devido ao enorme tamanho do catálogo Gaia, o cliente fica impossibilitado de poder utilizar todos os dados em simultâneo na sua máquina, pelo que estes dados teriam de ser processados de alguma forma. A solução passou por duas vertentes. Por um lado, redução da quantidade de informação que é necessária enviar para o cliente, apenas enviando secções do catálogo que são visíveis num dado momento.

Por outro lado, a simplificação dos dados quando estes dizem respeito a estrelas estão a uma distância suficientemente grande que permita que o utilizador não perca informação visual se o número de estrelas nessa área for reduzido.

Para combater este problema de complexidade, foi utilizada a estrutura de dados *octree*, com o objetivo de replicar a estrutura de dados que se situa no servidor de objetos.

Devido a limitações da *framework* Unity, cada nó da *octree* contém uma lista com um tamanho máximo de 65536 estrelas. Este limite deve-se ao tamanho de indexação utilizado pelo Unity para *arrays* que são transmitidos para os GPUs para processamento gráfico. Como Unity utiliza *arrays* com índices de 16 bits, isto limita a que todos os *arrays* que sejam enviados para os GPUs tenham um número máximo de 2 exp(16) (65536) índices.

Unity utiliza este limite para suportar um maior número de GPUs possível, visto nem todos os GPUs suportarem indexamento de *arrays* superior a 16bits, não sendo muito prejudicial para criação de aplicações interativas, como jogos de computador.

2.3.2.3 Visualização das estrelas

Para o IVELA, foram definidos dois tipos de visualização de dados, nuvem de pontos e *billboards*.

Nuvem de pontos, como o nome indica, são representações de objetos sob forma de pontos em 3D, sendo cada ponto representado por um pixel e por uma cor no ecrã, como é mostrado na Figura 2.4. Para criar este tipo de visualizações, é usada a estrutura de dados *mesh* [12] do Unity, normalmente associada a modelos assentes num conjunto de polígonos, sendo que na maior parte dos casos usados triângulos.

A estrutura permite guardar uma lista de pontos 3D e um conjunto de atributos a eles associados. Aquando do envio dos dados para o GPU pode-se definir a forma como os pontos e os atributos a eles associados são interpretados.

Por exemplo, para desenhar triângulos, cada grupo de 3 pontos e respetivos atributos são associados aos vértices dum triângulo. No caso da visualização sob a forma de nuvem de pontos, cada ponto é interpretado de forma isolada. Neste caso, para além da posição usa-se um atributo adicional para definir a cor do pixel desenhado.

O uso de *meshes* em placas gráficas é bastante eficiente, conseguindo-se processar milhões de vértices em tempo real numa grande variedade de hardware, sendo uma estrutura de dados muito eficaz para visualização de nuvem de pontos.

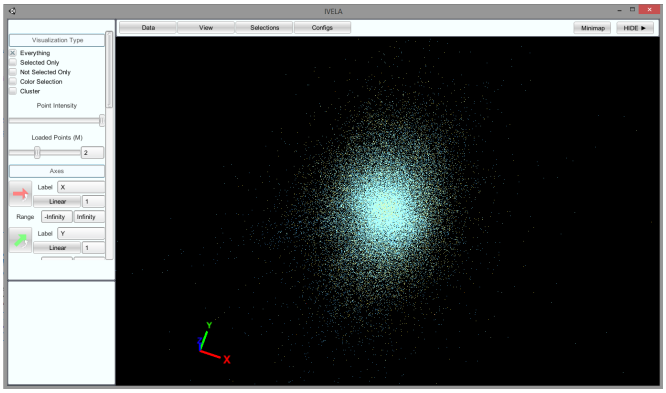

Figura 2.4: Visualização de um dataset usando a técnica de Point-Clouds

O tamanho de cada estrela em si, comparativamente com o espaço entre outras estrelas e a dimensão da galáxia, é tão pequeno que elas seriam vistas simplesmente como pontos no céu. Mas, quando se efetuam observações com instrumentos, grande parte destas estrelas não são observadas como pontos, mas pelo contrário, estas dão uma ilusão de tamanho significativamente superior ao tamanho real da estrela, como pode ser observado na Figura 2.5 [13] .

Este fenómeno é causado pelos espelhos usados nos telescópicos para captar a luz das estrelas e é tão notório quanto maior for a quantidade de luz produzida pela estrela, o que

[12] http://docs.unity3d.com/ScriptReference/Mesh.html

[13] https://www.nasa.gov/feature/goddard/hubble-uncovers-the-fading-cinders-of-some-of-our-galaxy-s-earliest-homesteaders

causa um brilho intenso que pode ser observado em todas as fotografias de astronomia.

Este efeito é normalmente usado em aplicações de exploração de estrelas, conseguindo gerar representações mais ricas e coloridas que visualizações usando nuvens de pontos, onde cada estrela é representada por apenas um pixel.

Figura 2.5: Imagem onde são observadas estrelas com um tamanho aparente causado pela luminosidade de cada estrela

Para representar cada estrela, dando um efeito semelhante ao descrito anteriormente, as nuvens de pontos não são apropriadas, podendo abordar-se este problema usando uma técnica tradicional em computação gráfica recorrendo a uma malha poligonal para visualizar cada estrela como um objeto que ocupa um determinado espaço, visualizando-se a sua fronteira.

Ao fazer isso estaríamos a multiplicar o número de estrelas pelo número de vértices usados nessa mesma malha. Para representar cada estrela, teria de ser usado uma esfera com dezenas ou centenas de vértices e polígonos, aumentando substancialmente a quantidade de informação que teria de ser processada para ter uma visualização deste tipo.

Como não é desejável aumentar a complexidade, a solução passa por usar uma textura para se proceder à visualização das estrelas. A ideia dos *billboards* é a de mostrar sempre um polígono com a face virada para a câmara.

Felizmente, um objeto esférico como uma estrela será sempre visualizado sob a forma aproximada dum círculo, pelo que não haverá problemas no uso desta técnica ao fazer variar o ponto de vista. Esta solução pode ser observada na Figura 2.6.

Usando apenas uma face virada para a câmara, esta face utiliza apenas 4 vértices e 2 triângulos para a representar, sendo significativamente menor do que se cada estrela fosse representada por uma esfera, que poderia conter dezenas a centenas de vértices e triângulos para ter uma representação de qualidade da sua silhueta.

O Unity fornece uma estrutura de dados que implementa este tipo de objeto, chamada sistema de partículas. Esta estrutura consiste na utilização dum *array* de objetos, onde cada um deles pode ser representado como um *billboard*, podendo definir-se a sua posição em 3D, o seu tamanho e a sua cor. Sendo este sistema de partículas implementado

nativamente no Unity e, estando esta *framework* constantemente a receber atualizações que melhoram o seu desempenho, a aplicação prática dos sistemas de partículas é muito eficiente.

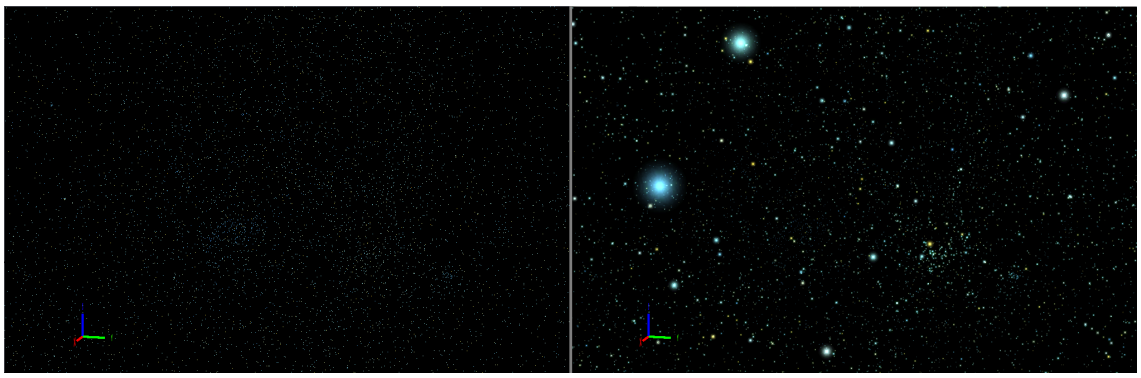

Figura 2.6: Demonstração do catálogo Hipparcos, onde na figura da esquerda é representado o catálogo usando uma nuvem de pontos, e na figura da direita, é representado o mesmo catálogo na mesma área usando *billboards*

Tal como a nuvem de pontos, esta estrutura de dados utiliza o mesmo conteúdo contido na *octree* para visualização das estrelas, aproveitando-se das mesmas técnicas de otimização efetuados pelo Unity. Utilizando o sistema de partículas, é possível efetuar visualizações de estrela com um conjunto na ordem das 100,000 a 1,000,000 estrelas, em tempo real, na maioria de hardware capaz de correr a aplicação.

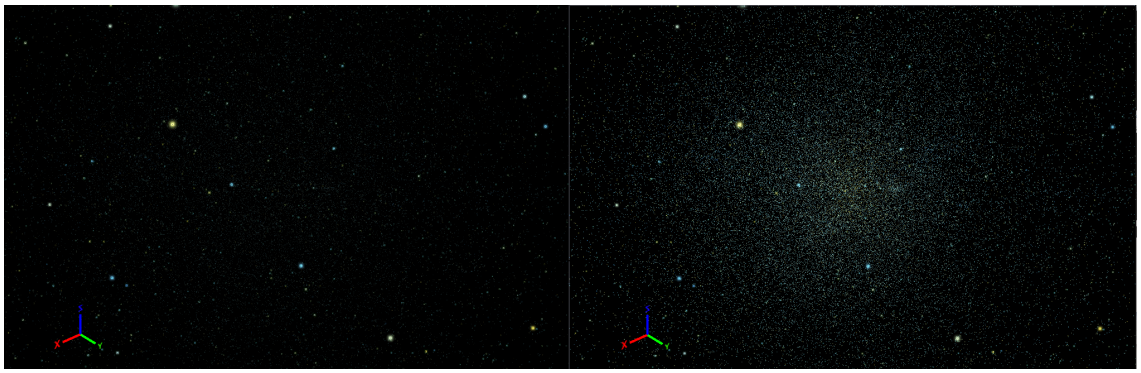

Figura 2.7: Nestas duas imagens é visualizada a mesma área da figura 2.6, mas a uma distância significativa. Na figura na esquerda, são apenas utilizados *billboards*, e na imagem na direita, são usados *billboards* em conjunto com a nuvem de pontos

A utilização de *billboards* para representar cada estrela introduz um novo problema. Como cada *billboard* é representado por uma face em 3D, quando esta face está a uma distância significativa da câmara tal que o tamanho da face é menor do que um pixel do ecrã, este *billboard* simplesmente desaparece, pois, o GPU não consegue definir um pixel para desenhar este *billboard*, sendo que quanto maior for a distância de cada estrela à câmara, mais se agravará este problema de visualização, como é observado na Figura 2.7.

Devido a este problema, a utilização de *billboards* não é suficiente para criar uma visualização das estrelas deste tipo, pelo que esta visualização terá de ser combinada juntamente com do tipo de nuvem de pontos, na qual se conseguem visualizar todas as estrelas, independentemente da distância a que estas se encontrão da câmara.

2.3.3 Visualização Dinâmica

A visualização local de catálogos de estrelas no cliente é ótima para catálogos de pequena dimensão, na ordem de 1-5 milhões de estrelas, mas no mundo da astronomia, este número é insignificante em comparação com a imensa quantidade de estrelas existentes no cosmos.

A missão Gaia terá como objetivo disponibilizar um catálogo de cerca de 2 mil milhões de estrelas. Rapidamente se chegou à conclusão que não existe hardware atual que consiga criar visualizações em 3D desta enorme quantidade de estrelas em tempo real e de forma interativa, pelo que a melhor solução para concluir este objetivo seria a utilização de um servidor remoto onde realizará um pré-processamento do catálogo para transmissão dos dados para o cliente.

Antes da visualização dinâmica ser iniciada, o cliente conecta-se ao servidor de objetos usando um sistema de autenticação, onde depois de autenticado, terá acesso a uma lista de visualizações disponíveis. Sabendo esta lista de visualizações, o cliente poderá dar inicio a uma visualização dinâmica, transmitindo o pedido ao servidor.

A estrutura de dados que contém toda a informação necessária para a visualização dinâmica encontra-se no servidor de objetos, sendo criada ao introduzir os dados numa estrutura *octree*.

Como descrito anteriormente, estes dados são inseridos em listas nas folhas da *octree*, que correspondem ao nível de detalhe mais elevado da árvore, onde se encontram as estrelas reais, e todos os outros nós da estrutura de dados representam listas simplificadas dos filhos desses nós.

Isto possibilita ter uma visualização de um catálogo de estrelas utilizando diferentes níveis de detalhe para regiões do catálogo que se situam a diferentes distâncias da posição em que o observador se encontra.

Sabendo que o servidor funciona em modo *stateless*, onde não conhece o estado da visualização dos clientes a ele conectados, isto impõe que os clientes tenham de calcular e realizar pedidos sobre quais dados que serão necessários para visualizar o catálogo. Este processo não tem custos de computação elevados para o cliente, libertando bastante capacidade de processamento pela parte do servidor, podendo assim atender um maior número de pedidos.

Quando esta visualização dinâmica é iniciada, o servidor de objetos envia uma mensagem para o cliente contendo *metadata* sobre como a *octree* está construída. O cliente, ao receber esta mensagem, irá recriar a estrutura da *octree* localmente, mas sem os dados das estrelas.

O processo de requisição dos dados ao servidor de objetos é efetuado cada vez que o utilizador modifica o volume de visão da câmara dentro da aplicação. Este processo envolve fazer uma pesquisa na *octree*, para determinar os octantes necessários para visualização.

Ao percorrer a *octree*, o algoritmo irá escolher os octantes com base na distância a que se encontram da câmara e se estes se encontram no campo de visão da câmara. Quanto menor for a distancia entre os octantes e a câmara, maior será a profundidade que o algoritmo irá percorrer na *octree*, escolhendo octantes com um nível de detalhe mais elevado.

Assim, para estrelas que se situam perto da câmara, o detalhe será muito elevado, mas para octantes que se situam a grandes distâncias, estes não necessitarão de ser representados por níveis de detalhe muito elevados, pelo que poderão ser utilizadas simplificações para representação das estrelas situadas nesse octante, reduzindo significativamente o número de estrelas que terão de ser requisitadas ao servidor de objetos e visualizadas no cliente, tendo um impacto visual mínimo na imagem final.

O cliente efetua algumas otimizações neste processo, ao comparar os octantes que encontrou ao percorrer a *octree*, com os octantes que já estão carregados em memoria, não necessitando assim de requisitar esses octantes uma segunda vez ao servidor.

Adicionalmente, quando o cliente começa a ter problemas de espaço de memória RAM, entra em ação o mecanismo de *cache* implementado no cliente, sendo libertadas as listas de objetos correspondentes a nós que não estão a ser visualizados.

Quando uma visualização é atualizada com novas estrelas, o cliente processa e aplica as novas listas às estruturas de dados da aplicação, de modo a que todas as funcionalidades continuem operacionais, tais como seleção de objetos, diferentes escalas nos eixos cartesianos, entre outros.

Ao utilizar esta visualização dinâmica, é possível criar visualizações de catálogos que contêm milhares de milhões de estrelas, em tempo real e interativamente. Pode ser observado na Figura 2.8 um catálogo de 100 milhões de estrelas, usando apenas 2 milhões de estrelas no cliente.

2.4 GAVIDAV

A exploração científica beneficia da habilidade de interagir visualmente, identificar e selecionar objetos em múltiplas representações numa maneira fácil e eficaz. O conteúdo que será disponibilizado pelo Gaia, de enormes proporções, impõe grandes desafios na maneira como se interage e visualiza este catalogo, pelo que o GAVIDAV irá disponibilizar ferramentas para possibilitar este tipo de interação com o catálogo.

O projeto Gaia Added-value Data Visualization, ou GAVIDAV, irá focar-se em três aspetos, acesso e visualização do conteúdo disponibilizado pelo Gaia, interatividade destas visualizações usando visualizações multi-vistas, e colaboração entre múltiplos utilizadores.

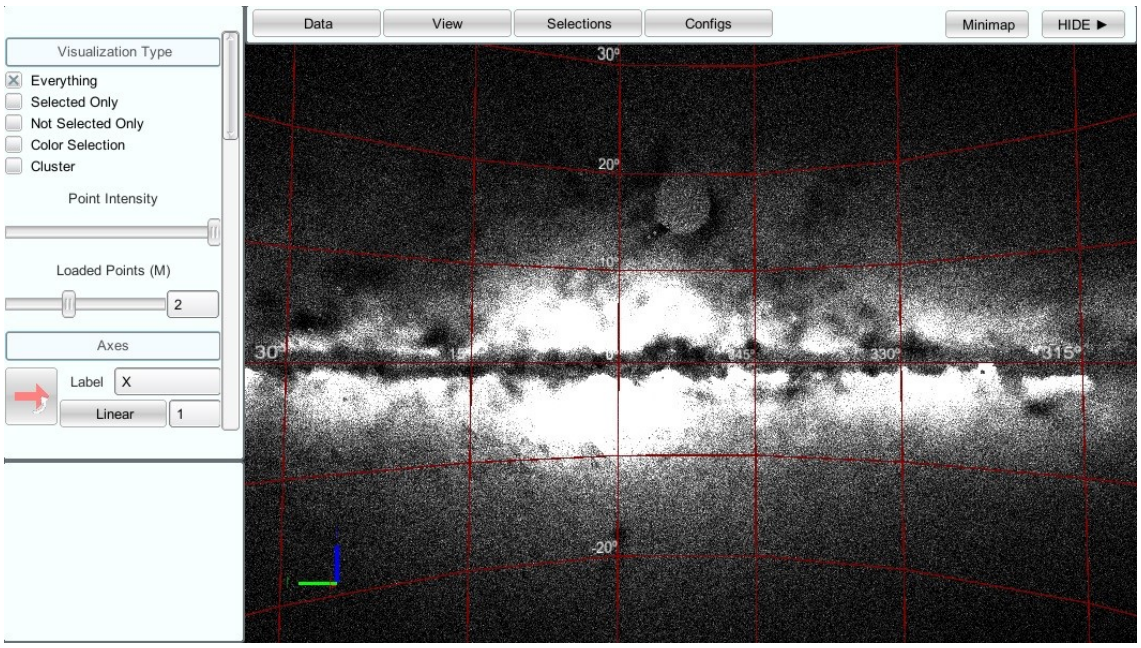

Figura 2.8: Visualização de um catalogo de 100 milhões de estrelas, usando apenas 2 milhões de estrelas no cliente

Tendo acesso a uma tão elevada quantidade de informação, torna-se difícil conseguir analisar estes dados de forma eficiente e eficaz, pelo que terão de ser criadas representações inteligentes destes dados, disponibilizando-os em múltiplas vistas e fornecendo ferramentas de interação complexas para a sua exploração.

A exploração eficiente de dados tem que ser interativa, envolvendo mecanismos de exploração como *panning*, *zooming*, mudança de perspetiva, seleção de dados, identificação de dados em múltiplas vistas, entre outros.

A colaboração é um aspeto muito importante para o desenvolvimento de ideias e partilha de informação, pelo que é necessário oferecer ferramentas de *highlighting* de regiões nas visualizações, adição de comentários, links e outros dados que são usados pelos utilizadores novos, que serão a maior fração dos utilizadores finais do Gaia.

2.4.1 Added Value Interfaces

Added Value Interfaces, ou AVIs, são módulos de interfaces para visualização dos dados fornecidos pelo Gaia, especializados em tipos de visualização e interação específicos na área de astronomia. Estes módulos focam-se em tipos de visualização e seleções concretas que serão usados dentro do portal GAVIP.

Para o âmbito do projeto GAVIDAV, estas AVIs serão criadas primariamente a partir dos casos de uso e necessidades dos utilizadores que irão utilizar o portal, maioritariamente astrónomos, oferecendo ferramentas e visualizações para estes poderem analisar e explorar o catalogo do Gaia. Estas AVIs poderão comunicar entre elas, conseguindo manter os seus estados sincronizados, oferecendo assim visualizações multi-vistas.

Uma AVI importante para este projeto, e para a missão Gaia, será uma AVI focada na visualização 3D das estrelas da nossa galáxia, fornecendo uma interface de visualização destes dados e ferramentas de interação com a visualização, sendo esta AVI o foco do tema do projeto desta tese de mestrado.

2.4.2 Plataforma

Para disponibilizar estas AVIs ao utilizador final, estes módulos utilizarão o portal Gaia Added Value Interface Portal, onde estas AVIs estarão guardadas em conjunto com este portal, situado perto da base de dados que armazena o catálogo do Gaia, onde terá um acesso rápido e eficiente.

Para as AVIs serem implementadas neste portal, estas terão de implementar um sistema cliente/servidor, onde é disponibilizada uma interface ao utilizador que é acedida através do *browser*, e este fará a comunicação com a secção do servidor da AVI contido no portal.

2.5 WebGL

Antes da introdução de WebGL, os *browsers* usavam HTML para criação de aplicações interativas, mas este era bastante limitado em termos de complexidade e performance, pelo que a única opção, para criar estas aplicações interativas mais complexas num *browser*, seria utilizar *plug-ins* de terceiros para conseguir desenvolver e utilizar este tipo de aplicação, tanto 2D como 3D.

Estes *plug-ins* ofereciam APIs e ambientes de criação para fácil desenvolvimento destas aplicações, mas com o requisito de que o utilizador teria de descarregar estes *plug-ins* e instalá-los no seu *browser*, o que em certos casos introduziria muitas limitações, tais como a não existência de suporte para tablets, mobile e consolas, sendo apenas suportado para plataformas em computador.

Para desenvolvimento destas aplicações para mobile, as opções mais populares eram Java, Flash, Silverlight e Unity WebPlayer, sendo que todas estas opções obrigam a que o utilizador descarregue *plug-ins* e que os instale no seu computador.

Tanto Java como Flash são limitados na capacidade de criação de ambientes 3D complexos, pelo que uma das melhores opções para desenvolver estas aplicações era o *plug-in* criado pelo Unity, Unity WebPlayer. Unity oferece uma opção para publicar aplicações e jogos de modo a serem utilizados diretamente no *browser*, conseguindo explorar muitas das otimizações fornecidas pelo motor de jogo, sendo que continua a ser um requisito o *plug-in* para ser possível utilizar estas aplicações no *browser*.

Para combater o requisito de uso de *plug-ins* em *browsers* para criação de aplicações complexas e interativas, a Mozilla, em 2011, lançou a primeira versão estável de WebGL. WebGL introduziu uma API em JavaScript para a criação de aplicações 2D e 3D interativas

compatíveis com qualquer *browser* sem o uso de *plug-ins*. WebGL conseguiu este feito ao introduzir uma API baseada em OpenGLES que pode ser usada em HTML5.

Desde o aparecimento de WebGL em 2011, este tem vindo a ganhar popularidade para uso em aplicações web. Contudo, WebGL é uma API de muito baixo nível quando comparada com a utilização do Unity. Daí haver a necessidade de introduzir bibliotecas, construidas com base em WebGL, para agilizar o desenvolvimento de aplicações web.

Em 2013, Google anunciou que iria desativar e remover o *plug-in* NPAPI [14] do seu próprio *browser* [16]. Este *plug-in* é essencial para o desenvolvimento de extensões e outros *plug-ins* tais como Unity WebPlayer, Java, Silverlight, entre outros. Com este cenário à frente, a melhor opção para o desenvolvimento destas aplicações passou a ser o uso de WebGL.

2.5.1 Bibliotecas e Plataformas

Com a popularidade de WebGL a aumentar, começaram a ser desenvolvidas bibliotecas que agilizam o processo de criação de aplicações visualmente complexas e que reduzem o fosso entre o nível de abstração da API e o das aplicações.

Não só foram desenvolvidas bibliotecas, como também motores de jogo, tais como Unity e Unreal Engine, acompanhados dos respetivos editores e demais ferramentas possibilitando a exportação dos projetos para plataforma WebGL, conseguindo assim continuar a suportar aplicações Web.

Nesta secção, serão analisadas diversas bibliotecas e programas de desenvolvimento de aplicações para web que melhor apresentam funcionalidades e capacidades para desenvolvimento do projeto proposto para esta tese de mestrado.

2.5.1.1 ThreeJS

ThreeJS [15] foi inicialmente desenvolvido com um foco na criação de uma biblioteca 3D leve, e com um nível de complexidade baixo para abranger o maior número de desenvolvedores possível, oferecendo uma API completa e otimizada, construída em cima de WebGL.

Semelhante a *frameworks* de desenvolvimento de jogos tradicionais, esta biblioteca utiliza os mesmos conceitos e processos para o desenvolvimento de aplicações web.

ThreeJS utiliza uma estrutura de dados Mesh, com uma estrutura e utilização semelhante à implementação realizada pelo Unity, onde vértice poderá ser representado como um pixel no ecrã, conseguindo assim criar nuvens de pontos de grande dimensão. ThreeJS também suporta a utilização de *billboards* [16] , possibilitando a criação de visualizações onde cada estrela tem um tamanho aparente.

[14] https://developer.chrome.com/extensions/npapi
[15] https://github.com/mrdoob/three.js
[16] http://threejs.org/examples/#webgl_custom_attributes_points3

Ao contrário do Unity, ThreeJS desenha estes *billboards* independentemente da distância a que se encontram da câmara, desenhando um *billboard* quando este tem um tamanho no ecrã superior a um pixel, e desenhando um pixel quando o *billboard* é inferior a um pixel. Isto torna a visualização de *billboard* mais eficiente, visto não necessitar de duas estruturas de dados para ter este resultado.

2.5.1.2 BabylonJS

BabylonJS [9] é um motor de jogo 3D baseado em WebGL e Javascript. Este oferece uma API poderosa para desenvolvimento de aplicações interativas na web, oferecendo funcionalidades como motor físico, partículas, efeitos de luz, otimizações, técnicas de síntese de imagem, entre outros.

Esta biblioteca oferece funcionalidades pré-criadas que são úteis para desenvolvimento de aplicações interativas, tais como diversos tipos de câmaras que podem ser criadas e controladas muito facilmente pelo utilizador, otimizações para eliminar processamento desnecessário de grandes quantidades de objetos não visíveis pelas câmaras, bem como implementações otimizadas dos algoritmos que serão usados na visualização dos objetos não descartados pelas operações de *culling*.

2.5.1.3 Web Workers

Uma vez que Javascript é um ambiente *single-thread*, significa que todo o processamento é efetuado apenas por um processador, o que conduz à deterioração da interface de utilizador, ou UI. Não só isto afeta a UI, que impossibilita o utilizador de realizar tarefas de grande escala em tempo-real, faz com que a comunicação com o servidor de objetos tenha de ser síncrona, estando as restantes tarefas paradas quando existe comunicação entre o cliente e servidor.

Para tornar este tipo de processamento e comunicação possíveis, poderão ser introduzidas Web Workers [17] [12], permitindo lançar vários Web Workers que comunicam através de mensagens no contexto da *thread* de UI, que é a *thread* principal. Web Workers criam *scripts* que correm em *background* na aplicação, que podem ser usadas para computação de tarefas de longo tempo sem bloquear a *thread* de UI ou qualquer outro *script* que trata do input do utilizador [1].

2.5.1.4 Unity

Recentemente, Unity anunciou o lançamento completo da plataforma WebGL [4], sendo agora uma plataforma oficialmente suportada por Unity, trazendo um grande número de melhoramentos para esta plataforma, como muitas otimizações e novas funcionalidades.

[17] https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API/Using_web_workers

Ao contrário de bibliotecas JavaScript que usam WebGL, Unity dispõe de um editor e da linguagem C# .NET para desenvolvimento de aplicações interativas, conseguindo ter um desenvolvimento de aplicações mais eficiente e estruturado.

Apesar de Unity suportar esta plataforma, existem ainda diversos problemas que dificultam o desenvolvimento de aplicações muito complexas nesta plataforma, não suportando *multi-threading* e existindo limitações na capacidade de processamento gráfico [18].

Destas limitações, *multi-threading* é a limitação mais preocupante, visto que no projeto IVELA, a aplicação comunica com o servidor de objetos via *multi-threading*, não deixando a aplicação e o utilizador parados à espera da resposta e informação vinda do servidor, pelo que a única opção para uma aplicação Web deste projeto construída com o Unity seria utilizar comunicação síncrona entre cliente e servidor.

2.5.1.5 Unreal Engine

Unreal Engine [19] é um motor de jogo para desenvolvimento de jogos e aplicações interativas. Tal como o Unity, estes dois motores de jogo são os motores de jogo públicos mais populares da atualidade para desenvolvimento de jogos, existindo uma grande competição entre os dois com o objetivo de fornecer as melhores ferramentas aos desenvolvedores de jogos.

Com a popularidade de WebGL a aumentar, e a adoção de WebGL no motor de jogo Unity, Unreal Engine incorporou a habilidade de exportar as aplicações para Web usando HTML5 e WebGL [10].

Semelhante a Unity, Unreal Engine oferece um editor e ferramentas para desenvolvimento destas aplicações, linguagem C++ com grande liberdade no que o desenvolvedor consegue criar, um grande número de otimizações para processamento gráfico de um número muito elevado de objetos, entre outros.

2.6 Trabalhos Relacionados

Neste capitulo serão analisadas algumas ferramentas usadas pelos astrónomos para análise de catálogos astronómicos, bem como projetos e aplicações desenvolvidas para o publico geral com o objetivo de explorar dados de astronomia.

2.6.1 TopCat

TopCat [20] [17] é o programa principal usado pelos astrónomos para visualização e análise de catálogos de estrelas. Este programa oferece uma ferramenta de interação visual

[18] http://docs.unity3d.com/530/Documentation/Manual/webgl-gettingstarted.html
[19] https://www.unrealengine.com
[20] http://www.star.bris.ac.uk/ mbt/topcat

de dados, facilitando a análise e manipulação de catálogos. É uma aplicação *standalone* escrita usando a linguagem Java.

Esta ferramenta especializa-se na capacidade de processamento de catálogos com milhões de dados, conseguindo criar visualizações destes dados de uma maneira interativa para uma análise eficiente dos dados, oferecendo ferramentas multi-vistas sincronizadas com diversos tipos de visualização de dados.

Apesar de suportar visualizações 3D, como pode ser visualizada na figura 2.9, este tipo de visualização não tem a capacidade de suportar a grande quantidade do catálogo Gaia de maneira interativa.

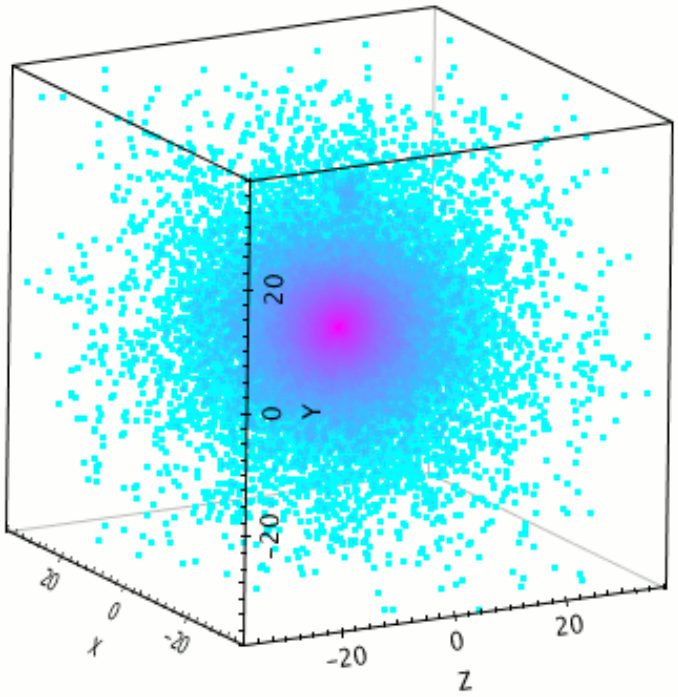

Figura 2.9: Visualização de uma nuvem de pontos em 3D na aplicação TopCat

2.6.2 VaeX

VaeX [21] é uma ferramenta gráfica para visualização de um grande volume de dados de catálogos em ambiente *desktop*.

Esta ferramenta foca-se na criação de visualizações de histogramas, *density-plots* e processamento gráfico de volumes 3D de uma quantidade de dados na ordem dos 10 mil milhões de dados em cerca de 1 segundo. Para exploração destes dados, VaeX oferece ferramentas de seleção em visualizações 1D e 2D, conseguindo observar estas seleções em visualizações multi-vistas.

[21] http://www.astro.rug.nl/ breddels/vaex

Apesar da capacidade de processamento de uma quantidade enorme de informação, atualmente, VaeX apenas suporta criação de volumes para ambientes 3D, observado na figura 2.10, não suportando scatter-plots em 3D.

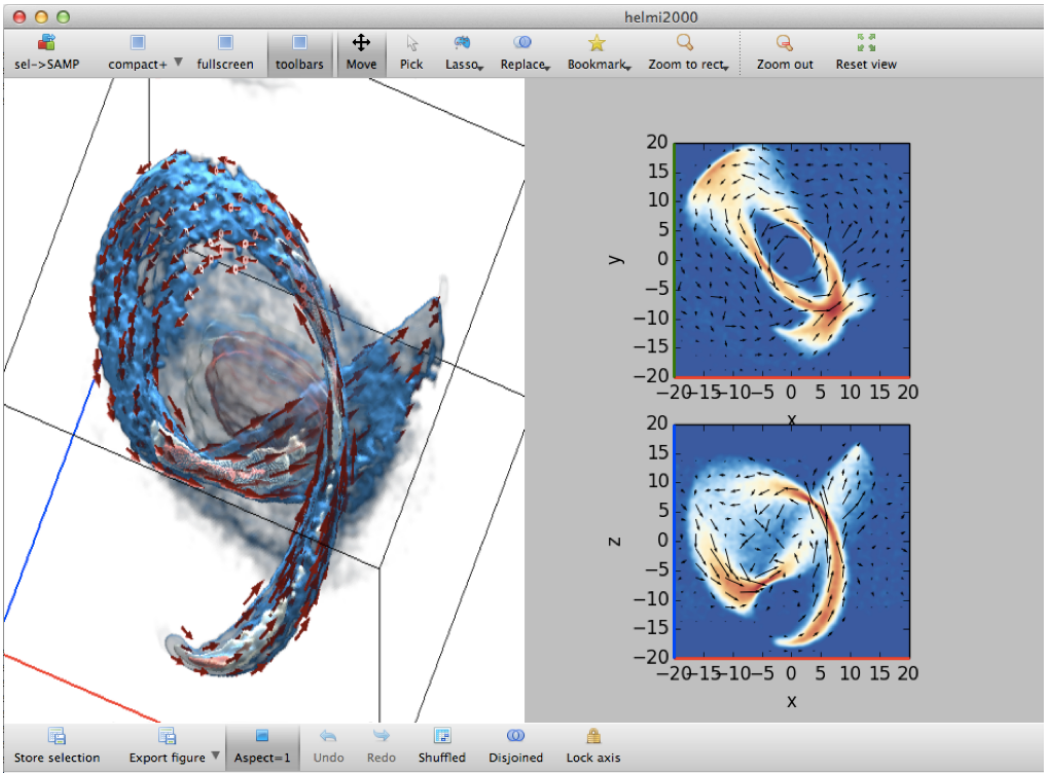

Figura 2.10: Visualização 3D volumétrica de um catálogo de estrelas na aplicação VaeX

2.6.3 Cosmography of OB Stars In The Solar Neighborhood

Esta aplicação [22] [2] constrói um mapa 3D da densidade especial das estrelas da vizinhança do nosso Sol usando o catalogo Hipparcos, como pode ser visto na figura 2.11.

Nesta ferramenta, são oferecidas funcionalidades para controlar o posicionamento da câmara para observar os dados de vários ângulos, bem como uma interface que possibilita ativar e desativar elementos da visualização de acordo com o que o utilizador quer visualizar.

2.6.4 Gaia Sandbox

Gaia Sandbox [23] é uma aplicação *standalone* que oferece uma visualização 3D em tempo real das estrelas da nossa galáxia. O utilizador pode navegar pela galáxia, sistema solar, e observar o satélite Gaia em movimento na sua orbita, visualizando como este recolhe informação sobre as estrelas vistas a partir do seu plano de referencia e ângulo de visão.

[22] http://sci.esa.int/hipparcos/ob-stars-interactive
[23] https://zah.uni-heidelberg.de/gaia2/outreach/gaiasandbox

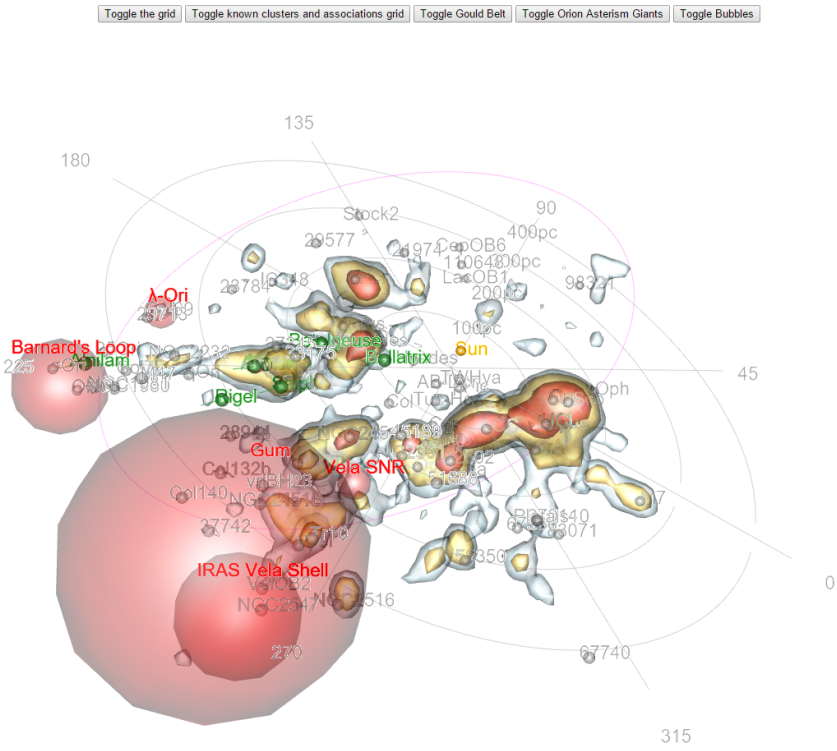

Figura 2.11: Ferramenta para visualizar as estrelas vizinhas do nosso Sol

Esta aplicação oferece outros tipos de funcionalidades, tais como suporte para vista estereoscópica, uso dos dados locais do utilizador, gravar o caminho realizado pela câmara e acrescentar novas funcionalidades à aplicação ao deixar o utilizador escrever código na linguagem de programação Python que pode ser depois incorporada na aplicação.

Para visualizar a galáxia, esta aplicação usa uma imagem de uma outra galáxia para conseguir ter uma representação semelhante à nossa galáxia, usando em conjunto com o catálogo Hipparcos, representando as estrelas mais próximas do nosso Sol. Uma representação da nossa galáxia usando esta aplicação pode ser observada na figura 2.12.

2.6.5 100,000 Stars

100,000 Stars [24] é uma aplicação que utiliza WebGL para criar uma visualização 3D interativa da nossa galáxia. Para isto, a aplicação utiliza as estrelas vizinhas ao nosso Sol a partir do catálogo de cerca de 120 mil estrelas Hipparcos.

Esta aplicação mostra ao utilizador a escala da nossa galáxia ao deixar o utilizador explorar as estrelas vizinhas ao nosso Sol, e fazer *zooming* até conseguir visualizar toda a nossa galáxia, como é vista por um observador externo [3].

Esta visualização é efetuada ao utilizar uma outra galáxia [5], semelhante à nossa, como base para a estrutura da galáxia. Como pode ser esperado, esta visualização não

[24] http://stars.chromeexperiments.com

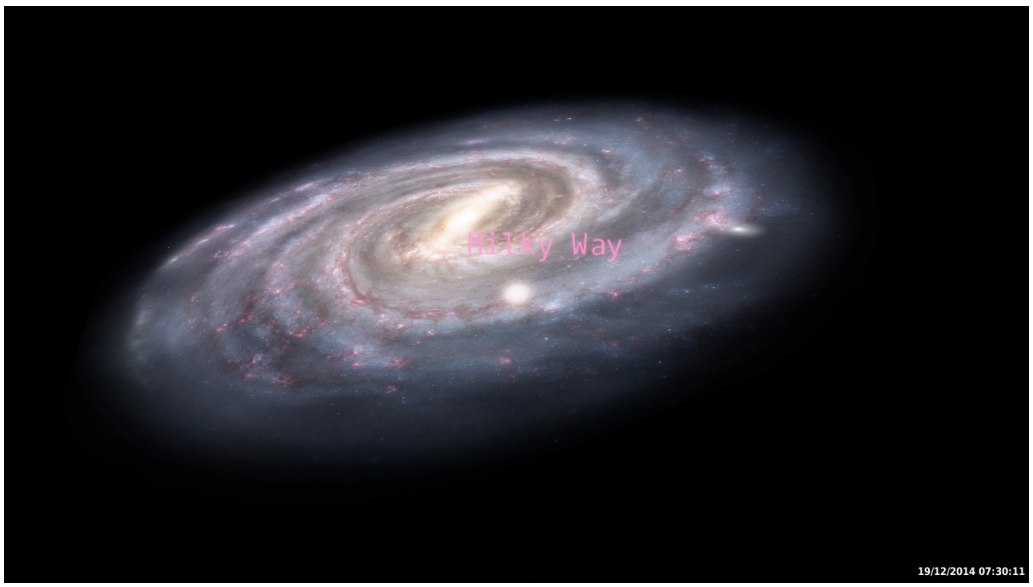

Figura 2.12: Aplicação Standalone Gaia Sandbox, mostrando uma representação na nossa galáxia

é correta, pois apesar de utilizar apenas uma fração de estrelas da nossa galáxia para representar a vizinhança do nosso Sol, são usadas imagens de uma outra galáxia para criar a uma representação da nossa galáxia, como é visto na figura 2.13.

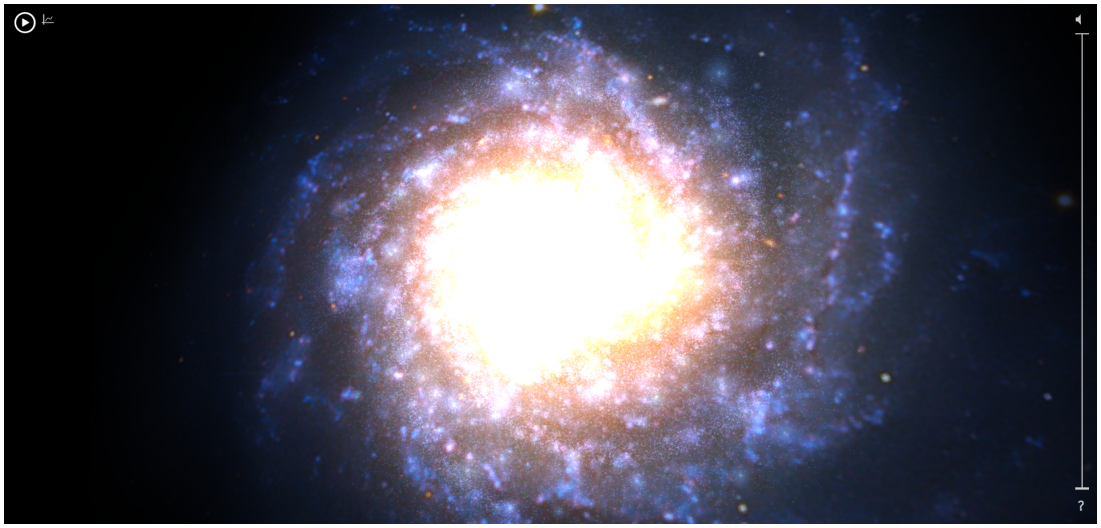

Figura 2.13: Aplicação web 100,000 Stars, mostrando uma representação da nossa galáxia

Capítulo 3

Abordagem

Neste capitulo de abordagem, serão identificados os diversos requisitos, bem como as dificuldades que estes requisitos acrescentam para a concretização deste sistema.

Fazendo parte do projeto GAVIDAV, este sistema tem como requisito a utilização de *browsers* para oferecer uma visualização 3D interativa para o grande público. Ao utilizar *browsers*, será necessário utilizar uma biblioteca 3D que possibilite a criação de ambientes em 3 dimensões de forma eficiente. Esta biblioteca terá de oferecer funcionalidades que facilitem a construção de visualizações 3D interativas para um grande volume de dados.

Sabendo que o catálogo de estrelas fornecido pela missão GAIA é de grandes dimensões, este sistema tem também como requisito a utilização do Servidor de Objetos, oferecido pelo projeto GAVIDAV, para aceder e recolher estes dados de forma inteligente. Utilizando a arquitetura e funcionalidades oferecidas pelo Servidor de Objetos, o sistema deverá requisitar o menor número de dados possível que consiga representar os dados que o utilizador queira visualizar.

Ao utilizar catálogos de estrelas desta dimensão, não será possível descarregar todo o catálogo diretamente na memória das máquinas dos utilizadores para ser usado na visualização. Assim, este sistema deverá realizar uma gestão eficiente destes dados recebidos, de forma a que não exceda as capacidades destas máquinas.

Adicionalmente, há de existir um mecanismo de controlo, no qual o sistema deixará o utilizador explorar o catálogo de estrelas dentro da visualização 3D. Estes controlos deverão ser intuitivos e de fácil utilização para navegar dentro dos dados.

3.1 Visão Geral

Sabendo os requisitos e necessidades deste sistema, é possível separar os diversos mecanismos por várias componentes, onde cada uma terá um objetivo concreto e irá interagir

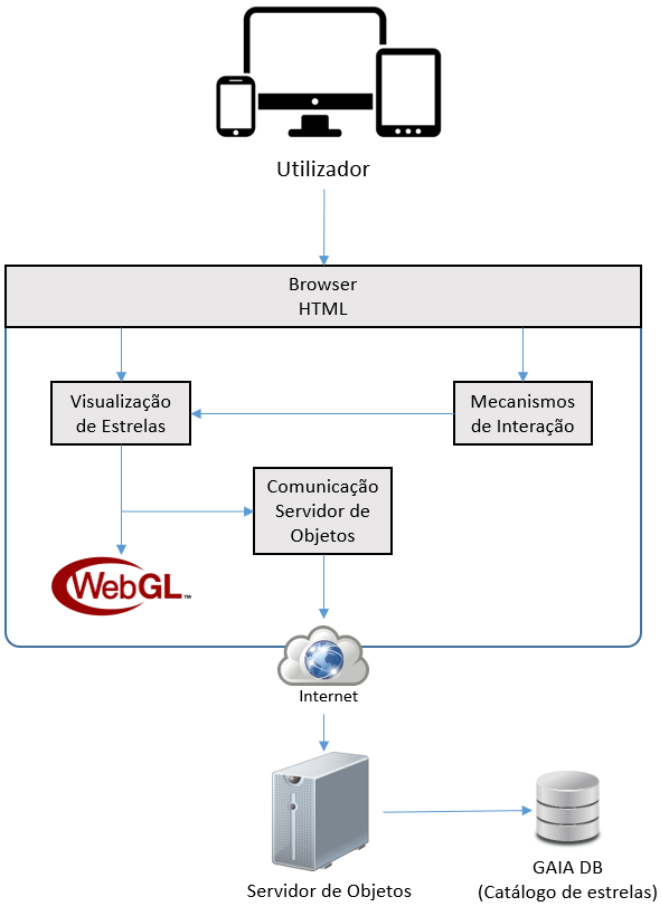

Figura 3.1: Diagrama da abordagem da arquitetura

com outros componentes para troca de informação, como pode ser observado na figura 3.1

Esta arquitetura contém três componentes principais, que em conjunto possibilitarão a criação de uma visualização 3D no *browser* para catálogos de estrelas de grandes dimensões.

Para o utilizador poder comunicar com o sistema, o input do utilizador será convertido de modo a que este seja compatível com os mecanismos de controlo oferecidos pelo sistema.

A componente de Mecanismos de Interação implementará todo o tipo de controlo que o utilizador poderá utilizar para navegar e explorar os dados em 3D. Para controlar a câmara de visualização, esta terá de aceder à componente de Visualização de Estrelas.

Ao utilizar uma biblioteca 3D no *browser*, a componente de Visualização de Estrelas criará o ambiente que irá conter o catálogo de estrelas, atualizando-o com novos dados à medida que o utilizador interage com a visualização.

Para aceder aos dados do GAIA, esta componente interage com a componente de Comunicação Servidor de Objetos, a qual terá como objetivo comunicar, pela Internet, com

o servidor de objetos para recolher os dados do GAIA. Após receber os dados necessários, esta componente devolve os dados processados à componente de Visualização de Estrelas, atualizando assim as estrelas que estão visíveis no ambiente 3D.

3.2 Opções Tecnológicas

De seguida, serão descritas as diferentes opções tecnológicas nas quais o sistema se irá basear.

Será definida a plataforma hardware no qual o protótipo se irá focar, a API 3D que será utilizada para produzir as visualizações detalhadas dos objetos em 3D, através do *browser*, e de que forma será o enorme volume de estrelas fornecido a este sistema, tentando minimizar problemas de performance.

3.2.1 Dispositivos

Atualmente há uma grande variedade de dispositivos disponíveis, desde *desktop*, *tablets* e *smartphones*, sendo todos eles capazes de produzir visualizações 3D interativas para uso pelo utilizador comum. Devido às propriedades físicas dos *smartphones*, não seria prático visualizar o enorme catálogo de estrelas GAIA num dispositivo de tão pequenas dimensões, pelo que mais facilmente se prevê as pessoas explorarem estes dados através de um *tablet* ou *desktop*.

Adicionalmente, existe uma convergência de hardware entre os *tablets* e *desktops*, embora não haja uma convergência de software base ou da forma como as pessoas interagem através das interfaces de cada dispositivo.

Sendo o ambiente *desktop* mais apropriado para desenvolvimento, este será utilizado para implementar e avaliar o prototipo funcional. No entanto, os controlos desenvolvidos para este ambiente não controlarão explicitamente as ferramentas de exploração destas visualizações, pelo que o sistema terá em conta de que o que for desenvolvido poderá ser estendido para outros dispositivos, como *tablet*.

3.2.2 WebGL

Para efeitos de visualização de um ambiente 3D num *browser* utilizando um grande volume de dados, será utilizado WebGL.

WebGL fornece uma API JavaScript especializada em processamento gráfico de gráficos 3D interativos dentro de *browsers*. Esta tira proveito das capacidades das placas gráficas para produzir ambientes extremamente complexos e detalhados de uma forma interativa para o utilizador.

Para além de WebGL ser suportado pela maioria dos *browsers* disponíveis, este também é suportado pela maioria dos dispositivos existentes, tais como em ambiente *desktop* (Windows, Mac e Linux) bem como em ambientes *tablet* e ainda *smartphones* (iOS e Android).

WebGL fornece uma API de baixo nível, não sendo muito adequada para usar de forma direta pelas aplicações. Felizmente estão disponíveis bibliotecas que agilizam o processo de criação de aplicações visualmente complexas, reduzindo o nível de abstração entre a API e a aplicação.

Para aumentar o nível de abstração do sistema, é importante que o processo de criação de visualizações 3D de estrelas não dependa de uma biblioteca em concreto, sendo desejável que as visualizações possam ser criadas utilizando qualquer uma das bibliotecas de alto nível construidas com recurso a WebGL.

Para efeitos de implementação deste sistema, será utilizada a biblioteca ThreeJS para criação destas visualizações. Esta biblioteca oferece uma *framework* de desenvolvimento de aplicações 3D que garante todas as funcionalidades necessárias para criação de estrelas em ambientes 3D.

3.2.3 Servidor de Objetos

O catálogo de estrelas GAIA tem uma quantidade de estrelas tão elevada que se torna impraticável transmitir e guardar localmente toda a informação necessária para produzir uma visualização destas estrelas.

Para possibilitar a visualização deste catálogo nos dispositivos dos utilizadores, estes dados são simplificados e transmitidos em pequenos pacotes através de um servidor que contém o catálogo de estrelas. O cliente interage com o servidor de objetos, através da sua API, para requisitar os dados necessários para a criação e visualização do catálogo de estrelas.

Ao recriar as estruturas de dados do servidor no cliente, podem ser calculados quais dados que o cliente necessita para criar a visualização, pedindo ao servidor de objetos apenas os dados necessários para essa visualização.

3.3 Resultados

Esta dissertação tem como foco propor uma arquitetura que irá possibilitar a visualização 3D de catálogos de estrelas de grandes dimensões para o grande publico em geral, utilizando um servidor de objetos remoto para disponibilização destes dados de forma inteligente.

Será realizada uma implementação de um protótipo experimental com base na arquitetura proposta, bem como uma avaliação pormenorizada do seu comportamento e desempenho.

Sabendo que esta implementação tem como foco o publico geral, grande parte destes utilizadores não tem conhecimento sobre a informação ou características deste catálogo de estrelas. Assim, serão desenvolvidas narrativas, nas quais o protótipo utilizará informação pré-definida para mostrar características e informação sobre o catálogo de estrelas.

3.3.1 Arquitetura

Será proposta uma arquitetura capaz de produzir aplicações de visualizações de estrelas em *browsers*, que será capaz de se adaptar às diferentes configurações de hardware dos dispositivos.

Dada a imensa quantidade de estrelas existentes no catálogo GAIA, o servidor de objetos irá dar suporte à entrega de conjuntos de pequenas dimensões de dados, quando necessário, e irá fornecer todas as informações pertencentes ao catálogo de estrelas em utilização.

Esta arquitetura deverá gerir todo o processo de *caching* dos dados recebidos do servidor de objetos, de modo a que o sistema minimize a necessidade de requisitar informação sobre conjuntos de estrelas e utilize de forma adequada a memória disponível no dispositivo.

Adicionalmente, deverá ser utilizada uma API 3D para criar as visualizações 3D interativas do catálogo de estrelas e dispor de mecanismos de interação dentro do ambiente 3D de modo a possibilitar a exploração destes dados.

3.3.2 Implementação

Neste protótipo serão implementados mecanismos e funcionalidades concretas que irão possibilitar a comunicação entre o servidor de objetos e o sistema, tratarão do processamento e armazenamento da informação recolhida do servidor de objetos para criação de visualizações 3D. Será também implementado um sistema de gestão de recursos da informação das estrelas e mecanismos de interação para exploração destes dados.

3.3.3 Avaliação

Com vista a uma avaliação experimental do sistema, serão realizados testes para diferentes configurações de hardware deste protótipo focando em aspetos de desempenho e memória.

O desempenho será medido com base na capacidade de o protótipo produzir uma visualização 3D interativa para o utilizador em tempo real. O protótipo deverá manter uma taxa de refrescamento aceitável ao longo da interação e exploração realizada pelo utilizador.

Não sendo possível medir a utilização de memória usada pelo protótipo a partir de *browsers*, a gestão de *caching* será realizada ao limitar a quantidade de informação que será guardada em memória, neste caso o número de estrelas.

Serão avaliadas diversas combinações destes parâmetros, medindo os seus impactos na quantidade de memória utilizada pelo protótipo.

3.3.4 Narrativa

Para além de oferecer mecanismos de interação para explorar os catálogos de estrelas em tempo real, serão desenvolvidas diversas narrativas que irão controlar a câmara dentro da visualização para mostrar diferentes características e informação sobre o catálogo de estrelas.

As narrativas serão definidas com base num conjunto de posições 3d que estabelecem um caminho, um conjunto de rotações que definem a orientação da câmara e um conjunto de informações ou características físicas sobre o catálogo de estrelas.

Estas narrativas serão pré-definidas e estarão disponíveis para os utilizadores poderem escolher e observar.

Capítulo 4

Arquitetura

4.1 Introdução

Neste capitulo será definida a arquitetura do sistema, sendo descritos os diversos componentes que a integram.

Cada componente será descrito e analisado individualmente na sua própria secção, onde será feita uma introdução sobre o papel deste componente dentro da arquitetura, e de que forma esta componente será utilizado por outras componentes, fornecendo uma API que permite a troca de informação.

4.2 Arquitetura do Sistema

A arquitetura será definida pelo diagrama na figura 4.1, onde se visualiza o papel dos diversos componentes, e como estes interagem entre si de forma a trocar informação necessária para criar e atualizar a visualização 3D interativa.

Para construir uma arquitetura que possibilite a visualização 3D de estrelas, compatível com o servidor de objetos, serão necessários dois conjuntos de componentes principais: Um conjunto de componentes que consiga recolher e preparar os dados do catalogo de estrelas localizados no servidor externo, e um conjunto de componentes que utilize esses dados para construir uma visualização interativa no *browser* do utilizador.

O primeiro conjunto de componentes, que contém a componente Wrapper de comunicação entre cliente e servidor e a componente proxy de servidor de objetos, tem como objetivo realizar toda a comunicação e preparação dos dados entre o servidor de objetos e a visualização 3D. Todos os dados que a visualização 3D necessite terão de ser fornecidos por estes componentes.

Para preparar os dados do catalogo de estrelas para a visualização 3D, esta arquitetura

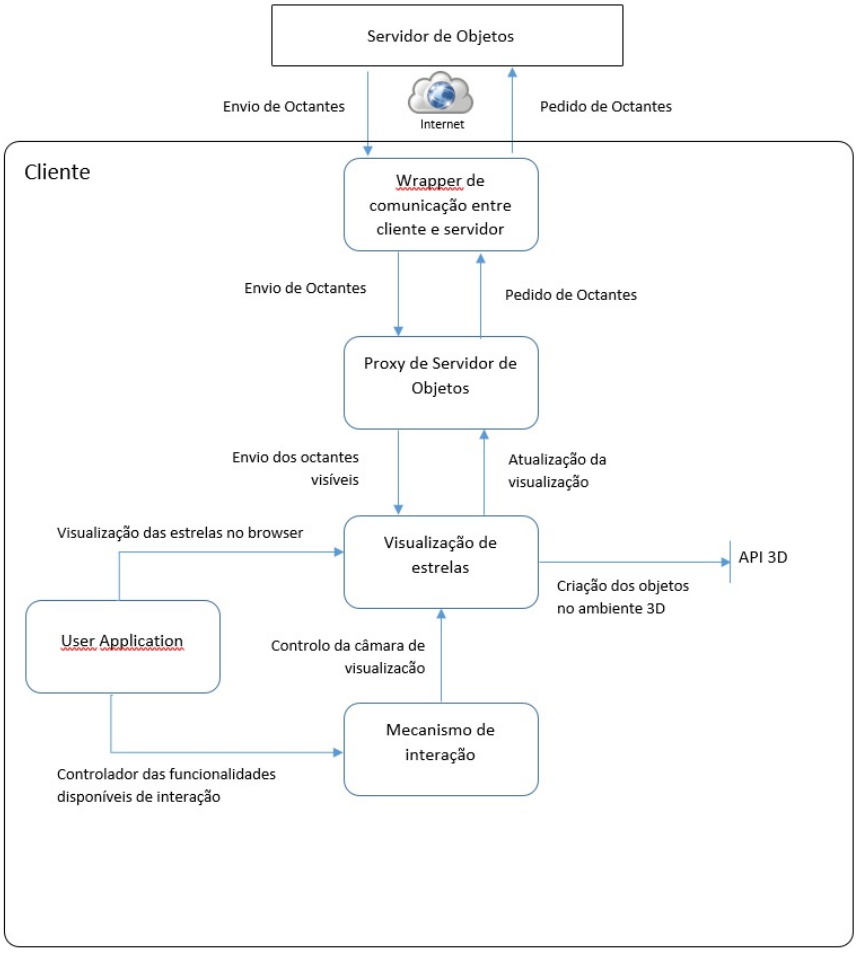

Figura 4.1: Diagrama da arquitetura proposta para a solução do problema

necessita de guardar localmente uma representação da estrutura de dados *octree* pré-processada pelo servidor de objetos, e utilizará esta estrutura de dados para calcular que dados devem ser requisitados para gerar a visualização 3D.

Sabendo que é necessário realizar um protocolo de comunicação entre o cliente e servidor, necessitaremos de abstrair toda esta comunicação da restante arquitetura, expondo uma API com funcionalidades que serão utilizadas pelos outros componentes da arquitetura.

Assim, estes componentes estarão subdivididos em duas partes: Componente Wrapper de Comunicação entre cliente e servidor, com o objetivo de abstrair todo o protocolo de comunicação que existirá entre o cliente e o servidor; e a componente Servidor de Objetos Proxy, a qual utilizará a componente de comunicação para requisitar os dados de estrelas do catálogo e construir todas as estruturas de dados locais no cliente, expondo uma API que as componentes de visualização poderão utilizar para gerir as visualizações.

O segundo conjunto de componentes, constituído pela componente visualização de estrelas, mecanismo de interação e user application, tem o papel de criar uma visualização 3D interativa de estrelas utilizando os dados recolhidos pela componente Servidor de

Objetos Proxy.

Para criar uma visualização interativa, é vantajoso separar a secção que gera a visualização 3D da secção que interage com essa visualização. Ao separar estas duas componentes, simplifica-se a adição ou modificação de mecanismos de interação, podendo no futuro adicionar novos tipos de dispositivos que controlem a visualização de diferentes maneiras, utilizando diferentes tipos de Input.

A componente de visualização de estrelas irá interagir com a componente Servidor de Objetos Proxy para requisitar conjuntos de estrelas que devem ser incluídas no ambiente 3D. Esta componente utilizará uma biblioteca que possibilita a criação de ambientes tridimensionais diretamente no *browser*. Adicionalmente, será exposta uma API que será utilizada pela componente de interação, no qual será usado o input do utilizador para controlar a câmara dentro da visualização.

Todos estes componentes interagem entre si para criar uma visualização 3D interativa de grandes quantidade de estrelas no *browser* de um utilizador, como pode ser observado pelo diagrama 4.1. A seguinte lista elabora o papel de cada um dos componentes:

- Wrapper de Comunicação entre Cliente e Servidor

  Esta componente tem como objetivo abstrair o protocolo de comunicação entre o cliente e o servidor, encapsulando todos os aspetos da comunicação, oferecendo uma API de interação entre servidor e o sistema.

- Proxy Servidor de Objetos

  Esta componente irá assegurar que todos os dados recolhidos do servidor estejam acessíveis para visualização. Este irá requisitar os octantes que estão atualmente visíveis na visualização, e irá entregar os octantes recebidos do servidor à componente de visualização de estrelas, que irá atualizar a visualização com estes novos dados.

  Adicionalmente, esta componente irá possuir um mecanismo de *caching* que deverá realizar a manutenção dos dados recebidos do Servidor de Objetos de tal forma que os dados guardados em memória não excedam as capacidades do dispositivo utilizado para explorar a visualização.

- Visualização de Estrelas

  A componente de visualização de estrelas irá criar as estruturas de dados que permitirão usadas para gerar as visualizações num ambiente 3D, utilizando a API 3D adotada. Este componente irá requisitar novos octantes ao servidor proxy sempre que o volume de visão seja alterado pela interação realizada pelo utilizador.

- Mecanismo de Interação

  Nesta componente, serão implementados mecanismos que irão interagir com a visualização de estrelas, podendo realizar ações de *panning*, *zooming*, rotação e movimento da câmara.

- User Application

  Para o utilizador poder controlar os mecanismos de interação, esta componente utilizará o input da plataforma usada e irá interagir com a componente de Mecanismos de Interação com este input.

Nas seguintes secções, cada uma das componentes será analisada em detalhe.

4.3 Wrapper de Comunicação Cliente-Servidor

4.3.1 Introdução

Para interagir com o Servidor de Objetos, esta componente implementará todo o protocolo de comunicação entre o cliente e o servidor, expondo às restantes componentes da arquitetura uma API das funcionalidades necessárias para a criação e atualização da visualização.

Abstraindo o protocolo de comunicação entre o cliente e o servidor, torna-se possível no futuro alterar este protocolo para utilizar outro tipo de servidor, sem ter a necessidade de alterar a restante arquitetura. Isto desde que os dados comunicados à restante arquitetura tenham o mesmo formato.

Para satisfazer as necessidades para visualização de catálogos de estrelas nesta arquitetura, a componente de Wrapper de Comunicação Cliente-Servidor deverá cumprir um conjunto de requisitos divididos nas seguintes categorias:

- Conexão

  Esta componente deverá estabelecer e manter uma conexão com o servidor de objetos.

- Acesso aos dados do catálogo

  Deverá ser possível requisitar a informação sobre o catálogo de estrelas que será utilizado para a visualização 3D.

- Acesso aos metadados

  Para poder construir uma visualização 3D do catálogo de estrelas, deverá ser possível pedir a informação sobre a *octree* pré-processada pelo servidor de objetos e deverá ser possível requisitar octantes específicos pertencentes a essa *octree*.

4.3.2 Proposta de API

O servidor de objetos disponibiliza uma API rica em funcionalidades, dando acesso aos dados pré-processados e não processados, bem como outros tipos de funcionalidades. No âmbito desta arquitetura, onde serão construidas visualizações 3D de estrelas, apenas serão utilizadas as funcionalidades necessárias para a construção e atualização da visualização 3D.

Assim a API será definida com base nas seguintes funcionalidades oferecidas pelo servidor de objetos:

- Login(username, password)

  Será possível conectar e realizar *login* com o servidor de objetos utilizando o *username* e *password* de um utilizador

- Disconnect()

  Será fornecido um comando para terminar a conexão com o servidor de objetos.

- GetVisualizationInfo()

  Este método retorna um objeto em formato JSON que irá conter toda a informação sobre o catálogo de estrelas.

- GetOctreeMetadata()

  Para poder requisitar informação sobre pedaços do catálogo, primeiro será necessário requisitar a informação metadata da *octree* localizada no servidor de objetos, especificando como esta está estruturada.

- GetOctants(octants)

  Este método irá requisitar um conjunto de octantes pertencentes a uma visualização especificada. A resposta a este pedido irá conter uma lista de conjuntos de estrelas pertencentes a cada um dos octantes requisitados, num formato JSON.

4.4 Proxy de Servidor de Objetos

4.4.1 Introdução

A componente Proxy de Servidor de Objetos utilizará a componente Wrapper para requisitar todos os dados das estrelas necessários para construir a estrutura de dados local e fornecer os conjuntos de estrelas à componente de visualização.

A estrutura de dados que será construída localmente é uma estrutura do tipo *octree*. Esta *octree* irá conter um conjunto de estrelas por cada octante, sendo cada octante identificado por um número único o qual será utilizado para requisitar ao servidor de objetos as estrelas pertencentes a este octante.

Sabendo que o catálogo de estrelas tem grandes dimensões, não é possível requisitar todos os octantes simultaneamente ao servidor de objetos, devido a limitações do hardware comum atual, pelo que se torna necessário requisitar apenas octantes que sejam visíveis pela câmara em cada momento, reduzindo a quantidade de informação em memória para a gerar a visualização.

Esta componente necessitará de utilizar o volume de visão da câmara, e percorrer a *octree* para determinar todos os octantes que intersetam esse volume de visão. Devido

à natureza da projeção perspetiva da câmara num ambiente 3D, quanto mais longe as estrelas se encontram da câmara, mais juntas elas ficam umas das outras, ocupando uma área menor na visualização.

Devido a este fenómeno, podemos reduzir o número de octantes que necessitamos para criar a visualização, mostrando níveis de detalhe reduzidos para regiões da *octree* que se encontrem a uma maior distância da câmara.

Ao encontrar todos os octantes que estão visíveis, esta componente utiliza a API da componente Wrapper para requisitar os octantes calculados, fornecendo-os à componente de visualização quando o pedido termina.

Não sendo possível guardar localmente todos os octantes que são recebidos do servidor, devido a limitações de hardware, torna-se necessário realizar uma manutenção destes octantes, efetuando *caching* de octantes e descartar os octantes que terão menos probabilidade de serem visualizados pela visualização.

A capacidade de realizar *caching* de octantes torna a interação entre componentes mais eficiente, não necessitando de requisitar constantemente a mesma informação sobre conjuntos de estrelas ao servidor de objetos.

4.4.2 Caching

A realização de *caching* dos octantes recebidos do servidor de objetos reduz significativamente o tempo de espera destes pedidos quando os octantes já se encontram localmente na máquina do utilizador. Enquanto que os pedidos iniciais destes octantes têm um tempo de espera significativo, necessitando de realizar carregamento de dezenas ou centenas de *MegaBytes* de informação pela Internet, os pedidos que envolvem octantes já carregados em memória são efetuados instantaneamente.

Apesar de a realização de *caching* ser crucial para atualizar a visualização 3D de forma eficiente, se o catálogo de estrelas tiver uma dimensão muito elevada, poderá não existir memória suficiente no dispositivo para guardar todos estes octantes simultaneamente, pelo que deverão existir formas de manter octantes importantes e descartar octantes menos importantes de memória.

Sabendo que é necessário realizar manutenção destes octantes, será definido um critério para identificar a importância de cada octante para a visualização atual. Ao atribuir um valor de importância para cada octante, é possível descartar aqueles que terão menos impacto no futuro da visualização.

Este critério será definido com base no tempo sem utilização na visualização por parte de cada octante. Se um octante se encontra muito tempo sem ser utilizado, significa que a sua importância para o futuro da visualização 3D é menor em comparação com outros octantes que tenham sido utilizados há menor tempo que esse, tendo assim um impacto menor na visualização se for removido.

Assim, os octantes que são utilizados regularmente terão mais probabilidade de serem mantidos em memória, e octantes que são raramente utilizados serão descartados.

4.4.3 Proposta de API

Esta componente irá expor uma API que será utilizada por duas componentes da arquitetura.

A componente Visualização de Estrelas utilizará esta API para requisitar conjuntos de octantes para serem utilizados na visualização, e a componente Wrapper de Comunicação Cliente e Servidor utilizará esta API para transmitir os octantes de estrelas recebidos pelo Servidor de Objetos.

Os métodos expostos na API desta componente serão definidos na seguinte lista:

- UpdateVisibleOctree(Frustum)

  Quando a componente Visualização de Estrelas invoca este método, serão calculados os octantes que estão atualmente visíveis com base no volume de visão da câmara. Os octantes visíveis e existentes em *cache* são transmitidos de volta à componente de visualização de estrelas, onde são introduzidos na visualização 3D instantaneamente, e os restantes octantes visíveis são requisitados ao servidor de objetos através da API da componente Wrapper.

- CacheVisibleOctants(Octants)

  Quando novas estrelas dos octantes requisitados ao servidor de objetos são recebidas pela componente Wrapper de Comunicação entre Cliente Servidor, esta invoca este método com essa nova lista de octantes para atualizar o sistema de *caching*.

4.5 Visualização de Estrelas

4.5.1 Introdução

A componente de visualização de estrelas é responsável pela criação do ambiente 3D interativo dentro do *browser*.

Esta componente irá interagir com a componente Proxy de Servidor de Objetos com o objetivo de requisitar os octantes que estão atualmente visíveis pela câmara. Esta interação é realizada cada vez que o volume de visão da câmara é alterado, invocando o método exposto pela API dessa componente.

Ao receber os novos octantes visíveis, esta componente deverá substituir os octantes antigos com estes novos octantes dentro da visualização, repetindo o processo de novo quando a câmara é afetada e o volume de visão muda.

4.5.2 Proposta de API

A componente de visualização de estrelas irá fornecer uma API para ser acedida pela componente de Mecanismos de Interação e componente Proxy Servidor de Objetos:

- UpdateCamera(Position, Orientation)

Este método irá atualizar a câmara de visualização com novas coordenadas e uma nova orientação. Ao realizar esta ação, a componente de visualização deverá calcular novamente os octantes que se encontram visíveis no novo volume de visão, iniciando assim o processo de atualização da visualização.

- UpdateVisualizationWithVisibleOctants(Octants)

  Para atualizar a visualização com novos octantes vindos do servidor de objetos, a componente Wrapper de Servidor de Objetos invoca este método com uma lista de octantes. A componente de visualização de estrelas utilizará esta lista de octantes para atualizar a visualização 3D com estes novos dados.

4.6 Mecanismos de Interação

4.6.1 Introdução

Esta componente tem como objetivo fornecer um mecanismo de controlo ao utilizador de modo a que este consiga interagir com a visualização 3D.

Ao controlar a câmara fornecida pela componente Visualização de Estrelas, este dá a possibilidade ao utilizador de conseguir explorar os dados do catálogo de estrelas diretamente no seu *browser*.

Assim, existirá uma ligação entre este componente e a componente de visualização, onde esta fornecerá uma interface que possibilita alterar quer a posição quer a orientação da câmara com base em input realizado pelo utilizador. Este input é fornecido através da componente User Application, sendo processado dependendo da plataforma usada e utilizado de forma genérica dentro deste componente.

O tipo de controlo de câmara definido para este componente é com base em controlo de câmara orbital, semelhante à ilustração [1] 4.2. Este tipo de controlo utiliza uma posição 3D, chamada *pivot*, ficando a câmara a orbitar em torno desse ponto, estando sempre apontada para ele.

O controlo orbital de câmara possui três parâmetros: o primeiro define a posição do ponto 3D central; o segundo define a distância a que a câmara se encontra deste ponto (raio da orbita); o terceiro define a orientação da câmara em torno desse ponto. Sabendo estes parâmetros, é possível posicionar a câmara no ambiente 3D.

Adicionalmente, este componente deverá ainda fornecer um mecanismo que, com base num conjunto de posições e orientações que define um caminho em 3D, a componente controlará a câmara de modo a que a câmara percorra esse caminho.

Este caminho é definido por conjuntos de pares de valores, onde um valor define a posição da câmara, e o outro define a rotação da câmara nessa posição. A câmara deverá iterar e realizar o percurso definido por este conjunto de valores.

[1] http://stackoverflow.com/a/11512384

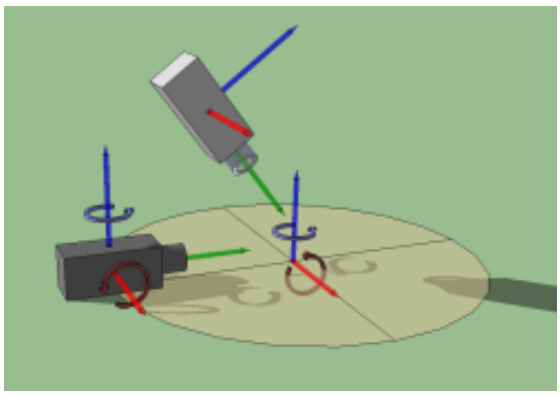

Figura 4.2: Demonstração de controlo orbital de uma câmara em torno de um ponto

Este mecanismo permite controlar a câmara de tal forma que consiga realizar um percurso pré-definido em 3D de modo a visualizar regiões de interesse sobre catálogos de estrelas.

4.6.2 Proposta de API

Para controlar os mecanismos de controlo desta componente, é proposta a seguinte lista de métodos expostos pela sua API:

- SetPivotPosition(x, y, z)

  Aplica uma nova posição do ponto *pivot*.

- SetDistance(distance)

  Aplica uma nova distância entre a câmara e o ponto *pivot*.

- SetLocalRotation(x, y)

  Aplica a rotação da câmara em torno do ponto *pivot*.

- AddPivotPosition(x, y, z)

  Aplica uma translação à posição do ponto *pivot*.

- AddDistance(distance)

  Adiciona um valor à distância entre a câmara e o ponto *pivot*.

- AddLocalRotation(x, y)

  Adiciona um valor à rotação da câmara em torno do ponto *pivot*.

- PerformPath(path)

  Utiliza as funcionalidades disponíveis nesta componente para percorrer o percurso com a câmara de visualização. O atributo fornecido define uma lista onde cada elemento contém uma posição 3D e uma rotação local da câmara.

4.7 User Application

A componente User Application comporta-se como a interface entre o utilizador e a aplicação.

Esta utiliza o input do utilizador para controlar a visualização 3D e explorar os dados.

Como cada plataforma pode conter um tipo de input diferente, esta componente deverá processar esse input de forma a que seja compatível com a componente Mecanismos de Interação. Este input é dependente da plataforma utilizada, podendo utilizar diferentes tipos de input para controlar a câmara de visualização de forma intuitiva para o utilizador.

Capítulo 5

Implementação

5.1 Introdução

Com base na arquitetura definida, de seguida será realizada uma implementação de um protótipo funcional.

Em cada secção é descrita a implementação realizada para cada um dos componentes da arquitetura, bem como as decisões efetuadas para solucionar cada um dos problemas identificados no capítulo de abordagem.

5.2 Wrapper de Comunicação Cliente-Servidor

A componente Wrapper de Comunicação Cliente-Servidor implementada neste protótipo tem como o objetivo interagir especificamente com o Servidor de Objetos.

Utilizando a API oferecida pelo servidor de objetos, um protocolo de comunicação foi desenvolvido para realizar a troca de mensagens entre o cliente e servidor.

5.2.1 Comunicação

Para realizar estes pedidos, é utilizada a biblioteca AJAX [1] que possibilita enviar e receber pedidos HTTP pela Internet. Esta componente utiliza essa biblioteca para realizar pedidos sobre cada uma das funcionalidades implementadas.

Segundo a especificação do servidor de objetos, estas mensagens requerem um conjunto de parâmetros que seja compatível com a sua API. Estes pedidos serão compostos por conjuntos de atributos que definem o tipo de pedido e o tipo de informação requisitada.

[1] http://api.jquery.com/jquery.ajax

Como exemplo, se o cliente efetuar um pedido para recolher a informação sobre um catálogo de estrelas, o pedido deverá conter dois atributos: "cmd-id", que representa o tipo de comando deste pedido, e "vis-id", que representa o identificador de uma visualização. Esta visualização no servidor de objetos representa o catálogo de estrelas pré-processado no servidor.

Sabendo estes parâmetros, este pedido seria efetuado da seguinte forma: $"cmd-id = visualization-info?vis-id = 1234567"$.

Quando esta componente recebe uma resposta vinda do servidor de objetos, esta processa o conteúdo da mensagem de tal forma que o seu resultado seja compatível com as restantes componentes.

Assim, a componente Wrapper poderá no futuro implementar outro tipo de servidor de objetos sem necessitar de alterar qualquer outra componente do sistema, desde que forneça o mesmo tipo de dados depois de processar as mensagens recebidas.

5.3 Proxy de Servidor de Objetos

A componente Proxy de Servidor de Objetos foi implementada para fornecer as funcionalidades que a componente de visualização de estrelas irá utilizar para recolher a informação e dados do catalogo de estrelas.

As duas funcionalidades principais envolvem o pedido da informação de um conjunto de octantes e o pedido dos octantes que estão visíveis pela câmara. Nesta secção, será descrita a implementação realizada para efetuar estes pedidos.

Para possibilitar o calculo dos octantes que estão visíveis, foi implementado um mecanismo que gera a estrutura da *octree* que se encontra no servidor. Ao possuir esta estrutura localmente, armazenando informação de meta-dados sobre cada octante, o cliente pode calcular quais são os octantes que estão visíveis na visualização sem comunicar com o servidor.

5.3.1 Octree

Para efetuar o calculo dos octantes que estão visíveis pela câmara de forma eficiente, é vantajoso possuir uma estrutura local que representa a *octree* que se encontra no servidor de objetos. Assim, é possível calcular que octantes estão visíveis em tempo-real sem existir qualquer tipo comunicação remota, diminuindo o tempo necessário para atualizar a visualização e o número de pedidos efetuados ao servidor.

Segundo a especificação do servidor de objetos, os meta-dados da *octree* são enviados sob forma de lista, onde cada elemento representa um octante. Os octantes são inseridos por largura, sendo o primeiro elemento da lista a raiz da *octree*.

Cada octante contém um identificador, a sua posição 3D, o seu tamanho no espaço 3D, o número de estrelas dentro do octante e uma lista de identificadores que representam os seus filhos.

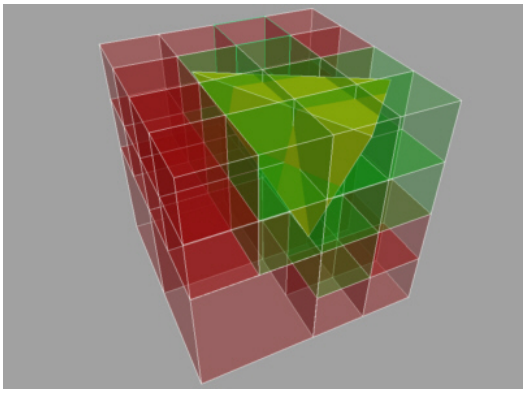

Figura 5.1: Visualização de octantes numa *octree* visiveis pela câmara

Sabendo que a lista contém os octantes inseridos por largura, esta lista é iterada inversamente pelo ultimo elemento da lista. Ao analisar cada octante desta forma, é garantido que os filhos pertencentes ao octante a ser analisado já foram processados, pois os pais de cada octante estarão sempre a uma posição inferior na lista.

Ao processar cada octante, este é inserido numa estrutura de dados do tipo *hashmap*, onde o objeto é associado ao seu identificador. Desta forma, quando um octante contém filhos, este pode aceder rapidamente a esse *hashmap* para recolher o objeto de cada filho, guardando a sua referencia na sua lista de filhos.

Ao terminar este processamento, o primeiro elemento da lista de octantes, a raiz da *octree*, irá conter uma referência para cada um dos seus filhos, e assim sucessivamente, ficando com a *octree* construída localmente. Ao possuir esta *octree*, torna-se possível calcular localmente os octantes que estão visíveis dentro do volume de visão da câmara.

5.3.2 Cálculo dos octantes visíveis

Para verificar se um octante se encontra visível dentro do volume de visão da câmara, é utilizada a seguinte funcionalidade fornecida pela componente Visualização de Estrelas.

Ao utilizar a biblioteca 3D ThreeJS, esta oferece um método que permite verificar se uma geometria primitiva, neste caso um cubo, interseta o volume de visão da câmara de visualização. Sabendo as dimensões de cada octante, incluído nos meta-dados, é possível verificar a visibilidade de cada octante utilizando este método.

Este teste de interseção de octantes com o volume de visão da câmara pode ser observado na figura 5.1 [2]

Para ter controlo sobre o número de estrelas que serão visualizadas em tempo real, é incluído um parâmetro que especifica o número máximo de estrelas visíveis que serão utilizadas em conjunto na visualização.

[2] http://wiki.maayalee.com/mediawiki/index.phptitle=Graphics_Programming_with_DirectX_9/Frustum_culling

É importante ter controlo sobre este parâmetro, pois desta forma é possível controlar o detalhe da visualização que será observada, podendo reduzir ou aumentar o detalhe da visualização dependendo das capacidades dos dispositivos utilizados. Dispositivos com pouca capacidade de processamento e de memória poderão visualizar o mesmo catálogo de estrelas com menos detalhe, e dispositivos mais poderosos poderão visualizar um maior número de estrelas em tempo real ao incrementar o valor deste parâmetro.

Ao percorrer a *octree* em largura de forma iterativa, o algoritmo irá, para cada octante, verificar se o número atual de estrelas dentro da lista de octantes visíveis já processados, mais o número de estrelas pertencente ao octante que está a ser atualmente processado, é inferior ao limite de estrelas em tempo-real definido anteriormente. Se for, o algoritmo irá de seguida verificar se o octante está visível pela câmara para decidir se este será adicionado à lista de octantes visíveis que serão usados na visualização.

Se o octante for visível pela câmara, este é inserido na lista de octantes visíveis, o número de estrelas deste octante são adicionadas ao número total de estrelas atualmente visíveis, e os filhos do octante serão adicionados à lista de octantes por processar.

De seguida, o algoritmo irá verificar se o octante já se encontra em *cache*. Se este for o caso, os dados do octante serão recolhidos da *cache* e inseridos numa lista, enquanto que os octantes que não se encontram em *cache* serão inseridos numa outra lista separada, onde serão requisitados ao servidor de objetos.

No final deste algoritmo, serão retornadas ambas as listas que contém todos os octantes visíveis pela câmara, limitado pelo número máximo de estrelas visíveis em tempo real na visualização.

5.3.3 Caching

Devido ao enorme tamanho do catálogo Gaia, são poucos os dispositivos que conseguem guardar toda a informação em memória local para ser utilizada ao longo de uma visualização. É necessário implementar um mecanismo que realize a manutenção do conteúdo dos octantes que estão em memória, descartando os menos importantes para a visualização que está a ser realizada em cada momento que os dados ultrapassam o limite de armazenamento.

É difícil prever o que o utilizador irá visualizar no futuro, pelo que deverão ser estimados quais os octantes que serão menos importantes para a visualização. Estes octantes terão de ser descartados de tal forma que seja efetuado o menor número possível de pedidos de octantes ao servidor ao longo da visualização.

Devido a limitações dos *browsers*, não é possível aceder à informação sobre a utilização de memória por parte do processo do protótipo, necessitando de um outro mecanismo para iniciar o evento que irá descartar dados dos octantes da memória.

Assim, será adicionado um parâmetro que define o limite de número de estrelas que poderão ser guardadas em memória. Quando o número total de estrelas em memória

ultrapassa este limite, é invocado um mecanismo que irá descartar os octantes menos importantes para a visualização.

Quando este mecanismo é invocado, o algoritmo irá utilizar o critério de tempo sem utilização de cada octante para descartar os menos importantes. Sabe-se que os octantes que estão à mais tempo sem serem utilizados terão uma menor probabilidade de serem novamente usados. Assim, no caso critico de ser necessário de descartar octantes de memória, os octantes que não são utilizados à mais tempo são um bom candidato.

No fim de ordenar os octantes por tempo sem utilização, serão eliminados, um a um, os octantes com mais tempo sem utilização, até o número de estrelas em memória ser inferior ao limite definido.

5.4 Visualização de Estrelas

Para implementar a visualização 3D interativa deste protótipo, a componente Visualização de Estrelas utiliza a biblioteca ThreeJS para criar o ambiente 3D utilizando WebGL.

Adicionalmente, esta componente irá interagir com a componente Proxy Servidor de Objetos para requisitar o conjunto de octantes visíveis que serão utilizados na visualização 3D.

5.4.1 Visualização 3D

Para introduzir um conjunto de estrelas pertencentes a um octante no ambiente 3D, é criado um novo objeto 3D do tipo ThreeJS.Points, que representa uma nuvem de pontos em 3D. Este objeto é composto por um material transparente, que utiliza uma textura de uma estrela para representar cada uma das estrelas.

Para atribuir um conjunto de estrelas de um octante a um objeto 3D, cada posição 3D de cada estrela será utilizada para representar um vértice da geometria do objeto. A biblioteca ThreeJS utiliza esta lista de pontos 3D para desenhar cada estrela no ambiente 3D.

O resultado da visualização 3D de um catálogo de estrelas neste protótipo pode ser observado nas seguintes figuras 5.2 e 5.3.

A criação de um novo objeto 3D tem custos elevados, bem como a destruição destes objetos. Assim, quando um objeto deixa de ser utilizado, este é desativado e guardado em *cache* para mais tarde ser reutilizado novamente. Isto torna a remoção e adição de octantes da visualização eficiente.

5.4.2 Atualização da visualização

Para implementar uma visualização dinâmica do catálogo de estrelas em tempo real, é implementado um mecanismo eficiente que utiliza os octantes visíveis que se encontram em *cache* para atualizar a visualização de estrelas instantaneamente, e irá requisitar, de

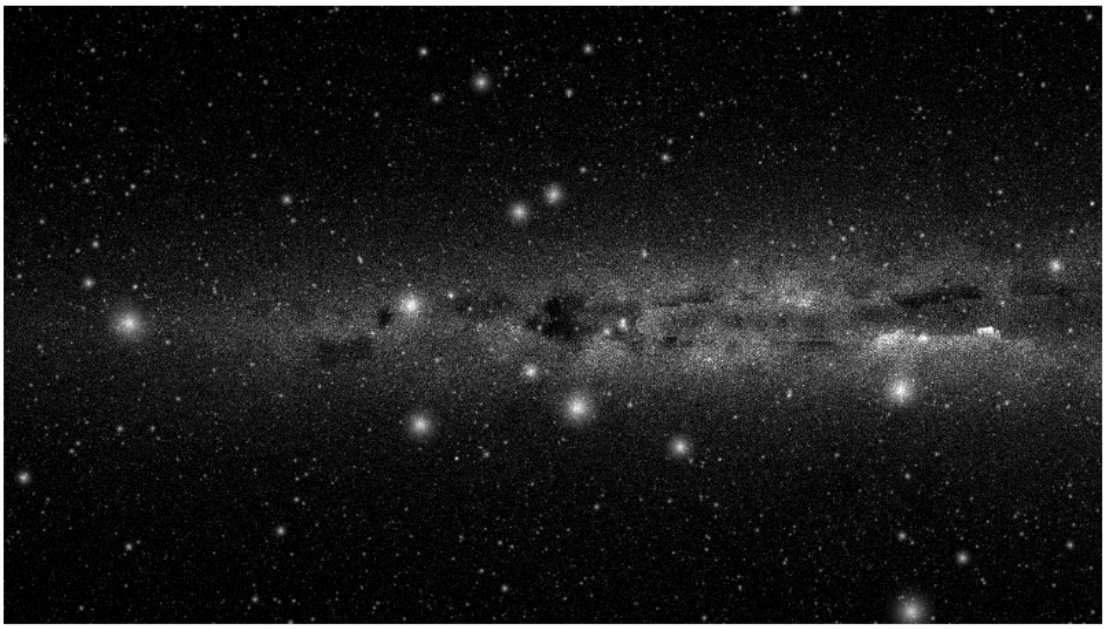

Figura 5.2: Visualização 3D de um catálogo de estrelas no protótipo desenvolvido

forma assíncrona, os octantes que não se encontram em memória, não interrompendo ou diminuindo a interação do utilizador com a visualização.

Para conseguir este tipo de atualização eficiente, serão efetuadas quatro etapas: Inicialmente será detetado se a câmara se moveu; de seguida são calculados os octantes que estão visíveis na nova posição da câmara através da componente Proxy Servidor de Objetos, depois são requisitados os octantes que não se encontram em *cache* ao servidor; por fim, a visualização é atualizada com os novos octantes recebidos. Esta interação pode ser observada através do diagrama de sequência 5.4

Para detetar se a câmara se moveu entre o ultimo e atual estado da visualização, os valores que representam a posição e direção da câmara são guardados em cada instante. Comparando estes valores entre dois instantes indica se existiu algum tipo de movimento.

Quando o movimento da câmara é detetado, o algoritmo irá requisitar de seguida os octantes que estão atualmente visíveis à componente Proxy Servidor de Objetos. Como descrito anteriormente, este método irá percorrer a *octree* e calcular os octantes visíveis com base no volume de visão da câmara, devolvendo no final uma lista destes octantes.

Esta lista irá conter um conjunto de octantes que já se encontram em memória, e um conjunto de IDs dos octantes que terão de ser requisitados ao servidor. Ao possuir esta lista, a componente de Visualização de Estrelas irá desativar os octantes que já não se encontram visíveis na visualização, substituindo-os pelos novos octantes visíveis. Os octantes que estão visíveis, mas que não se encontram em memória, terão de ser requisitados ao servidor.

Se o algoritmo pedisse todos estes octantes ao servidor de uma só vez, tornaria a criação e atualização da visualização 3D muito demorada. Apesar de o método ser assíncrono

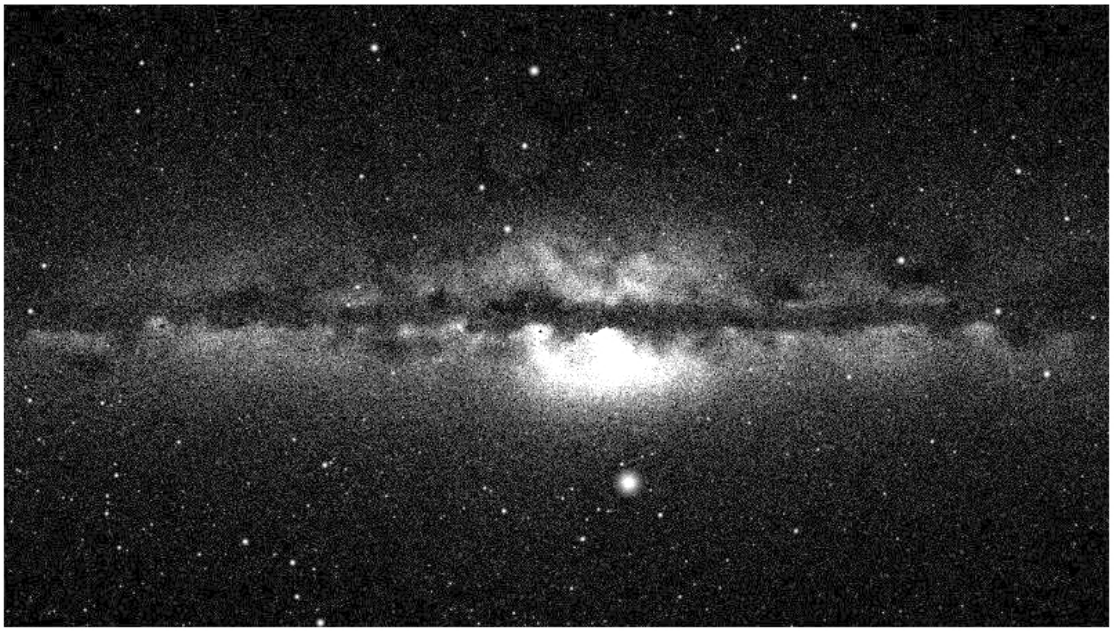

Figura 5.3: Visualização 3D de um catálogo de estrelas no protótipo desenvolvido, visualizando o centro da nossa galáxia

e o utilizador não ser interrompido, o tempo de espera para receber os octantes poderá ser muito elevado se a quantidade de dados for muito elevada. Neste caso, o utilizador receberia atualizações completas, mas demoradas.

Para diminuir o tempo de espera da atualização da visualização, o conjunto dos octantes visíveis será dividido em conjuntos mais pequenos, onde um conjunto de octantes será pedido apenas após receber os dados do conjunto pedido anteriormente. Isto torna a atualização da visualização mais eficiente, mostrando resultados ao utilizador de uma forma mais rápida e progressiva.

Assim, para realizar os pedidos desta forma, o conjunto de octantes visíveis que serão requisitados será guardado numa variável no qual será acedida no momento em que são realizados os pedidos assíncronos.

Para inicializar o processo de pedidos de octantes, é recolhido o primeiro conjunto e o algoritmo irá utilizar a API da componente Proxy Servidor de Objetos para requisitar os octantes ao servidor de objetos através da componente Wrapper de Comunicação entre Cliente e Servidor.

Para evitar que sejam calculados novos octantes visíveis ao movimentar a câmara, durante o processo de atualização, uma *flag* é ativada, no qual irá bloquear a verificação do estado da câmara, evitando realizar pedidos enquanto o utilizador interage com a visualização 3D.

No entanto, quando o utilizador explora os dados e modifica o estado da câmara, os octantes visíveis já não serão os mesmos em relação à atualização que está a ser efetuada. Assim, quando a câmara é afetada durante a atualização da visualização, a componente

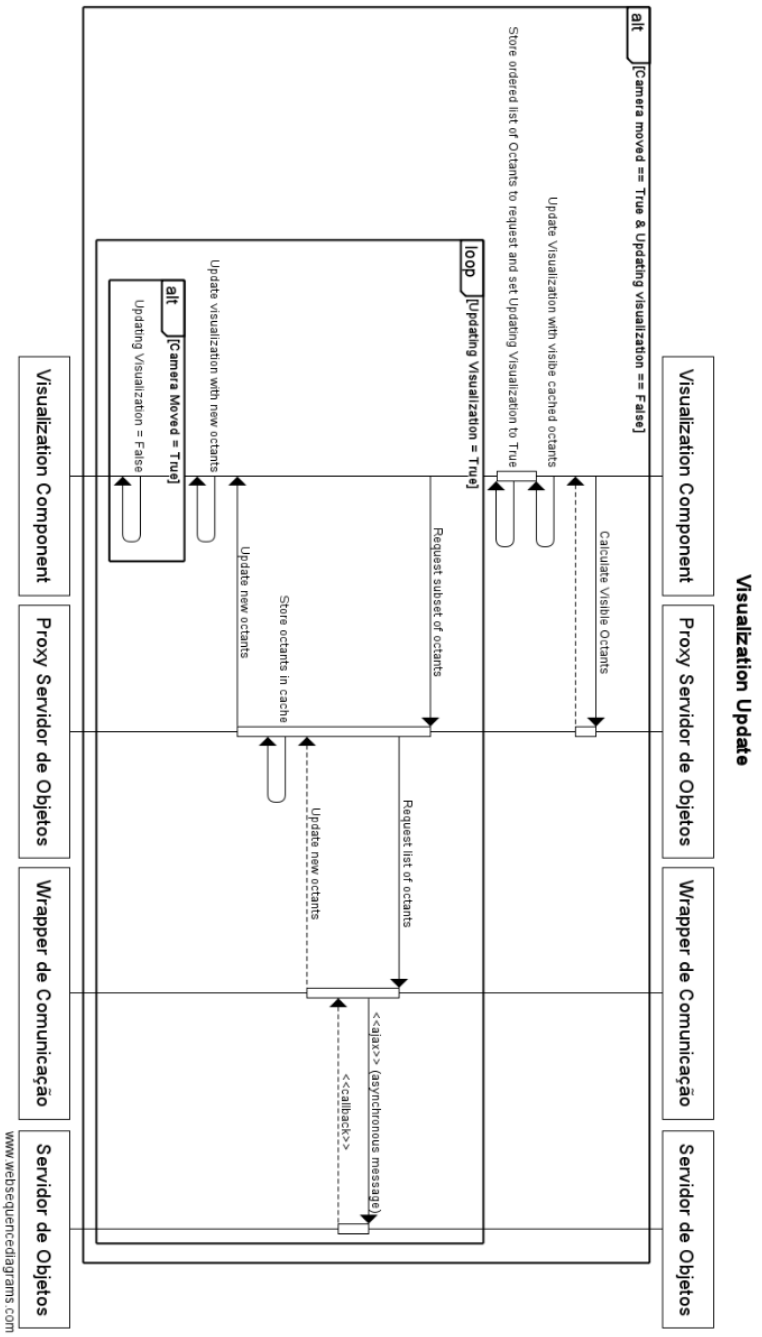

Figura 5.4: Diagrama de sequência do processo de atualização da visualização

termina o processo de atualização quando recebe o novo conjunto de dados, recomeçando todo o processo de novo. Este mecanismo evita requisitar octantes ao servidor que já não se encontram dentro do volume de visão da câmara.

Ao receber os dados do servidor pela componente Wrapper de Comunicação entre Cliente e Servidor, estes novos octantes são transmitidos à componente Proxy Servidor de Objetos, onde serão guardados em *cache*, e de seguida, são transmitidos à componente de Visualização de Estrelas para atualizar a visualização 3D com estes novos octantes.

Ao terminar a adição destes novos octantes à visualização, é realizado um novo pedido com um novo conjunto dos octantes visíveis que estão por processar, efetuando o processo novamente até não existir octantes visíveis que não se encontram em *cache*.

No fim, quando todos os octantes visíveis se encontram na visualização 3D, a *flag* é desativada, e o processo retorna ao estado normal.

5.5 Mecanismos de Interação

A componente Mecanismos de Interação implementa o comportamento que a câmara pode efetuar dentro da visualização 3D.

O tipo de interação implementado nesta componente, conhecido como controlo orbital 4.2, controla a câmara de tal modo que orbite em torno de um ponto 3D, chamado *pivot*. Em adição à rotação em torno de um ponto, será possível controlar a distância entre a câmara e esse ponto, podendo aproximar ou afastar a câmara da área de visualização.

Por fim, é incluído um mecanismo que permite fornecer um conjunto posições e orientações que definem um percurso que esta componente utilizará para movimentar a câmara pelo ambiente 3D.

5.5.1 Controlo de câmara

Para controlar a câmara no ambiente 3D, esta componente necessita de três tipos de input: controlo posicional do ponto *pivot*; controlo da distância da câmara ao ponto *pivot*; controlo da rotação da câmara em torno desse ponto.

O ponto *pivot* é controlado através de uma variável que define a sua posição 3D no ambiente. Esta variável é atualizada através de um método disponibilizado pela sua API, no qual a componente User Application poderá invocar.

Esta atualização pode ser incremental ou total. Sendo aplicada uma atualização total, o valor da variável é substituído pelo valor fornecido no método, enquanto que uma atualização incremental adiciona o valor fornecido ao valor atual da variável.

Isto oferece a possibilidade de o utilizador fornecer um valor especifico para posicionar este ponto *pivot* no ambiente, ou utilizar a diferença do valor do input do utilizador entre o valor do seu ultimo estado e do seu valor atual para realizar uma translação na posição do ponto *pivot*.

Para controlar a câmara de visualização em torno deste ponto, esta componente utiliza dois valores: o valor da distância que a câmara está do ponto *pivot*, e a rotação aplicada nos eixos yy e xx em relação ao ponto *pivot*, onde a rotação no eixo yy está contida entre [0, 360[graus e a rotação no eixo xx está contida entre [-90, 90] graus. Desta forma, o utilizador consegue orbitar a câmara em torno do ponto *pivot*, e controlar a sua distância em simultâneo.

Tal como efetuado para atualizar a variável de posição do ponto *pivot*, as variáveis que controlam a câmara poderão ser atualizadas de forma incremental ou total, onde cada uma pode ser afetada atravez da API desta componente.

5.5.2 Posicionamento da câmara

Para calcular a posição da câmara utilizando as variáveis fornecidas, os dois ângulos serão utilizados para encontrar a posição 3D da câmara na superfície de uma esfera de raio igual a um. Este valor define o vetor 3D normalizado da origem até esta posição 4.2.

Ao multiplicar este vetor pela variável que define a distância da câmara ao ponto *pivot*, obtém-se o vetor que representa a posição local da câmara em relação a esse ponto.

Assim, é possível definir a seguinte função que devolve a posição da câmara no ambiente 3D com base nas variáveis fornecidas, onde (x,y,z) representa a posição 3D do ponto *pivot*, (α, β) representam os ângulos de rotação da câmara, e 'd' representa a distância entre a câmara e o ponto *pivot*:

$$F((x,y,z),(\alpha,\beta),d) = (x,y,z) + (Sin(\beta), Sin(\alpha), Cos(\alpha)) * d \quad (1)$$

Sabendo a posição resultante da câmara, de seguida será aplicada a sua rotação, onde será direcionada em direção ao ponto *pivot*. Utilizando os valores de rotação invertidos resulta em ângulos que apontam no sentido oposto em relação ao vetor calculado anteriormente.

Assim, aplicando como rotação local da câmara os ângulos $(\alpha, \beta, 0)$, transformará a orientação da câmara para que esta aponte em direção ao ponto *pivot*.

5.5.3 Percurso 3D

Utilizando os mecanismos implementados para controlar a câmara, é possível utilizar estas variáveis para controlar a câmara ao longo de um percurso pré-definido, onde um percurso é definido por um conjunto de elementos, onde cada elemento contem uma posição e rotação, como demonstrado na figura 5.5 [3] .

Sabendo que o objetivo deste mecanismo é posicionar e rodar a câmara ao longo deste percurso, a câmara será posicionada ao utilizar a posição fornecida para posicionar o ponto *pivot* no ambiente 3D, a distância entre a câmara e o ponto *pivot* será igual a 0, e será aplicada a orientação fornecida à orientação da câmara.

Para posicionar e orientar a câmara ao longo de dois elementos do percurso, este mecanismo utilizara ambos elementos para calcular o seu estado dentro do percurso. Definindo um valor real 't', contido entre os valores [0,1], tal que quando t = 0 a câmara possui os valores do elemento atual, e quando t = 1, a câmara possui os valores do elemento seguinte, e quando 0 < t < 1, a câmara está posicionada entre ambos os elementos.

[3] https://cinematography.inria.fr/resources/intuitive-and-efficient-camera-control-with-the-toric-space

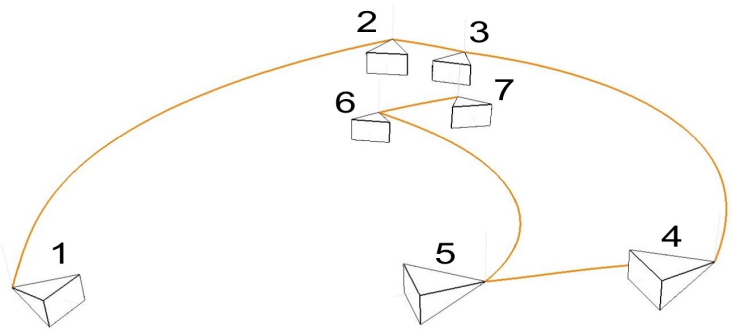

Figura 5.5: Visualização de um conjunto de posições e rotações que definem o percurso realizado por uma câmara 3D

Ao definir este valor 't', é possível definir uma função que calcula a posição da câmara entre elementos da seguinte forma:

$$F(pos0, pos1, t) = pos0 * (1 - t) + pos1 * t(2)$$

O calculo da rotação intermédia entre elementos do percurso difere do calculo da posição. Isto resulta da natureza dos valores dos ângulos, que variam entre 0º e 360º.

Se a transformação for realizada linearmente, como efetuado para o posicionamento da câmara, o valor de rotação que será efetuado poderá não ser o menor valor possível para essa rotação.

Como exemplo, supondo que se quer realizar uma rotação de 20º para 340º, utilizando uma transformação linear resultará numa rotação com um valor 340º-20º=320º, rodando a câmara pelo caminho mais longo entre estas duas rotações, sabendo que o caminho mais próximo tem um valor de 40º ao invés de 320º.

De forma a rodar corretamente a câmara em torno de dois valores, será necessário calcular qual o valor de rotação mais curto entre estes dois valores. Isto é ser calculado ao verificar se a diferença entre o ultimo e o primeiro valor é maior que 180º. Se este valor for superior a 180º, é acrescentado um valor de 360º ao valor da primeira rotação. Se a diferença entre os dois valores for inferior a -180º, é decrementado um valor de 360º ao segundo valor.

Após esta modificação, a rotação da câmara poderá ser efetuada utilizando a mesma transformação linear realizada para o calculo da posição da câmara entre elementos do percurso.

Utilizando o exemplo anterior, com o objetivo de realizar uma rotação de 20º para 340º, ao verificar que a diferença entre o valor final e o valor inicial é superior a 180º, é adicionado um valor de 360º à primeira rotação, resultando em 380º. Ao realizar a transformação linear, o valor de rotação agora aplicado entre estas duas rotações é igual a 40º, resultando numa rotação correta entre estas duas rotações.

Capítulo 6

Avaliação

Neste capitulo, será realizada uma avaliação do protótipo desenvolvido com base na arquitetura e implementação descritos nos capítulos anteriores.

Serão efetuados testes para avaliar as capacidades do protótipo, e verificar que o protótipo satisfaz todos os requisitos necessários para criar uma visualização 3D interativa no *browser* de grandes catálogos de estrelas usando o Servidor de Objetos.

Para avaliar as capacidades do protótipo, este será testado recolhendo dados sobre o seu funcionamento, desempenho e utilização de memória. Nestes testes, será observado o comportamento do sistema de *caching* e como o protótipo reutiliza o conteúdo guardado em *cache* para diminuir o número de pedidos efetuados ao Servidor de Objetos durante a visualização.

Estes testes serão focados na componente Proxy de Servidor de Objetos, visto esta ser a componente responsável por controlar o número de estrelas que estão visíveis em tempo real e o número máximo de estrelas que serão mantidas em memória, utilizando o sistema de *caching*.

Como primeiro passo, serão definidos os dados usados em cada teste. De seguida, será descrito o método de como cada teste será realizado, e como serão recolhidos os resultados ao longo de cada teste. Por fim, será efetuada a realização dos testes e análise dos resultados obtidos.

Estes testes serão realizados num computador portátil, cujo Hardware tem as seguintes especificações:

- CPU: Intel® Core™ i5-4210U (Dual-Core), clock speed: 1.70/2.70 Turbo GHz
- GPU: AMD Radeon™ R7 M260
- RAM: 8GB DDR3L @ 1600 MHz

- Display: 1,366 x 768 px

6.1 Dados

Não possuindo acesso aos dados do catálogo GAIA na altura da realização desta dissertação, os testes serão realizados utilizando o catálogo GUMS [15], que representa uma simulação dos dados finais da missão GAIA. Não utilizar o catálogo GAIA para esta avaliação não é problemático, visto o catálogo GUMS ter sido construído com o objetivo de simular os dados reais, tendo o mesmo tamanho, formato e tipo de dados.

Devido ao enorme tamanho do catálogo GUMS e a limitações da máquina pessoal utilizada como Servidor de Objetos, não é possível utilizar o catálogo GUMS completo, pelo que serão utilizadas versões de menor dimensão destes dados. Para realizar estes testes, serão utilizadas duas versões deste catálogo, uma com 10 milhões de estrelas, e outra com 50 milhões de estrelas.

Medindo o comportamento do protótipo com estas duas versões, é possível verificar quão adaptável o protótipo é para catálogos de estrelas com uma quantidade de dados elevada o suficiente que não seja possível para um dispositivo normal armazenar todos os dados em memória. Assim, será possível estimar o comportamento do protótipo para uma quantidade ainda mais elevada de dados, tal como o catálogo GAIA de 1000 milhões de estrelas.

6.2 Preparação

Para realizar os testes de forma consistente, é implementado um mecanismo que realiza um conjunto de testes de forma automatizada.

Utilizando o mecanismo de percurso 3D, desenvolvido na componente Mecanismos de Interação, serão definidos conjuntos de posições e rotações, onde a câmara se irá movimentar dentro da visualização sem interação por parte de um utilizador.

Ao utilizar este mecanismo, é garantido que todos os testes são efetuados de igual forma, podendo comparar os resultados para os diversos testes efetuados.

O protótipo possui dois parâmetros ajustáveis que afetam a visualização durante a sua execução. Serão realizados testes para diversas combinações dos seguintes parâmetros, verificando qual o seu impacto na visualização 3D e nos recursos utilizados pelo protótipo:

- Número máximo de estrelas visíveis na visualização 3D
- Número máximo de estrelas que poderão ser armazenadas em *cache*

Para recolher dados ao longo do percurso, estes serão recolhidos a cada segundo passado até terminar o percurso. Serão recolhidos os seguintes tipos de dados:

- Número de octantes/estrelas requisitados ao servidor, medido como a soma total destes valores desde a ultima recolha destes dados

- Número de estrelas visíveis na visualização, com base no calculo dos octantes atualmente visíveis dentro do volume de visão da câmara. Este resultado inclui estrelas ainda não armazenadas em memória
- Número de estrelas visíveis e carregadas na visualização, onde estas estrelas já foram requisitadas ao Servidor de Objetos, armazenadas em cache e adicionadas à visualização 3D
- Número de estrelas encontradas em *cache* e reutilizadas na visualização, medido como a soma total destes valores desde a ultima recolha destes dados
- Número de octantes/estrelas descartados da memória, medido como a soma total destes valores desde a ultima recolha destes dados
- Utilização média de memória RAM pelo processo do protótipo, medido como a média durante o decorrer de cada teste
- Taxa de refrescamento da visualização (*frames per second*)

Para cada subconjunto de dados do catalogo GUMS, serão efetuados dez testes utilizando os mesmos parâmetros, medindo no fim, a média dos resultados obtidos.

6.3 Resultados

Utilizando os dados definidos anteriormente, e o processo que será utilizado para realizar os testes e a recolha de dados, serão efetuados de seguida os testes para ambos os catálogos de estrelas.

Os resultados serão apresentados em forma de gráfico de linhas 2D, onde o eixo horizontal representa o tempo decorrido durante o teste, e o eixo vertical representa o valor do tipo de dados recolhido ao longo do teste.

Para avaliar o protótipo com ambos os catálogos de estrelas, cada avaliação utilizará os mesmos parâmetros que afetam a quantidade de estrelas visíveis em cada momento durante o percurso, e o número máximo de estrelas que poderão ser armazenadas em *cache*.

Serão utilizadas duas variações destes parâmetros de forma a poder comparar o comportamento do protótipo quando este aumenta significativamente o número de estrelas que poderá visualizar e armazenar.

Os resultados destes testes mostrarão o impacto que um catálogo de estrelas de 50 milhões de estrelas terá no protótipo em comparação com um catálogo de 10 milhões de estrelas.

Os parâmetros utilizados para ambas as avaliações do protótipo serão os seguintes:

- Parâmetros tipo 1

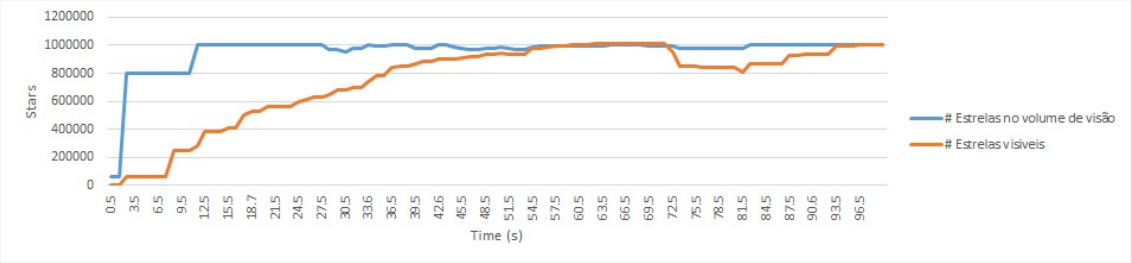

Figura 6.1: Resultados do número de estrelas em tempo real durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

Número máximo de estrelas em tempo real igual a 1 milhão de estrelas

Número máximo de estrelas que poderão ser guardadas em memória igual a 1.25 milhões de estrelas

- Parâmetros tipo 2

  Número máximo de estrelas em tempo real igual a 2.5 milhões de estrelas

  Número máximo de estrelas que poderão ser guardadas em memória igual a 2.75 milhões de estrelas

6.3.1 Avaliação com catálogo de 10 milhões de estrelas

6.3.1.1 Parâmetros tipo 1

O gráfico 6.1 representa o número de estrelas que a visualização calcula estarem visíveis em cada momento (linha azul), com base no calculo dos octantes visíveis dentro do volume de visão da câmara, e o número de estrelas que estão realmente visíveis na visualização em cada momento (linha laranja), no qual já foram requisitadas ao Servidor de Objetos e inseridas na visualização.

Analisando este gráfico, é observado que o número de estrelas na visualização é incrementado de forma linear e suave ao longo do tempo, não requisitando um número excessivo de estrelas em cada pedido efetuado ao servidor. Num pequeno período de tempo, o número de estrelas visíveis converge para o número real que a visualização calcula estarem visíveis.

Ao efetuar os pedidos desta maneira, e de forma assíncrona, a visualização 3D mantém-se fluida para o utilizador, como pode ser observado no gráfico 6.2, conseguindo manter a taxa de refrescamento entre 40 e 60 imagens por segundo.

O protótipo requisita os pedidos por ordem de importância que cada octante tem para a visualização atual. Como na maioria dos casos os octantes mais importantes também contêm mais estrelas, estando no todo da *octree*, não é ótimo requisitar um número excessivo de octantes que contenham um número elevado de estrelas em cada pedido, limitando assim o número de estrelas que serão requisitadas ao Servidor.

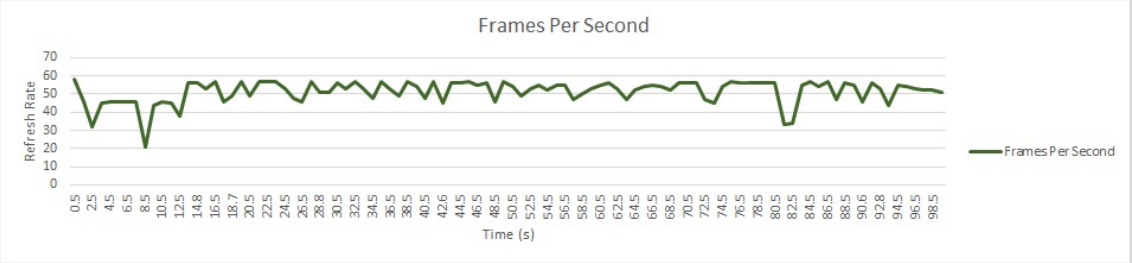

Figura 6.2: Taxa de refrescamento durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

Figura 6.3: Número de estrelas requisitadas durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

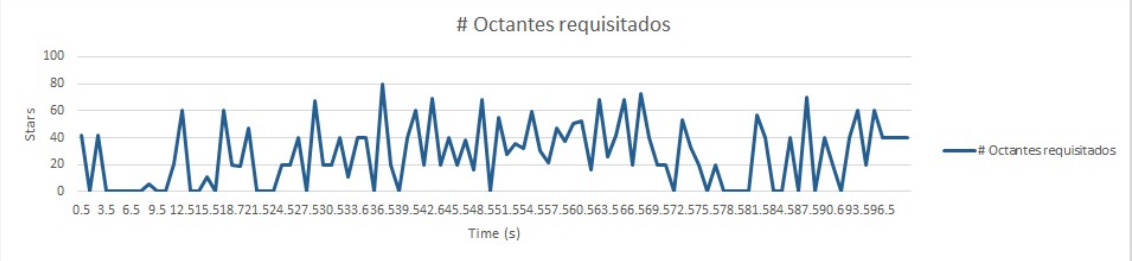

Figura 6.4: Número de octantes requisitados durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

Pode ser observado, no gráfico 6.3, que durante os primeiros 40 segundos, o número de estrelas requisitadas ao Servidor de Objetos é elevado na maioria dos pedidos, diminuindo ao longo do percurso.

Existe um número elevado de octantes que contém uma quantidade pequena de estrelas, pelo que estes são sempre agrupados de forma a realizar o menor número de pedidos possível. Como existem muitos octantes de pequenas dimensões, o número de octantes pedidos ao longo do percurso mantém-se constante, como pode ser observado no gráfico 6.4

Ao longo da visualização, o número de estrelas requisitadas diminui, visto estas serem guardadas em *cache* quando são recebidas pelo protótipo 6.5 e reutilizadas quando estas saem e entram novamente dentro do volume de visão da câmara de visualização. 6.6

Devido à importância dos primeiros octantes requisitados ao Servidor de Objetos,

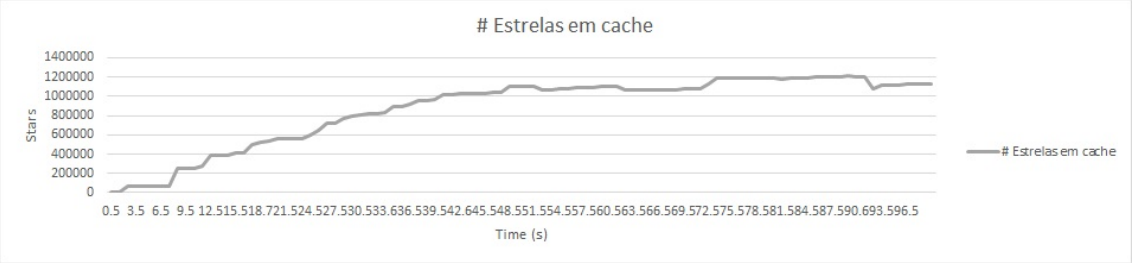

Figura 6.5: Número de estrelas guardadas em *cache* durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

Figura 6.6: Número de estrelas reutilizadas na visualização durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

estes octantes têm uma dimensão física tão elevada que são quase sempre visíveis pela câmara de visualização. Como estes são raramente excluídos da visualização 3D, são poucas vezes reutilizados ao longo da visualização, como pode ser observado pelo número de estrelas reutilizadas nos primeiros 50 segundos da visualização no gráfico 6.6.

Após este período de tempo, octantes de menor dimensão já são requisitados ao Servidor de Objetos e guardados em *cache*. Como estes octantes têm uma dimensão física menor, estes são removidos e reutilizados na visualização mais frequentemente, observando que o número de estrelas reutilizadas aumenta com o decorrer do percurso.

Nos últimos momentos do percurso, entre os segundos 70 e 80, pode ser observado no gráfico 6.1, que o número de estrelas visíveis tem uma queda acentuada. Isto deve-se à câmara rodar e apontar para uma região onde as estrelas ainda não foram requisitadas ao Servidor de Objetos. De seguida, a câmara volta a rodar para a mesma zona, podendo observar o sistema de *caching*, no gráfico 6.6, a reutilizar um elevado número de estrelas para atualizar a visualização.

Ao longo do percurso, o protótipo guarda em *cache* as estrelas que vai recebendo do Servidor de Objetos. Este número sobe até ao momento que ultrapassa o limite imposto do número máximo de 1.25 milhões de estrelas que pode ser guardado em memória. Pode ser observado no gráfico 6.7, que quando este número ultrapassa o limite, o sistema de *caching* descarta os octantes menos importantes para a visualização, reduzindo a quantidade de memória RAM utilizada pelo protótipo.

Deste modo, é possível garantir que o protótipo não utilize memória excessiva quando

Figura 6.7: Número de estrelas descartadas da cache durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 1

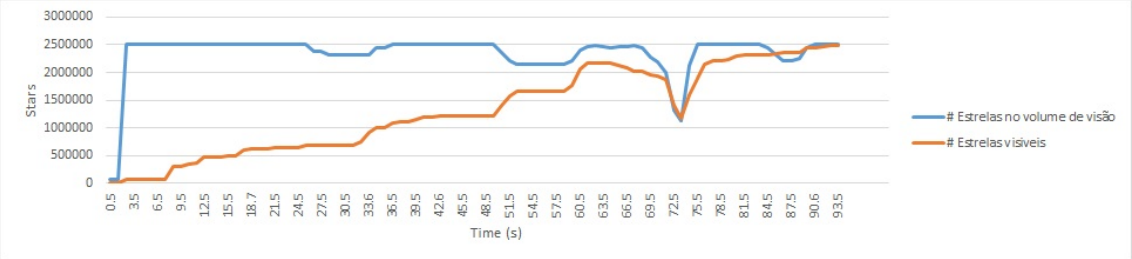

Figura 6.8: Resultados do número de estrelas em tempo real durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 2

o utilizador explora a visualização. Durante este percurso, o processo do protótipo utiliza em média 550MB de memória RAM, sendo aceitável para a maioria dos dispositivos *desktop* de gama baixa e média.

6.3.1.2 Parâmetros tipo 2

Ao analisar os resultados utilizando os parâmetros tipo 2, verifica-se que a quantidade de informação utilizada pelo protótipo incrementa significativamente. Contudo, o comportamento do protótipo mantém-se consistente em comparação com os resultados anteriores. Como pode ser observado no gráfico 6.8, o número de estrelas inseridas na visualização aumenta de forma linear, sem realizar pedidos com quantidades excessivas de estrelas, até alcançar o limite de número de estrelas visíveis em tempo real.

Como a visualização possui um limite mais elevado do número de estrelas em tempo real, o tempo de convergência do número de estrelas para esse novo limite aumenta, demorando mais tempo até a visualização atualizar todas as estrelas que estão visíveis pela câmara.

Tal como nos resultados anteriores, existe uma descida mais acentuada do número de estrelas que estão visíveis entre os segundos 70 e 80, observando que o protótipo de seguida, quando o numero de estrelas visíveis aumenta, reutiliza as estrelas guardadas em *cache* para atualizar a visualização rapidamente, como pode ser observado no gráfico 6.9

Figura 6.9: Número de estrelas reutilizadas da cache na visualização durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 2

Figura 6.10: Número de estrelas requisitadas ao Servidor de Objetos durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 2

Devido ao maior número de estrelas nesta visualização, a quantidade de estrelas requisitadas ao Servidor de Objetos durante o inicio da visualização prolonga-se durante um maior período de tempo.

Ao contrário dos resultados anteriores, esta visualização nunca ultrapassou o número máximo de estrelas que poderiam ser guardadas em memória, não existindo necessidade de descartar estrelas. Observando o gráfico 6.10, no final do percurso, o número de estrelas aproxima-se desse limite. Se o percurso tivesse uma maior duração, muito possivelmente o sistema de *caching* inicializaria o processo de descartar estrelas de *cache*.

Em termos de desempenho, a mudança dos limites não teve grande impacto no comportamento do protótipo, como pode ser observado no gráfico 6.11, onde a taxa de refrescamento mantém-se elevada ao longo do percurso. Durante o percurso, o protótipo utilizou em média 850MB de memória RAM, sendo aceitável para a maioria dos dispositivos.

6.3.2 Avaliação com catálogo de 50 milhões de estrelas

Tal como a avaliação com o catálogo de estrelas de 10 milhões de estrelas, esta avaliação utilizará os mesmos valores atribuídos aos limites da visualização e sistema de *caching* para avaliar o protótipo com o catálogo de 50 milhões de estrelas.

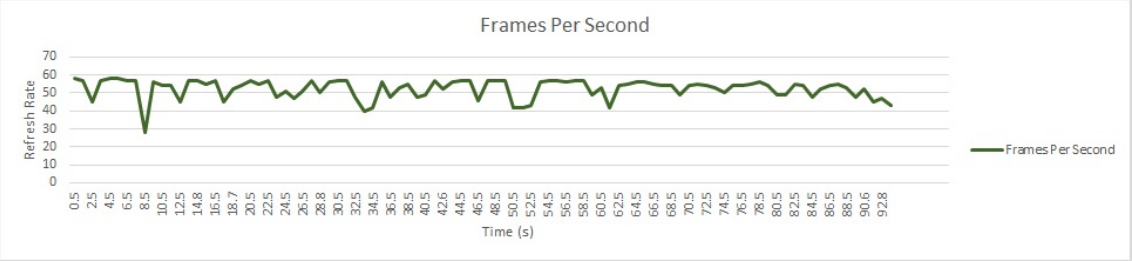

Figura 6.11: Taxa de refrescamento durante o percurso, na avaliação com catálogo de 10 milhões de estrelas, utilizando os parâmetros tipo 2

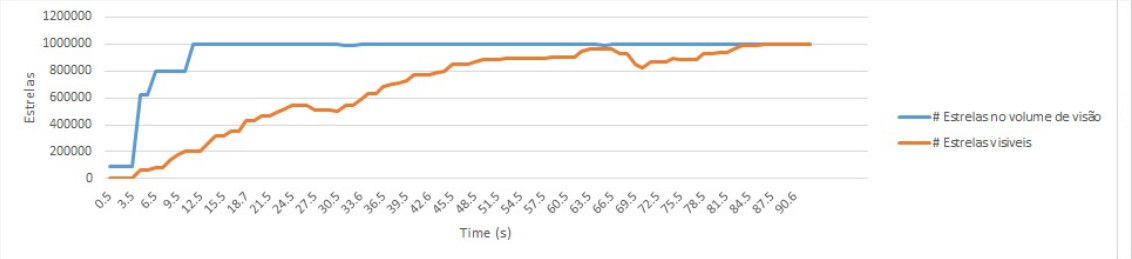

Figura 6.12: Resultados do número de estrelas em tempo real durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 1

Figura 6.13: Número de estrelas requisitadas durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 1

6.3.2.1 Parâmetros tipo 1

Analisando o gráfico 6.12, onde apresenta o número de estrelas atualmente visíveis, é possível verificar que não existe uma diferença elevada em comparação com os resultados obtidos na secção 6.3.1.1 . O número de estrelas inseridas na visualização durante o primeiro minuto mantém-se entre os mesmos valores em cada pedido realizado ao Servidor de Objetos, como pode ser observado no gráfico 6.13.

De forma semelhante ao catálogo de 10 milhões de estrelas, o desempenho do protótipo mantém-se razoavelmente consistente e elevado ao longo da visualização, como pode ser visto no gráfico 6.14.

O sistema de *caching* também se comporta de maneira semelhante, reutilizando conjuntos de estrelas quando a câmara observa regiões que contém octantes em *cache*, gráfico 6.15, e descarta os octantes menos importantes para a visualização quando o número de

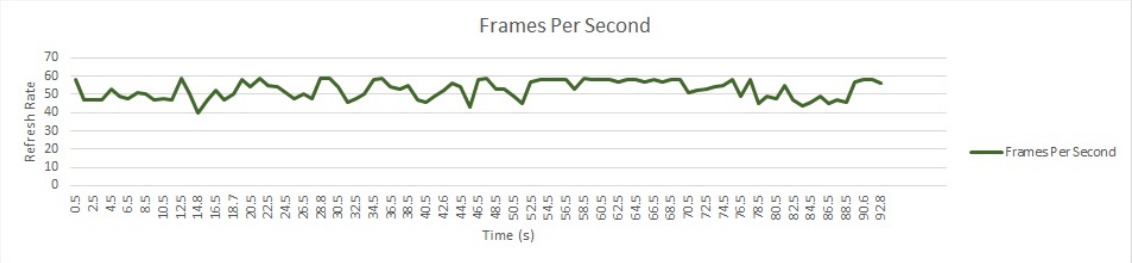

Figura 6.14: Taxa de refrescamento durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 1

Figura 6.15: Número de estrelas reutilizadas na visualização durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 1

Figura 6.16: Número de estrelas descartadas da cache durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 1

estrelas ultrapassa os limites definidos para este teste, gráfico 6.16.

Ao utilizar um catálogo com uma dimensão cinco vezes superior ao catálogo utilizado na secção anterior, o tamanho dos metadados que representam a *octree* deste catálogo é um pouco mais elevado, resultando na utilização média de memória RAM pelo protótipo subir para 575MB. Este aumento não é significativo, pelo que o valor se mantém dentro de níveis aceitáveis para os dispositivos atuais de baixa gama.

Sabendo que o catálogo de 50 milhões de estrelas ocupa 27GB de espaço, e o catálogo de 10 milhões de estrelas ocupa 7GB de espaço, o impacto desta diferença no cliente utilizando ambos os catálogos é insignificante, ou mesmo inexistente em termos de desempenho e funcionamento do protótipo.

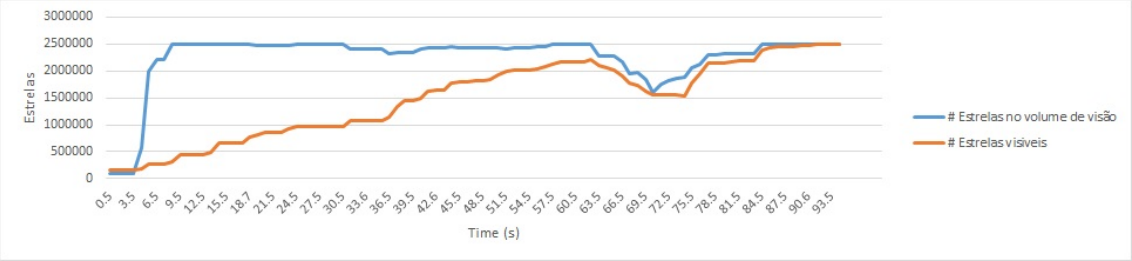

Figura 6.17: Resultados do número de estrelas em tempo real durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 2

Figura 6.18: Número de estrelas reutilizadas da cache na visualização durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 2

6.3.2.2 Parâmetros tipo 2

Como esperado nos resultados anteriores, o protótipo comporta-se de forma semelhante em comparação com o catálogo de estrelas de 10 milhões utilizando os mesmos parâmetros.

Observando o gráfico 6.17, existe uma subida gradual de novos conjuntos de estrelas adicionadas à visualização durante a primeira metade dos testes realizados, requisitando os octantes mais importantes para a visualização neste período de tempo.

Entre os segundos 60-80 desse gráfico, existe igualmente uma queda do número de estrelas que a câmara está a visualizar. Esta queda não é tão elevada em comparação com a queda do número de estrelas utilizando o catálogo de 10 milhões de estrelas, como pode ser observado no gráfico **??**. Isto deve-se a este catálogo de estrelas ter uma maior quantidade de estrelas que engloba todo o catálogo, existindo um menor número de quedas e quedas menos acentuadas.

Como nos testes realizados anteriormente, verifica-se no gráfico 6.18 que o sistema de *caching* reutiliza grandes quantidades de estrelas ao longo da visualização, principalmente nos momentos em que a câmara volta a observar estrelas que estão em *cache*.

Mesmo existindo um maior número de estrelas neste catálogo, a atualização de novas estrelas na visualização mantém-se dentro dos mesmos valores, demorando tempo semelhante a requisitar e receber octantes do Servidor de Objetos. Devido a esta demora, o sistema de *caching* não ultrapassa o número de estrelas máximo que poderá ser guardado em *cache*, pelo que não são descartadas estrelas durante o percurso efetuado.

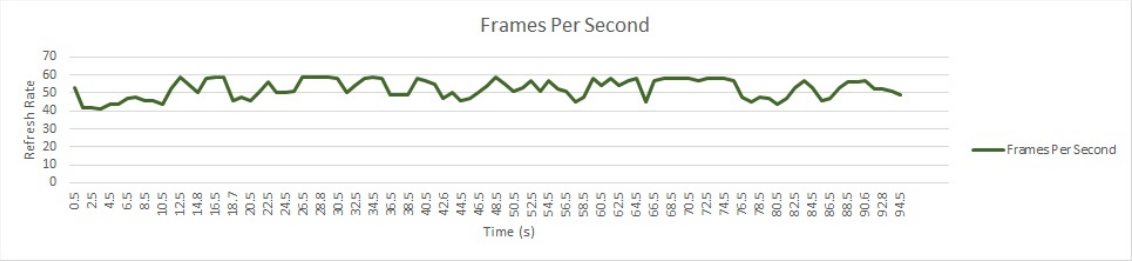

Figura 6.19: Taxa de refrescamento durante o percurso, na avaliação com catálogo de 50 milhões de estrelas, utilizando os parâmetros tipo 2

Adicionalmente, a taxa de refrescamento mantém-se elevada durante o percurso efetuado, como pode ser observado no gráfico 6.19.

A utilização média de memória RAM ao longo do percurso mantém-se em valores semelhantes em comparação com os resultados do catálogo de estrelas de 10 milhões. Como utiliza os mesmos valores para limitar o número de estrelas utilizadas na visualização e em memória, a quantidade de memória RAM usada será semelhante. Como a dimensão dos metadados é maior para este catálogo, a utilização média de memória do protótipo sobe para 880 MB de RAM.

6.3.3 Análise dos Resultados Obtidos

Analisando os resultados dos testes efetuados neste capítulo, verifica-se que o protótipo desenvolvido é capaz de criar visualizações interativas sobre catálogos de estrelas de grandes dimensões.

O protótipo utiliza o sistema de *caching* para guardar os octantes que são recebidos do servidor de forma a os poder reutilizar no futuro, reduzindo o número de pedidos que serão efetuados ao Servidor de Objetos num longo período de tempo. Isto também reduz o tempo de espera do utilizador por novos dados quando este explora a visualização.

Utilizando as capacidades do protótipo de limitar a quantidade de dados que são utilizados para visualizar os catálogos de estrelas, o protótipo nunca excede as capacidades físicas dos dispositivos.

Observando os resultados obtidos, ambos os catálogos de estrelas se comportam de forma semelhante, tendo em conta que a dimensão entre os dois é elevada, onde o segundo catálogo de estrelas tem cinco vezes mais informação que o primeiro catálogo.

Não sendo atualmente possível testar o protótipo desenvolvido com o catálogo Gaia, não será possível verificar se o protótipo se irá comportar de igual forma como se comportou durante os testes realizados nesta avaliação.

Em ambos os testes, o protótipo manteve uma taxa de refrescamento aceitável e semelhante para uma visualização interativa, bem como uma utilização razoável de memória RAM para armazenar os dados em *cache*.

Verifica-se nos resultados dos testes, que estes dois valores de desempenho não dependem do tamanho original do catálogo de estrelas. Estes dependem dos valores atribuídos aos limites utilizados pelo protótipo para limitar a quantidade de dados que a visualização utiliza do catálogo de estrelas.

Sabendo esta informação, é possível estimar que o protótipo se deverá comportar de forma semelhante com o catálogo de mil milhões de estrelas do Gaia se utilizar os mesmos valores de limitação de recursos no mesmo percurso 3D efetuado pelos testes.

Capítulo 7

Conclusões

Com o decorrer da missão GAIA, realizada pela Agência Espacial Europeia, um dos objetivos principais tornou-se em conseguir expor a vasta quantidade de informação, recolhida pelo satélite, de mil milhões de estrelas da nossa galáxia para o grande publico.

Para isto, foi iniciado o projeto GAVIDAV, no qual utilizará as capacidades e vantagens dos *browsers* para conseguir alcançar o publico geral, de tal modo que os utilizadores poderão facilmente aceder a diversas ferramentas para explorar os dados oferecidos pelo GAIA.

Este projeto GAVIDAV inclui uma ferramenta de visualização 3D que possibilita ao utilizador explorar os dados fornecidos pelo GAIA num ambiente tridimensional.

O projeto IVELA, realizado anteriormente ao GAVIDAV, focou-se em fornecer uma aplicação *desktop* com o objetivo de criar uma ferramenta de exploração, interação e análise de dados astronómicos de grandes dimensões. Para além de suportar variedades de catálogos de estrelas, este foi também construído de forma a suportar o catálogo de estrelas do GAIA.

Não sendo prático utilizar o catálogo de estrelas na sua totalidade para criar uma visualização 3D, foi construído um Servidor de Objetos no âmbito do projeto IVELA, que fornece um acesso inteligente aos dados de catálogos de estrelas de grandes dimensões.

Sendo o Servidor de Objetos uma solução fulcral para o projeto IVELA, este foi também incluído no projeto GAVIDAV para fornecer um acesso eficiente e eficaz ao grande volume de dados do GAIA.

Realizando um estudo do estado de arte no contexto deste projeto, foi efetuada uma análise e descrição dos objetivos da missão GAIA e do projeto GAVIDAV, bem como uma análise pormenorizada sobre o projeto IVELA e Servidor de Objetos.

De forma a estudar as melhores soluções atualmente disponíveis para resolver a complexidade de visualização de grandes volumes de estrelas dentro de um *browser*, foi efetuada uma análise de WebGL, no qual elimina a necessidade de instalação de *plug-ins* no *browser*, e oferece uma API 3D poderosa, capaz de criar ambientes 3D de alta complexidade.

Sabendo que WebGL é uma API de muito baixo nível, foram analisadas diversas bibliotecas de alto nível, onde reduzem o nível de complexidade necessário para criar aplicações 3D utilizando WebGL.

Adicionalmente, foram analisadas diversas soluções atuais para visualização e exploração de catálogos de estrelas, tanto no *browser* como em ambiente *standalone*. Estas visualizações apenas suportam catálogos de estrelas de pequenas dimensões, não sendo possível visualizar o grande catálogo do GAIA.

Para abordar este problema, foram identificados os requisitos necessários para possibilitar a criação de visualizações 3D interativas dentro do *browser*, e as dificuldades que estes requisitos apresentam. Com base nos requisitos analisados, foi definida uma arquitetura, compatível com o servidor de objetos, capaz de produzir uma visualização 3D no *browser* para o catálogo GAIA.

Esta arquitetura divide e agrupa os requisitos por componentes que interagem entre si para transmitir informação sobre o catálogo de estrelas, interagir com o servidor de objetos, e produzir a visualização 3D interativa.

Com base na arquitetura definida, foi implementado um protótipo no qual fornece uma ferramenta capaz de realizar uma visualização 3D e exploração dos dados do catálogo GAIA diretamente num *browser*. Este protótipo consegue escalar a sua visualização de forma a que consiga visualizar mais ou menos quantidades de estrelas, sendo compatível com a maioria dos dispositivos existentes.

Ao realizar uma avaliação sobre o protótipo desenvolvido, verifica-se que satisfaz todos os requisitos necessários para criar uma visualização 3D interativa no *browser*, verificando que é capaz de escalar para o grande volume de dados do catálogo GAIA.

7.1 Trabalhos Futuros

Para além de o protótipo desenvolvido satisfazer todos os requisitos e objetivos deste projeto, existem certas secções da sua implementação que poderão ser melhoradas no futuro.

Na implementação do protótipo desenvolvido, o sistema de *caching*, pertencente à componente Proxy de Servidor de Objetos, utiliza apenas um critério para ordenar os octantes que estão em *cache* pela sua ordem de importância para a visualização 3D. Este critério, que utiliza o tempo sem utilização do octante na visualização, poderá ser apenas um critério de uma lista de critérios que classificam a importância de cada octante.

Utilizando mais critérios para identificar os octantes que serão descartados, poderá melhorar a eficiência deste algoritmo, diminuindo o número de octantes que o protótipo

necessitará de requisitar novamente, que por sua vez tornará a atualização da visualização mais rápida com novos dos dados do catálogo de estrelas.

Para utilizar o protótipo num dispositivo de forma eficiente, o utilizador necessitará de especificar os limites impostos à visualização, de forma a que o seu dispositivo não tente utilizar mais recursos do que os que possui. Assim, poderia ser implementado um mecanismo onde o protótipo testa inicialmente as capacidades do dispositivo do utilizador automaticamente, verificando quais os limites que o dispositivo é capaz de processar.

Para controlar a câmara de visualização, foi implementado um mecanismo de controlo do tipo orbital. Para aumentar a variedade de formas que o utilizador pode utilizar para interagir com a visualização, poderão ser implementados outros tipos de interação com a visualização, como exemplo, *free-flight*, onde a câmara navega pelo ambiente 3D sem uma âncora no espaço.

Para o desenvolvimento deste protótipo, foi escolhida a plataforma *desktop* para implementação. No futuro, poderão ser adicionadas novas plataformas que poderão utilizar outros tipos de input para controlar e interagir com a visualização 3D, tais como *tablets* e *smartphones*.

Bibliografia

[1] *Bidelman, Eric. "The Basis of Web Workers" HTML5Rocks, July (2010).*

[2] *Bouy, H., and J. Alves. "Cosmography of OB stars in the solar neighbourhood. "Astronomy Astrophysics 584 (2015): A26.*

[3] *Chang, Michael. "Making 100,000 Stars" HTML5Rocks Case Studies, November (2012).*

[4] *Echterhoff, Jonas. "Unity 5.3 WebGL Updates" Unity blog, December (2015).*

[5] *ESO/IDA/Danish 1.5 m/R.Gendler and A. Hornstrup, "Spiral Galaxy NGC 1232", European Southern Observatory, December (2009).*

[6] *Jackins, Chris L., and Steven L. Tanimoto. "Oct-trees and their use in representing three-dimensional objects."Computer Graphics and Image Processing 14.3 (1980): 249-270.*

[7] *Kent, Stephen M. "Sloan digital sky survey."Science with astronomical near-infrared sky surveys. Springer Netherlands, 1994. 27-30.*

[8] *Meagher, Donald J. "The Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer, IPL-TR-80-111, Rensselaer Polytechnic Institute, Image Processing Lab (October 1980).*

[9] *Microsoft. 2013. Babylon.js.*

[10] *Mozilla. "Unreal Engine 4.7 Binary release includes HTML5 Export" The Mozilla Blog, February (2015).*

[11] *Nazarov, Rovshen, and John Galletly. "Native browser support for 3D rendering and physics using WebGL, HTML5 and Javascript."BCI (Local) 1036 (2013): 21.*

[12] *Okamoto, Shusuke, and Masaki Kohana. "Load distribution by using web workers for a real-time web application."International Journal of Web Information Systems 7.4 (2011): 381-395.*

[13] *Perryman, M. A. C., et al. "GAIA: Composition, formation and evolution of the Galaxy."Astronomy Astrophysics 369.1 (2001): 339-363.*

[14] *Perryman, Michael AC, et al. "The HIPPARCOS catalogue."Astronomy and Astrophysics 323 (1997): L49-L52.*

[15] *Robin, A. C., et al. "Gaia Universe model snapshot-A statistical analysis of the expected contents of the Gaia catalogue."Astronomy Astrophysics 543 (2012): A100.*

[16] *Schuh, Justin. "Saying goodbye to our old friend NPAPI."Chromium Blog, September (2013).*

[17] *Taylor, M. B. "TOPCAT STIL: starlink table/VOTable processing software."Astronomical Data Analysis Software and Systems XIV. Vol. 347. 2005.*