ARAMIS HORNUNG MORAES

Middleware Interativo para Sistemas Imersivos

Curitiba - PR, Brasil

2020

ARAMIS HORNUNG MORAES

Middleware Interativo para Sistemas Imersivos

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná como requisito parcial para obtenção do título de mestre em Informática.

Pontifícia Universidade Católica do Paraná - PUCPR

Programa de Pós-Graduação em Informática - PPGIA

Orientador: Prof. Dr. Edson José Rodrigues Justino

Curitiba - PR, Brasil

2020

Dados da Catalogação na Publicação
Pontifícia Universidade Católica do Paraná
Sistema Integrado de Bibliotecas – SIBI/PUCPR
Biblioteca Central
Pamela Travassos de Freitas – CRB 9/1960

M827m 2020

Moraes, Aramis Hornung
Middleware interativo para sistemas imersivos / Aramis Hornung Moraes ; Orientador: Edson José Rodrigues Justino. – 2020.
114 f. : il. ; 30 cm

Dissertação (mestrado) – Pontifícia Universidade Católica do Paraná, Curitiba, 2020
Bibliografia: f. 113-114

1. Informática. 2. Conectividade (Computadores). 3. Fulldome. 4. Interação homem-máquina. 5. Middleware. 6. Projeção audio-visual. 7. Realidade virtual. I. Justino, Edson José Rodrigues. II. Pontifícia Universidade Católica do Paraná. Pós-Graduação em Informática. III. Título.

CDD 22. ed. – 001.64

Pontifícia Universidade Católica do Paraná
Escola Politécnica
Programa de Pós-Graduação em Informática

DECLARAÇÃO

Declaro para os devidos fins que o aluno **ARAMIS HORNUNG MORAES**, defendeu sua dissertação de Mestrado intitulada **"Middleware Interativo Para Sistemas Imersivos"**, na área de concentração Ciência da Computação, no dia 09 de março de 2020, no qual foi aprovado.

Declaro ainda que foram feitas todas as alterações solicitadas pela Banca Examinadora, cumprindo todas as normas de formatação definidas pelo Programa.

Por ser verdade, firmo a presente declaração.

Curitiba, 18 de maio de 2020.

Prof. Dr. Emerson Cabrera Paraiso
Coordenador do Programa de Pós-Graduação em Informática
Pontifícia Universidade Católica do Paraná

Dedico este trabalho ao meu Pai, minha Mãe e ao meu Irmão e sua Família.

Agradecimentos

Primeiramente agradeço a Deus.

Agradeço ao meu Pai e minha Mãe que me apoiaram durante todo o processo.

Ao meu orientador, Prof. Dr. Edson José Rodrigues Justino pelo acolhimento e experiência.

Aos professores: Manoel de Campos Almeida, Dr. Edson Scalabrin, Dra. Mauren Abreu de Souza e Dr. Flávio Bortolozzi pelos aconselhamentos e conhecimento despendido.

Ao Coordenador do Curso de Ciência da Computação Luiz Antônio Pavão pela introdução à área.

Agradeço à Pontifícia Universidade Católica do Paraná (PUCPR) pela oportunidade.

Por fim, aos colegas do grupo de pesquisa.

"O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001"[1]

[1] <http://capes.gov.br/CECOL/Portaria_CAPES_DOU_206_de_2018.pdf>

"A gravidade explica os movimentos dos planetas,
mas não pode explicar quem colocou os planetas em movimento.
Deus governa todas as coisas e sabe tudo que é ou que pode ser feito.
Isaac Newton

Resumo

Este trabalho propõe um sistema que visa adaptar ambientes imersivos, principalmente *fulldome*, para reprodução de conteúdo interativo e *streams* de vídeo da *internet*. Atualmente, o conteúdo de realidade virtual é abundante, principalmente em plataformas *online*. Entretanto, ambientes *fulldome*, não estão usando essas plataformas e estão ficando isolados, com cada vez menos variedade de conteúdo para reprodução. O objetivo do trabalho então é de atualizar esses ambientes, adicionando funcionalidades que possibilitem a integração dos mesmos com as plataformas de conteúdo online, além de disponibilizar uma ferramenta que permita o desenvolvimento de conteúdo interativo. A solução do trabalho proposto também visou a interação do usuário com o ambiente por meio de dispositivos móveis, tais como celulares *smartphones* e *tablets*, em tempo de execução, de modo a facilitar a interação com a aplicação. Do ponto de vista técnico, a partir da implementação feita, se obteve resultado compatível com o objetivo do trabalho. Conseguiu-se reproduzir vídeos diretamente do YouTube e executar aplicações interativas em tempo real em um teatro *fulldome*, tudo controlado por meio de um *smartphone*.

Palavras-chave: *Fulldome*, Conectividade, Realidade Virtual, Ambientes Imersivos.

Abstract

This work proposes a middleware that aims to adapt immersive environments, mainly fulldome, to execute interactive content and internet video streams. Currently, virtual reality content is abundant, especially on online video streaming platforms. However, fulldome environments are not using these platforms and are isolated, with less and less variety of content. The objective of this work is to update these environments, adding features that enable the integration with online platforms that have immersive content, in addition to providing a tool that allows the development of interactive content. The proposed solution also aimed at user interaction with the environment through mobile devices such as smartphones and tablets, at runtime, in order to facilitate the interaction with the application. From a technical point of view, from the implementation made, a result compatible with the objective of the study was obtained. It was possible to play videos directly from YouTube and run interactive applications in real time on a fulldome theater, all controlled through a smartphone.

Keywords: Fulldome, Connectivity, Virtual Reality, Immersive Environments.

Lista de ilustrações

Lista de algoritmos

Lista de tabelas

Lista de abreviaturas e siglas

PPGIA Programa de Pós-Graduação em Informática

PUCPR Pontifícia Universidade Católica do Paraná

API *Application Programming Interface*

CSMs *Cascaded Shadow Maps* - Método de aplicação de sombras em malhas geométricas 3d.

IDE *Integrated development environment* - Ambiente integrado de desenvolvimento.

GLSL *OpenGL Shading Language* - Linguagem de programação de *shaders* usado em *APIs* OpenGL

GPU *Graphics processing unit* – Processador gráfico.

Hardware Skinning Método de animação de malhas geométricas que apresenta melhor desempenho, pois, faz uso do processador gráfico para animação e não do processador aritmético.

HDR *High Dynamic Range* – ou Alcance Dinâmico, em português. São métodos utilizados em computação gráfica para alargar o alcance dinâmico (o trecho entre o valor mais escuro e o mais claro de uma imagem).

HLSL *High Level Shading Language* – linguagem de programação de *shaders*, da Microsoft, usado em *APIs DirectX*

Framework É um *software*, formado por um ou vários programas, que podem ser modificados para atender um propósito específico. No contexto deste trabalho, o *framework* é um conjunto de programas (softwares gráficos, softwares de rede, som e outros) que juntos formam um software maior, com propósito de ser usado para criar trabalhos derivados, como jogos digitais, simuladores de realidade virtual, dentre outros.

Middleware É um *software* normalmente empregado para integrar outros componentes de *softwares*.

FpMED *Framework* para multi-projeção estereoscópica distribuído

Fisheye (imagem) Refere-se a uma imagem gerada com ângulo de perspectiva amplo, normalmente, a partir de 135 graus de abertura.

Domemaster Descreve um formato de vídeo padrão para plataforma *fulldome*.

HUB (atividade) Descreve-se algo como um *hub* quando este é um centro relevante para determinada atividade.

Edutainment Termo inglês que faz junção entre as palavras "*Education*" e "*Entertainment*".

RV Realidade Virtual

VR *Virtual Reality*

RA Realidade Aumentada

AR *Augmented Reality*

Display Equipamento para visualização; tela; monitor

Open Source Código aberto, caracteriza *software* que possui código fonte aberto.

Free Software *Software* livre. Caracteriza *software*, que além de código fonte aberto, possui aspecto de liberdade. Isso é feito através da obrigatoriedade de se distribuir o *software* juntamente com o seu respectivo código fonte, perpetuando a liberdade de se estudar e modificar o programa.

DOF *Deph of Field*, Profundidade de campo. Também pode significar *Degree of Freedom*, graus de liberdade.

Form Factor (Imagem) Aspecto da imagem que define e prescreve o tamanho, a forma e outras especificações da mesma.

DLP *Digital Light Processing*.

Tracker (captura de movimentos) Equipamento que permite o seu rastreio no espaço.

Motion capture Captura de Movimentos, denomina o processo de gravação/captura de movimento e transposição do movimento de uma referência para um modelo digital.

Story Telling Do inglês, Narração de histórias.

Anti Aliasing Filter (imagem) Método de redução de serrilhamento em imagens

Hyperlinks Uma referência dentro de um documento em hipertexto a outras partes desse documento ou a outro documento.

URL *Uniform Resource Locator*

Voxel	Um *voxel* representa um valor em um *gride* regular em um espaço tridimensional.
UHD	*Ultra High Definition*.
Fulldome	Plataforma de visualização que possuí superfície de visualização em formato de cúpula.
FPS	*Frames per Second* - Número de vezes que uma aplicação gráfica desenha sua cena na tela no intervalo de um segundo.
RTT	*Render to Texture*, é uma funcionalidade presente em motores gráficas que permite usar texturas geradas em tempo de execução por uma câmera.
GB	*gigabytes*, equivalente a um bilhão de *bytes*.
MB	*megabytes*, equivalente a um milhão de *bytes*
JSON	*JavaScript Object Notation*
Gigaflops	Um bilhão de cálculos com ponto flutuante por segundo.
Teraflops	Um trilhão de cálculos com ponto flutuante por segundo.
LCD	*Liquid-Crystal Display*, do inglês, tela de cristal líquido.
GIS	*Geographic Information System*, do inglês, Sistema de informação geográfica.
REST	*Representational state transfer*, do inglês, Transferência Representacional de Estado.
PWA	*Progressive Web App*, é um tipo de aplicação servida através de navegadores *web*.

Lista de símbolos

α	Letra grega minúscula alfa
ρ	Letra grega minúscula Ro
θ	Letra grega minúscula Teta
ϕ	Letra grega minúscula Fi
Γ	Letra grega Gama
Λ	Lambda
ζ	Letra grega minúscula zeta
$\in$	Pertence

Sumário

1 Introdução

1.1 Contextualização

A tecnologia de realidade virtual (RV) começou a ser comercializada, de maneira branda, no início dos anos 90. Foram poucas empresas que investiram nesse mercado na época, pois além de um desafio tecnológico, os produtos resultantes eram de custo incompatível para produção em grande escala. Os equipamentos produzidos, se baseavam em óculos de realidade virtual (em inglês *VR headset*). A tecnologia disponível na época era insuficiente para se criar um produto que proporcionasse uma experiência agradável por tempo prolongado. As telas integradas nos óculos de realidade virtual, por exemplo, tinham baixa resolução, cerca de 300 *pixels* de largura e altura. Os sensores não eram tão precisos ou havia uma demora de responsividade. Os gráficos não eram adequados para uma experiência imersiva e as lentes do conjunto ótico dos *headsets* tinham distorções. O mercado para *hardware* de RV desaqueceu. Houve pouca criação de produtos do gênero entre os anos 2000 e 2010.

A retomada da tecnologia de realidade virtual se deu com o início do desenvolvimento do *headset* de RV *Oculus Rift* em 2010. Um motivo para isso foi a tecnologia ter se aprimorado. Em 2010 já existiam telas pequenas de 7 polegadas com alta resolução de 1200 *pixels* de largura por 800 de altura. Os microprocessadores em 2010 já eram usados em celulares *smartphone* com arquitetura *ARM* e tinha 1GHz de frequência de processamento e 512 MB de memória *RAM*. Ficou viável, a partir dessa época, o desenvolvimento de novos *headsets* de RV.

Após o lançamento do *Rift* em agosto de 2012, empresas como Facebook, Google, Microsoft e Steam investiram no desenvolvimento de produtos de RV. A empresa Oculus através de arrecadação de fundo colaborativo lançou a primeira versão comercial do Oculus em 2012. O Facebook compra a empresa Oculus em 2014, dando seguimento no desenvolvimento do *headset*. Em 2015 o Google disponibiliza no YouTube a possibilidade para *reprodução de vídeos em 360º*[1]. No mesmo ano o Facebook *também adiciona essa funcionalidade*[2] em sua rede social. Durante o ano de 2016, vários produtores de conteúdo publicam nestas plataformas vídeos 360º.

Ao longo do ano de 2016, empresas de filmadoras e máquinas fotográficas criam

[1] <https://www.theverge.com/2016/4/18/11450484/youtube-live-360-degree-video-announced-neal-mohan-interview> visto em 08/03/2020

[2] <https://www.facebook.com/facebookmedia/blog/facebook-360-updates-and-introducing-surround-360> visto em 08/03/2020

câmeras específicas para *fotografar e filmar em 360º*[3]. Nesse ano, a empresa Steam, lançou o seu próprio *headset* de RV chamado HTC Vive. No mesmo ano, a Microsoft de maneira a unir tanto o ramo de RV quanto realidade aumentada (RA), lança o *headset* de realidade aumentada *HoloLens*. A diferença entre RA e RV, é que a realidade aumentada estende a visão real do mundo, adicionando elementos virtuais em cima do mundo real. Apesar dessa característica, determinados conteúdos de realidade virtual, podem ser reproduzidos em produtos para realidade aumentada, como é o caso do *HoloLens*.

Em 2017, estúdios de jogos digitais começam a desenvolver títulos para as plataformas de realidade virtual, tais como o *Rift*, *HTV Vive* e outros. Atualmente existe uma vasta quantidade de conteúdo de realidade virtual disponível no mercado e na *internet*. Esse conteúdo vem se acumulando desde 2017 que é quando o ramo *estava em alta no mercado*[4].

Apesar dos *headsets* de RV serem o método mais popular para visualização de conteúdo de RV, existem outras plataformas de visualização imersivas similares, bem consolidadas e exploradas por empresas do ramo. Temos, além do *headsets*, as *Caves*, os cinemas de tela curva (*IMAX*) e os planetários digitais (*Fulldome*).

Cave é um termo amplo, que dentro do contexto de realidade virtual, diz respeito a salas de visualização imersiva, podendo ter diversos formatos, mas geralmente trata-se de uma configuração cilíndrica ou 4 projeções retilíneas nas paredes em torno do espectador.

Um cinema *IMAX* se destaca dos demais, pois usualmente seus teatros possuem uma tela de projeção 6 vezes maior que o empregado em cinemas convencionais. A tela normalmente é curva, até mesmo para compensar a distância das extremidades ao espectador. Esses teatros acabam criando o efeito da imersão por possuírem telas amplas, cuja curvatura cria a sensação de estar envolvendo o espectador, mesmo que parcialmente.

Fulldome é uma plataforma similar a um cinema, com o diferencial de que a visualização é omnidirecional. Teatros *fulldome* geralmente são compostos por uma tela que cobre o espectador, em forma de uma cúpula (semiesfera). Estes agora são mundialmente encontrados em locais como parques temáticos, planetários e centros de ciência, universidades, feiras corporativas dentre outros (LANTZ, 2007). As instalações de *fulldome* geralmente são encontradas em instituições de ensino, incluindo planetários digitais e centros de visualização (ou seja, salas de aula), ou são teatros abertos ao público que cobram uma taxa pela admissão (teatros públicos) (LANTZ, 2007). Segundo o *Fulldome Theater Compendium*[5], dos 27 teatros digitais *fulldome* no Brasil, 8 pertencem ao estado, 9 a museus e 8 a universidades.

Nos últimos 5 anos, ocorreram avanços significativos na produção de conteúdo

[3] <https://vrscout.com/news/nikon-keymission-360-camera-price-release/> visto em 08/03/2020
[4] <https://trends.google.com/trends/explore?date=all&q=virtual%20reality> visto em 08/03/2020
[5] <https://www.lochnessproductions.com/lfco/lfco.html> visto em 31/12/2018

imersivo, como vídeos 180º e 360º, apesar disso, no que diz respeito a ambientes imersivos, tal como variantes de *caves* e planetários *fulldome*, poucas soluções para reprodução destes novos conteúdos têm sido divulgadas.

1.2 Objetivos

1.2.1 Objetivo Geral

Desenvolver um middleware capaz de gerar aplicações interativas e reproduzir *streams* de vídeos imersivos, diretamente da *internet*, para plataformas imersivas, principalmente *fulldome*.

1.2.2 Objetivos Específicos

1. Determinar uma arquitetura de infraestrutura para execução do *middleware*.
2. Definir como será construído o *middleware* (arquitetura simplificada do *software*).
3. Definir um procedimento para gerar imagens *fulldome*.
4. Elaborar um processo para calibrar projeções em ecrãs *fulldome*.
5. Desenvolver uma interface que possibilite a interação dos usuários com a aplicação *fulldome* através do *middleware*.
6. Integrar plataformas da *internet* que contenham mídia de realidade virtual com o ambiente imersivo *fulldome*.
7. Demonstrar o funcionamento do *middleware* com exemplos de aplicações implementadas a partir do mesmo.

1.3 Justificativa

A demanda por conteúdo interativo e educativo para plataformas *fulldome* e a potencialidade de ganho em atividades didáticas no uso destas plataformas para ensino assim como a disponibilidade de tal infraestrutura *fulldome* na PUCPR, foram motivos influenciadores que conduziram o desenvolvimento desse trabalho.

A pouca exploração por parte dos produtores de conteúdo para a plataforma do *fulldome* foi causada pela pequena quantidade de teatros *fulldomes* quando comparado com salas de cinema. Outro fator é a demora da tecnologia em atender as necessidades técnicas para produção de conteúdo para o *fulldome*. Mas esse cenário vem mudando nos últimos anos (FELINTO; ZANG; VELHO, 2012).

Segundo o *Fulldome Theater Compendium*[6] do site da *Loch Ness Productions*[7], existem mais de novecentos teatros digitais ao redor do mundo (2019), número que praticamente dobrou desde 2015 e continua crescendo a cada ano. O rápido crescimento dos cinemas *fulldome* foi em parte devido a adaptação de planetários analógicos com sistemas de projeção digital (LANTZ, 2007). Existiam mais de três mil planetários em todo o mundo, atendendo a mais de cem milhões de visitantes anualmente (LANTZ, 2007). A plataforma *fulldome* é uma tendência para educação e entretenimento (FELINTO; ZANG; VELHO, 2012). Existem grupos dedicados àqueles que tenham interesse na plataforma ou estejam desenvolvendo conteúdo para as mesmas. Um destes *HUBs* de referência é o *site* do *FDDB (the fulldome database)*[8]. Ele está focado em assuntos da plataforma *fulldome* e tecnologias voltadas para múltiplas projeções em geral e até mesmo assuntos que não estritamente as tecnologias, como por exemplo, vagas para trabalho na administração de planetários etc.

Tanto o *site* do *FDDB*[9] quanto o da *Loch Ness Productions*[10] mantém um banco de dados atualizado dos ambientes *fulldome*. Ressalta-se que nem todos os planetários listados possuem a infraestrutura completa. O que se garante é a existência da construção de uma cúpula. Nem todas as instalações citadas possuem projetores digitais e computadores para realizar os espetáculos como visto em teatros *fulldome* digitais. Apesar disso, possuir a infraestrutura básica, que é o prédio com a cúpula, é o principal, já que nesse caso, estão presentes os pré-requisitos para instalar o restante dos equipamentos. Planetários convencionais, os quais não possuem tecnologia digital, vem atualizando seus sistemas de projeção para tecnologia digital (FELINTO; ZANG; VELHO, 2012). Um fato positivo é que os equipamentos tendem a ter os custos reduzidos com o decorrer do tempo. Novas tecnologias estão surgindo com preços mais acessíveis e com melhor custo benefício. São exemplos destes equipamentos, projetores de fósforo a laser, que possuem melhor nitidez de imagem, maior duração de vida útil da lâmpada e custo reduzido a longo prazo de utilização devido a manutenções menos frequentes. Os equipamentos de aquisição de imagens também estão tendo os custos reduzidos, sendo exemplos disso a câmera filmadora *Lady Bug* (FELINTO; ZANG; VELHO, 2012) e a *Nikon KeyMission 360*[11] que além de preço acessível, oferecem uma solução pronta para produção de vídeos 360º.

A falta de conteúdo interativo também é causada pelo difícil acesso dos desenvolvedores a um ambiente *fulldome*. Tomando como referência o Brasil, existem apenas 4 outros ambientes similares ao teatro *fulldome* da PUCPR. Ao redor do mundo a situação se repete, pois os mesmos são projetos de alto custo para serem construídos (TREDINNICK;

6 <https://www.lochnessproductions.com/lfco/lfco.html> visto em 31/12/2018
7 <https://www.lochnessproductions.com/> visto em 31/12/2018
8 <https://www.fddb.org/> visto em 09/02/2020
9 <https://www.fddb.org/domes/> visto em 09/02/2020
10 <https://www.lochnessproductions.com/lfco/lfco.html> visto em 09/02/2020
11 <https://imaging.nikon.com/lineup/action/keymission_360/spec.htm> visto em 09/02/2020

RICHENS, 2015), tornando-os ainda mais incomuns. O desenvolvimento de *displays* imersivos *fulldome* de grande escala tem sido impulsionado em grande parte por interesses comerciais que investem intensivamente em tecnologias proprietárias (LANTZ, 2007). Existem, no entanto, domos mais simples, feitos com fibra de vidro ou domos portáteis infláveis utilizados normalmente em eventos de exposição, que comparado com teatros de alto padrão, não representam custo significativo. Estes domos podem tornar a plataforma ainda mais popular e acessível.

Fazer uso de tecnologias imersivas tal como o *fulldome* mostrou a possibilidade de uma melhora na aprendizagem, visto que esses ambientes são peculiarmente bons para ajudar na percepção e memorização espacial, assim como tornam o aprendizado mais lúdico e intuitivo (LI et al., 2014). O conceito de *edutainment* se torna cada vez mais popular, trazendo conteúdos semelhantes a jogos em cenários educacionais baseados em realidade virtual para tornar o aprendizado mais eficaz e lúdico (LI et al., 2014). Em comparação com a mídia educacional tradicional, como cinemas e museus, a mídia *fulldome* torna o aprendizado mais atrativo (LI et al., 2014). Os resultados de um estudo indicam que os alunos que fizeram uso de ambientes imersivos tal como o *fulldome* se sentiram mais envolvidos ao mesmo tempo em que tinham uma memória mais duradoura da experiência de aprendizagem (LI et al., 2014). Outro estudo indicou que a condição de alta imersão melhora significativamente a precisão e a eficiência geral para alguns tipos específicos de tarefas complexas que exigem pesquisa exaustiva e/ou julgamentos espaciais precisos (SCHUCHARDT; BOWMAN, 2007). Pesquisadores acreditam que esses ambientes imersivos proporcionam maior contexto espacial do que os monitores de tela plana, podendo melhorar o aprendizado de maneiras que não são possíveis através de outros métodos pedagógicos (LANTZ, 2007).

Houve nos últimos 5 anos, a difusão de grande quantidade de conteúdo imersivo na *internet*, não necessariamente para ambientes *fulldome*, mas para óculos de realidade virtual. Mesmo assim, esse conteúdo poderia ser utilizado também na plataforma *fulldome*, o que atualmente não é feito e ainda não existe uma solução consolidada para integração destas plataformas de conteúdo com ambientes imersivos. Um exemplo de onde podemos encontrar tal conteúdo é no próprio *YouTube* que, *em 13 de março de 2015*[12], introduziu oficialmente suporte para vídeos 360º. Seguindo esse anúncio, diversos produtores de conteúdo disponibilizaram na plataforma vídeos em 360º. Existe então, após 5 anos, abundância de conteúdo de realidade virtual na internet e até o momento não consta na literatura ou no mercado um produto que integre essas plataformas com ambientes imersivos (teatros imersivos). O que existe são aplicativos oficiais da própria Google para reprodução de vídeo 360º do YouTube no celular e soluções específicas para óculos de realidade virtual. É evidente que essa área ainda está por ser explorada, com o desenvolvimento de uma

[12] <https://www.premiumbeat.com/blog/youtube-announces-360-degree-video-heres-the-scoop/> visto em 09/02/2020

solução que integre conteúdo de realidade virtual da internet com ambientes imersivos, tal como teatros *fulldomes*.

Em vista da oportunidade que foi oferecida por parte da *PUCPR* para a modernização de sua infraestrutura *fulldome*, também estamos definindo uma arquitetura mais simples e baseada em software de código fonte aberto. Uma das razões é de evitar problemas de obsolescência a curto prazo e expandir a possibilidade de criar novas tecnologias, visto a maior facilidade de trabalhar com projetos *open source* do ponto de vista de flexibilidade para modificações e ajustes.

1.4 Estrutura da Dissertação

A presente dissertação está estruturada em cinco capítulos: no primeiro e corrente capítulo, mostra-se uma contextualização breve sobre o tema, o objetivo geral, objetivos específicos e a justificativa para o desenvolvimento deste trabalho. No segundo capítulo apresenta-se o embasamento teórico do estudo com relação aos ambientes imersivos *fulldome*, encerrando-se com considerações do autor sobre o tema. No terceiro capítulo, é apresentada a metodologia e a infraestrutura para o desenvolvimento da pesquisa. Também é apresentada a classificação da pesquisa, bem como o delineamento da mesma. No quarto capítulo é apresentado o método proposto, sendo abordado em detalhes as etapas que cumprem os objetivos específicos descritos na subseção 1.2.2. Ao fim do quarto capítulo são apresentados exemplos de aplicações, experimentos realizados e as considerações sobre o tema. No quinto capítulo são feitas as considerações finais do trabalho.

2 Revisão de Literatura

Neste capítulo será abordado o material literário disponível sobre *fulldome*. Ele foi dividido em duas partes. Na primeira, iremos abordar o conceito sobre *fulldome* de acordo com a literatura; na segunda parte iremos mostrar efetivamente quais são as aplicações interativas existentes para a plataforma. Ao final, serão feitas algumas considerações do autor quanto ao material levantado.

2.1 Estudo Bibliográfico

2.1.1 *Fulldome* e Ambientes Imersivos

O *fulldome* é uma plataforma de visualização imersiva, na qual a tela envolve o espectador. Normalmente a tela é uma cúpula (semiesfera) onde são projetadas a/as imagem(ens). Trata-se de uma plataforma de visualização incomum. Por essa razão, ainda existem poucas referências na literatura sobre o assunto, como visto na Tabela 1. Os formatos de ambientes *fulldome* variam em diversos aspectos. Podem ser projetados para apenas um espectador ou centenas; a projeção pode ser multicanal ou canal único, a depender da quantidade de projetores que compõe a imagem no domo. Esses também podem ser capazes de trabalhar com estereoscopia ou não.

Tabela 1 – Documentos encontrados nas bases científicas com palavra-chave fulldome.

Base Científica	**Número de artigos encontrado com palavra *fulldome***
Springer	31
IEEE Xplore	4
Scopus	25
ACM Digital Library	8

Fonte: Autoria Própria, consultas feitas em 04/01/2019

Historicamente, ao redor do mundo, houve poucos ambientes que foram devidamente desenvolvidos para receber projeções panorâmicas, como por exemplo *La Géode* em Paris, França (FELINTO; ZANG; VELHO, 2012). A tecnologia de exibição de vídeo rasterizado de grande escala originou-se em sistemas simuladores de treinamento militar, migrando depois para planetários em meados da década de 1990 (LANTZ, 2007). As inovações na projeção digital na última década por parte de grandes empresas tal como *IMAX* alimentaram uma explosão de cinemas digitais em grande escala.

Displays imersivos podem ser divididos em três categorias gerais:

Os monitores para um único usuário, de pequena escala, incluem visores montados na cabeça, monitores de mesa e pequenos domos de projeção pessoal (LANTZ, 2007).

Os monitores de escala média fornecem colaboração para dezenas de usuários e incluem telas imersivas retilíneas, telas curvas e grande paredes de projeção multicanal, todas acomodando um pequeno número de usuários colaborativos (LANTZ, 2007). Um exemplo amplamente usado nessa categoria são os ambientes *CAVE*. As *CAVEs* foram desenvolvidas para experiência imersiva em Realidade Virtual. Normalmente aplicações em *CAVEs* utilizam *trackers* para rastrear a posição da cabeça do usuário com 6 graus de liberdade (DOFs) para navegar dentro do ambiente virtual e interagir com o conteúdo (LI et al., 2014).

Visores imersivos de grande escala empregam telas de campo de visão ampla, como cúpulas, para fornecer uma alta sensação de imersão para centenas de espectadores (LANTZ, 2007).

Telas cilíndricas ou cúpulas são preferíveis a telas retangulares imersivas em aplicações cinematográficas, pois proporcionam uma aparência mais perfeita em uma maior faixa de ângulo de visão (LANTZ, 2007).

Em comparação com as instalações tradicionais de treinamento em realidade virtual, como as *CAVEs*, o *fulldome* é capaz de hospedar um grupo maior de espectadores e reduzir o custo de seções per capita, além de realizar sessões colaborativas com vários espectadores (LI et al., 2014).

Uma configuração de projeção *fulldome* de canal único é caracterizada por possuir apenas um projetor. Este fica centrado ao meio da sala, apontando para cima. Normalmente o projetor é equipado com uma lente grande angular que "espalha" a imagem de modo a cobrir todo o domo. Tem-se este tipo de configuração para pequenos domos, não mais que dezenas de espectadores. Ambientes *fulldome* educacionais, planetários portáteis e não cinematográficos tendem a usar a abordagem de um único projetor com lente *fish eye*, pois são mais baratos e mais simples de manter (LANTZ, 2007).

A configuração multicanal é feita a partir da composição de imagens por mais de um projetor, podendo variar de um par de projetores ou mais. Sistemas multicanal também possibilitam visualização estereoscópica. Configuração de múltiplos projetores é comumente empregada em salas fulldome de grandes teatros.

Um outro método de projeção *fulldome*, descrito por Paul Bourke é feito com uso de um espelho convexo (BOURKE, 2005). O ambiente é formado por um espelho, uma cúpula e um projetor.

A Figura 1 mostra o método proposto por Bourke, onde temos um domo A que fica

posicionado na vertical e possui uma abertura no ponto B por onde passa a projeção do projetor C que fica atrás do domo. A projeção de C reflete no espelho convexo D o qual faz a imagem projetada voltar para a parte interna do domo A. Projetar uma imagem neste espelho esférico gera um efeito similar a uma lente grande angular, aumentando o ângulo de abertura da projeção. A diferença é que o espelho deixará a imagem invertida.

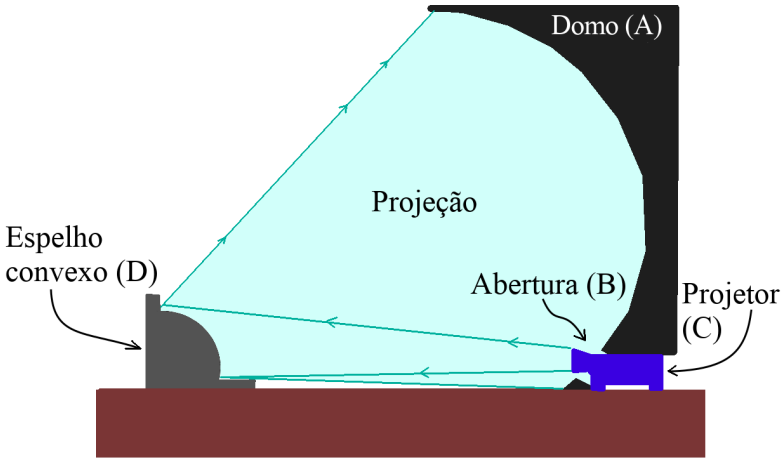


Figura 1 – Demonstração do sistema de projeção com espelho esférico proposto por Bourke.

Fonte da ilustração: Autoria própria.

2.1.2 Aplicações em *Fulldome*

Os cinemas *fulldome* hospedam uma variedade de aplicações, incluindo reprodução linear de animações ou filmes, simulação em tempo real, cinematografia ao vivo e aplicações interativas de audiência (LANTZ, 2007).

Ambientes *fulldome* normalmente têm carência de variedade de conteúdo para apresentação (TREDINNICK; RICHENS, 2015). Apesar de existir conteúdo substancial para a plataforma *fulldome*, a grande maioria se limita a filmes, animações e apresentações pré-programadas retilíneas, sem interação por parte do espectador e voltadas para o contexto da astronomia (vício de origem). A razão de haver tanto conteúdo da área astronômica para *fulldome* é por eles serem plataformas cinematográficas adaptadas de planetários. Conteúdo interativo para *fulldome* é incomum, pois não existem ferramentas profissionais apropriadas para produção desse tipo de material.

Apesar disso, existem exemplos no meio acadêmico de aplicações interativas feitas para *fulldome*:

AstroSurf (GODDARD et al., 2016) é um jogo para plataforma *fulldome* que faz uso de sistema multicanal de 8 projetores. Os usuários conseguem manipular os elementos na tela por meio de controles de *Xbox*. Apesar de haver interatividade entre os usuários e o ambiente, o jogo tem um caráter retilíneo com duração fixa. O jogo usa a *game engine Unity 3D* para fazer a renderização da cena.

O *Mine Coaster* (LI et al., 2014) é outro jogo feito para plataforma *fulldome* multicanal. Foi criado para mostrar que a plataforma que os autores desenvolveram servia tanto para *story telling* quanto para jogos interativos. A interação dos usuários é feita por *trackers*, que ajudam na captura de movimentos com câmeras especiais que triangulam suas posições.

Dentre estas aplicações, temos também ferramentas com contexto mais científico e voltado para a astronomia, tal como os *softwares* de planetários. São exemplos destas aplicações os softwares *Starry Night*, *Stellarium*, *DigitalSky 2* da *Sky-Skan* e *Digistar 3* da *E&S* (LANTZ, 2007).

2.1.3 Softwares para fulldome

Os *softwares* comerciais para *fulldome* ainda representam um universo restrito de aplicações, normalmente voltadas para um nicho de reprodução de vídeo. Alguém que desejar desenvolver aplicações, que fujam do contexto de reprodução de vídeo, terá que adaptar algum *software* existente para a plataforma, ou até mesmo desenvolvê-lo do início. É possível desenvolver o *middleware* usando motores de jogos profissionais consolidados tal como a *Unity* ou a *Unreal*. Em se tratando de jogos para computador, esses motores de jogos são de fato ideais. Entretanto, aplicações científicas, como visualização de volumes adquiridos com ressonâncias magnéticas ou radiografias, demandam eficiência do uso de recurso computacional por parte do *software*, que por sua vez deve ser otimizado para determinada tarefa, o que implica em modificações no código fonte do próprio motor de jogo/gráfico.

Esse trabalho demonstrará na subseção 4.2.1 uma aplicação para visualização de modelos volumétricos reconstruídos a partir de imagens do projeto do *Visible Human Project*. Tal aplicação não é algo trivial de se implementar. A mesma deve ser feita com tecnologia adequada, que faça uso de maneira eficiente e otimizada dos recursos computacionais, como a memória *RAM*, pois do contrário seu funcionamento é inviável. Usamos esse exemplo de aplicação como uma forma de demonstrar que o sistema proposto é eficiente a ponto de se conceber tal aplicação.

Com *softwares* atualmente disponíveis no mercado para *fulldome* ou adaptando motores de jogos profissionais para tal, é possível reconstruir uma representação dos modelos tridimensionais para anatomia, usando por exemplo o *Digital Sky 2*[1]. Softwares similares ao *Digital Sky 2*, que são específicos para planetários, além de possuírem um conjunto de ferramentas específicas para apresentações no contexto da astronomia, possibilitam carregar material auxiliar tal como *slides*, vídeos, modelos tridimensionais etc. Eles não são otimizados para carregar, tomando como exemplo nossa aplicação, mil ou até mesmo quatro mil imagens de alta resolução. Essas funcionalidades no *Digital Sky 2*, *Digi Start*

[1] <https://www.skyskan.com/products/ds> visto em 07/01/2018

3[2] assim como ferramentas similares, são apenas de cortesia e possuem um limite de uso. O mesmo acontece para *game engines* comerciais, normalmente baseadas em editores de cena, onde carregar mil elementos na cena, muitas vezes trava a aplicação. Nesse sentido é obrigatório desenvolver uma ferramenta à parte, especificamente projetada para suportar o alto volume de dados que nossa aplicação irá processar.

2.2 Considerações

Sobre a configuração de canal único, a qual usa lente grande angular ao centro do domo, uma das vantagens é que o *form factor* da imagem não necessita de uma grande adaptação, pois o mesmo é o "*domemaster*", padrão utilizado por produtores de conteúdo *fulldome*.

Outra vantagem é que esse tipo de configuração permite uma maior mobilidade, onde domos infláveis podem ser utilizados. A desvantagem é que o projetor se situa ao centro do domo, onde pode vir a ocupar eventual espaço dos espectadores. O centro do domo é o melhor local para observação, pois é onde menos se percebe a distorção da imagem.

Tomando como referência os sistemas *fulldome* multicanal que compõem múltiplas projeções de imagens para formar a imagem final no domo todo, a grande vantagem dessa configuração é que a composição de projetores resulta em uma imagem com maior resolução. Ideal para ambientes grandes com capacidade de comportar centenas de espectadores. A utilização de mais projetores permite maior concentração de luz por área, resultando em uma maior luminosidade na composição final da imagem. Configurações multicanal permitem implementar visualização estereoscópica. Em sistemas com capacidade estereoscópica é possível sobrepor a mesma imagem para intensificar a luminosidade. A desvantagem de sistemas multicanal é que estes ambientes, ao contrário do de canal único, são menos portáteis, normalmente empregados em projetos de salas fixas. Outro eventual problema é o preço, tanto para aquisição de múltiplos projetores quanto para a sua manutenção, que normalmente são projetores industriais caros se comparado aos projetores para uso comum ou de escritório.

Com relação ao método de projeção proposto por Bourke, utilizando o espelho convexo para espalhar a imagem no domo, julga-se menos complexo, comparado aos demais, pois não faz uso da lente grande angular. É uma opção mais atrativa pelo fato de economizar com lentes e projetores profissionais que tenham capacidade de trabalhar com lentes especiais, pois nem todos permitem troca das lentes. Isso acaba por tornar esse método mais acessível pelo custo reduzido se comparado com todos os outros. O problema é que o uso do espelho não distribui a luz com tanta intensidade quanto com o uso de

[2] <https://www.es.com/Digistar/> visto em 07/01/2018

uma lente grande angular. A imagem final fica com menos brilho. A distorção no espelho também é maior se comparada ao da lente grande angular. Normalmente é ideal para domos de pequena ou média escala, podendo acomodar de uma até dez pessoas, não muito além disso.

Com relação à estereoscopia em ambientes *fulldome*, existe uma dificuldade em gerar imagens estereoscópicas para os mesmos. Como sabemos, a tela cobre os 360º em torno do espectador. Em uma projeção estereoscópica, caso feita simplesmente capturando imagens *fish eye* em dois pontos no espaço, simulando a paralaxe não é o suficiente, pois, assim que observado em 180º, essas imagens se invertem na perspectiva do observador. Ou seja, o canal de visualização esquerdo passa a ser visto com o olho direito e vice-versa, o que acaba quebrando a imersão.

A técnica para geração da imagem verdadeiramente estereoscópica omnidirecional deve ser feita "pixel-a-pixel" como descrito por Bourke (BOURKE, 2009). No método descrito por Bourke, a aplicação não tinha caráter de renderização em tempo real. É possível implementar o método de Bourke a nível de aplicação em tempo real, porém é necessária programação com *shaders* gráficos *GLSL* ou *HLSL*, onde a geração das imagens seria feita inteiramente pelo *shader*. *Shader* é um programa que é executado no *pipeline* de renderização gráfica de modo a alterar como são desenhados cada pixel. Programação com *shaders* por sua vez possui uma curva de aprendizado extensiva. *É um estudo específico, que desenvolvedores de conteúdo sentem dificuldade para entender*[3].

Alguns dos problemas mencionados quanto ao *fulldome* são o vício de origem e a falta de conteúdo alternativo para a plataforma. O vício de origem trata-se do conteúdo relacionado ao contexto de astronomia, visto que *fulldome* é adaptação do cinema em planetário. O mesmo tem carência de conteúdo multimídia interativa, que os filmes retilíneos não suprem. O conteúdo para *fulldome* remete muito a animações geradas por computador, ou filmes feitos com câmeras *fish eye*, não havendo qualquer interação por parte do espectador. Apesar de incomum, foram mostrados alguns exemplos de aplicações acadêmicas implementadas para *fulldome* que possuem interatividade (TREDINNICK; RICHENS, 2015; LI et al., 2014; GODDARD et al., 2016). O problema, no entanto, é que não existe apenas uma configuração para disposição de projetores como é o caso dos sistemas *fulldome* multicanal.

Tomando como referência o jogo para *fulldome* AstroSurf (GODDARD et al., 2016), apesar de o sistema ser multicanal e conter 8 projetores, a disposição da perspectiva de projeção é fixa para o sistema que eles definiram. Os autores geram um mosaico com todas as visões que cada projetor deve mostrar. Essa perspectiva de cada projetor é pré-definida e fixa. Não é esperado comutação da posição entre estes projetores, nem se pode alterar a

[3] <https://www.reddit.com/r/opengl/comments/2gh77v/help_am_i_so_stupid_or_its_really_hard_to/> visto em 30/12/2018

rotação ou transladar qualquer projetor. A distribuição das imagens é feita via hardware específico, que canaliza cada uma das imagens do mosaico para o seu respectivo projetor. Na nossa proposta o sistema é independente do número de projetores e da disposição dos mesmos, sendo possível mapear tudo. Em nenhum dos estudos citados, os autores de trabalhos para aplicações fulldome propuseram até então uma maneira de resolver a calibração dos projetores para diferentes configurações de ambientes. Nenhum dos estudos citados que desenvolveram uma aplicação *fulldome*, fizeram-no de maneira genérica. Ou seja, são aplicações bem definidas, inflexíveis e sem versatilidade. O *middleware* descrito neste trabalho serve como alicerce para desenvolver uma aplicação *fulldome*, não uma aplicação fixa.

O sistema proposto pode trabalhar com renderização distribuída em múltiplos computadores, similar ao sistema *DIVERSE* (KELSO et al., 2002) que funciona com projeção de cena em sistemas distribuídos. Essa modalidade de aplicação de renderização gráfica distribuída, se mostrou contra produtiva, visto que implica em uma complexidade excessiva para produção de conteúdo. Existem problemas ao se desenvolver aplicações gráficas de maneira distribuída, por exemplo, dessincronização entre projeções; múltiplos contextos e estados de variáveis de bibliotecas que compõem o sistema; impossibilidade de se integrar sistemas externos como navegadores de *internet* (subseção 4.1.6), etc. Essa abordagem cria dificuldades para desenvolvedores de conteúdo. Por esse motivo é que ambientes com arquiteturas similares a essa mostraram-se um desafio fora do comum aos desenvolvedores de conteúdo, os quais, como visto até então, se limitaram a produzir conteúdo simples, como vídeos *fulldome* e apresentações lineares. A solução para esse cenário, que é o da Arena Digital na PUCPR, seria usar uma arquitetura mais simples, que poupa o desenvolvedor de conteúdo de trabalhar com complexidades desnecessárias introduzidas com a abordagem distribuída.

Apesar deste trabalho suportar arquiteturas distribuídas, estamos propondo uma arquitetura que centraliza a renderização em uma única máquina (subseção 4.1.1). Ela ainda permite que o sistema antigo (distribuído), coexista com o sistema novo, permitindo dessa forma, a manutenção do software já empregado pela instituição, seja pelo custo de investimento, mas também por agregar valor ao continuar prover soluções sofisticadas já existentes tais como o *software* de visualização astronômica e o *software GIS*, os quais já servem de maneira satisfatória os usuários.

No próximo capítulo será apresentada a metodologia, que está dividida em duas partes. Primeiramente são apresentados os materiais utilizados na pesquisa. Em seguida, a classificação do método utilizado e o delineamento da pesquisa.

3 Metodologia e infraestrutura para o desenvolvimento da pesquisa

Neste capítulo são apresentados o detalhamento da infraestrutura necessária para o desenvolvimento da pesquisa, a classificação da pesquisa e seu delineamento.

3.1 Infraestrutura de pesquisa

O material necessário para a elaboração do sistema consiste no uso do teatro digital *fulldome* que se encontra no campus da PUCPR Curitiba; as bibliotecas e componentes de softwares citados anteriormente, assim como ambiente de desenvolvimento C++ e *Go-Lang*[1]. Para algumas aplicações de demonstração do *middleware* empregou-se a base de imagens do *Visible Human Project* e outra base feita com o escâner *F2S2* (SILVA et al., 2018).

3.1.1 Teatro Fulldome na PUCPR

O Teatro digital *fulldome* chamado de Arena Digital, localizado na PUCPR Curitiba, foi inaugurado em abril de 2013. A construção de visual moderno foi projetada pelo arquiteto *Manoel Coelho*[2]. A construção tem aparência baseada em cilindro que *se expande em movimento espiral, semelhante a uma galáxia*[3]. Possui dois pisos: o térreo, onde se encontra a bilheteria, cafeteria, lounge e o piso superior que fica o teatro.

O projeto técnico do teatro foi realizado pelas empresas *OmnisLux* e *Sky-Skan*. O sistema é composto por quatro projetores *led* de alta-resolução de 2900 lúmens *Barco F32 series*[4] com tecnologia de processamento digital da luz (*Digital Light Processing - DLP*). Uma característica importante dessa tecnologia é que ela permite que a luz projetada possa ser separada em frequências específicas. O projetor pode, por exemplo, ser configurado para emitir o canal de luz vermelha entre 430 até 480 terahertz (THz). A abordagem para visualização estereoscópica é relativamente recente e foi desenvolvida pela *INFITEC*[5] que fez uso da tecnologia *DLP* para posteriormente visualizar estereoscopia pelo método de filtro por interferência. O teatro também possui um sistema de som de 13.000 *Watts* integrado com o equipamento de vibração das 120 poltronas. Esse aspecto caracteriza o teatro como sendo 4D, pois trabalha os sentidos do usuário além da visão e áudio.

[1] <https://golang.org/> visto em 12/01/2020
[2] <http://www.mcacoelho.com.br/> Acessado em 31/12/2018.
[3] <http://www.mcacoelho.com.br/?page_id=49> Acessado em 03/03/2020.
[4] <https://www.barco.com/en/product/f32-series> Acessado em 05/03/2020.
[5] <http://infitec.net> Acessado em 31/12/2018.

A Figura 2 mostra um esquema simplificado do sistema de projeção da Arena Digital. Ele é controlado por seis computadores, sendo um computador dedicado para cada um dos quatro projetores, um para controlar o sistema de som e um para o operador do teatro. Cada um destes computadores é equipado com: 12 GB de memória *RAM*, 1 placa gráfica *NVIDIA Quadro 4000* (486 *Gigaflops* de capacidade de processamento) e 1,2 TB de armazenamento. O sistema é gerenciado por softwares como *VNC* e o próprio software da *Sky-Skan* chamado de *Digital Sky 2*.

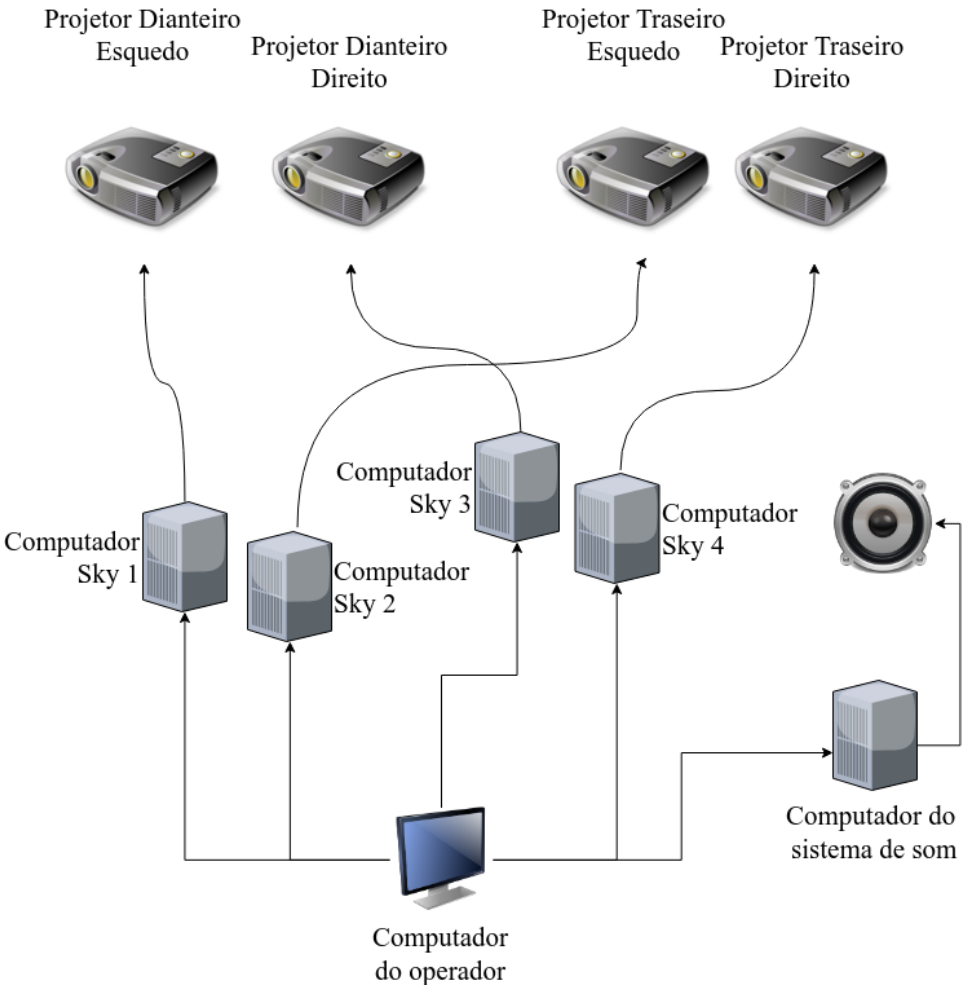


Figura 2 – Diagrama simplificado dos equipamentos da Arena Digital.

Fonte: Autoria própria.

A Figura 3 é uma fotografia feita dentro na Arena Digital. Em primeiro plano, o terminal principal do operador e ao fundo, o domo, com a imagem de uma cena sendo projetada.

A estereoscopia do ambiente é feita usando a tecnologia de processamento digital da luz, que basicamente possibilita o projetor separar cada um dos três canais de cores (vermelho, verde e azul), em faixas de frequências diferentes. Os espectadores então usam óculos onde uma lente filtra para um olho os três canais de cores em faixas de frequências

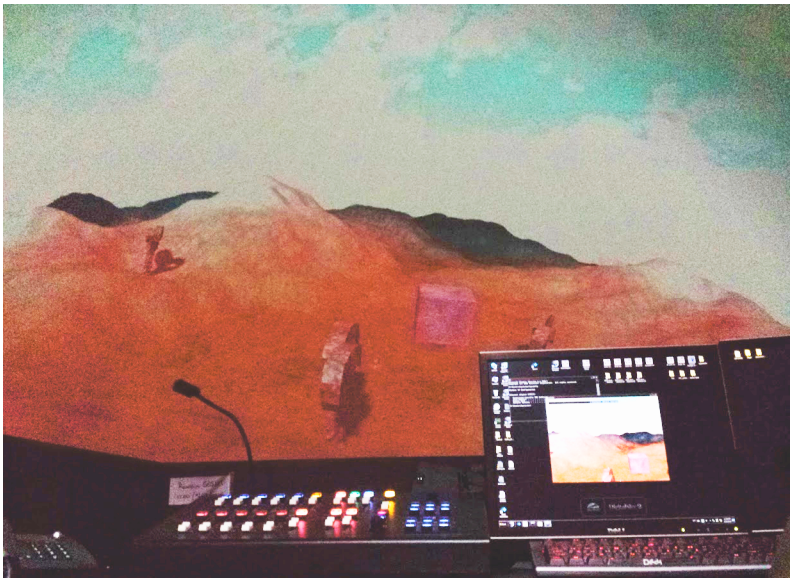


Figura 3 – Fotografia no interior da Arena Digital.

Fonte da ilustração: Autoria própria.

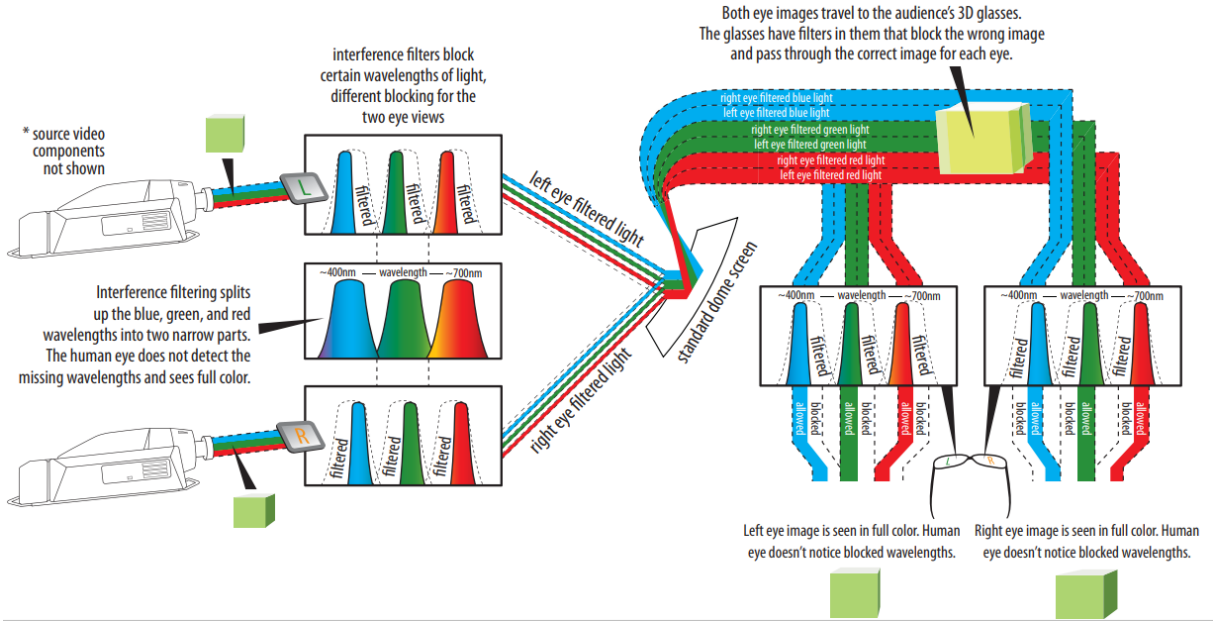
mais altas e a outra lente, filtra para o outro olho, os três canais de cores em intervalos mais baixos (SIMON et al., 2010), permitindo assim, ver uma imagem distinta para cada olho. O fato de as imagens serem projetadas com os três canais de cores gera consequentemente menos aberração de cores durante a visualização estereoscópica, sendo mais agradável do que no método anáglifo convencional (vermelho e azul). A Figura 4 ilustra esse processo de visualização estereoscópica, através do uso da tecnologia de processamento digital da luz.

3.1.2 Componentes a serem utilizados

Será utilizada a *Urho3D*[6], como componente base para desenhar gráficos e demais recursos de mídia interativa. Trata-se de uma *game engine*[7] de alta performance de código aberto que faz uso de tecnologias modernas se comparado com as alternativas comerciais atuais tal como as *game engines Unity*[8] e *Unreal*[9]. Algumas das características da *Urho3D* são: método de sombreamento como *CSMs* (*Cascaded Shadow Maps*[10], método para sombreamento de cenas complexas com elevado número de modelos tridimensionais); animação de geometrias a partir da *GPU* (*hardware skinning animation*[11]); renderização *HDR* (*High-Dynamic-Range*[12], pré-processamento da imagem que produz um melhor

[6] <https://urho3d.github.io/> visto em 30/01/2019
[7] <https://unity3d.com/what-is-a-game-engine> visto em 30/12/2018
[8] <https://unity3d.com/de> visto em 30/01/2019
[9] <https://www.unrealengine.com/en-US/what-is-unreal-engine-4> visto em 30/01/2019
[10] <https://developer.download.nvidia.com/SDK/10.5/opengl/src/cascaded_shadow_maps/doc/cascaded_shadow_maps.pdf> visto em 21/01/2019
[11] <http://developer.download.nvidia.com/SDK/10/direct3d/Source/SkinnedInstancing/doc/SkinnedInstancingWhitePaper.pdf> visto em 21/01/2019
[12] <https://www.sony.de/electronics/support/articles/00163663> visto em 30/01/2019

Figura 4 – Tecnologia de processamento digital da luz existente nos projetores que a Arena Digital utiliza.



Fonte: https://www.skyskan.com/files/documents/definiti_3d-skyskan_data_sheet_print.pdf

contraste e uma gama mais ampla de cores e brilho, o que torna a imagem final mais realista e imersiva). A *Urho3D* possui suporte para linguagens de *shaders HLSL*[13] e *GLSL*[14], capacidade de trabalhar com drivers gráficos *Direct3D*[15] 9, *Direct3D11*, *OpenGL*[16] 2.0, 3.2, *OpenGL ES*[17] 2.0 para plataformas móveis e *WebGL* v1.4 para plataformas *web* (navegadores *web*). A *Urho3D* é considerada uma *game engine* pois além de módulo para gráficos, ela possui módulos para áudio, rede, simulação de física e até mesmo algoritmos básicos de inteligência artificial.

A *Urho3D* pode ser compilada e utilizada nas plataformas *desktop Windows* (Família *New Technology* a partir do *Windows 7* ou superior) *Mac OS* (*OS X Mountain Lion 10.8* ou superior), *iOS*, *Android*, *tvOS* e até mesmo *single boards* baseados em arquitetura *ARM*[18] tais como *Raspberry Pi*[19] e *ODROID*[20].

Há a necessidade de realizar comunicação paralela entre os computadores do usuário e do ambiente *fulldome*, seja para iniciar a *engine* ou para interagir com a aplicação. Para

[13] <https://docs.microsoft.com/en-us/windows/desktop/direct3dhlsl/dx-graphics-hlsl> visto em 30/01/2019
[14] <https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language> visto em 30/01/2019
[15] <https://docs.microsoft.com/en-us/windows/win32/getting-started-with-direct3d?redirectedfrom=MSDN> visto em 13/01/2020
[16] <https://www.khronos.org/opengl/> visto em 30/01/2019
[17] <https://www.khronos.org/opengles/> visto em 30/01/2019
[18] <https://www.techopedia.com/definition/5900/advanced-risc-machine-arm> visto em 30/01/2019
[19] <https://www.raspberrypi.org/> visto em 30/01/2019
[20] <https://www.hardkernel.com/> visto em 30/01/2019

que isso ocorra, usamos outras tecnologias auxiliares além da *Urho3D*. Por isso é importante a *Urho3D* ser capaz de se integrar a nível de código nativo com outros programas, o que não é facilmente feito com a *Unity* e *Unreal*. Escrevemos um *middleware* na linguagem de programação *Go*[21] e integramos diretamente com a parte da aplicação escrita em C/C++ (*Urho3D*). Go é uma tecnologia moderna e mantida pela Google que possui muitas conveniências nativamente. Existem muitas bibliotecas de código aberto disponíveis no *Github*[22] para esta linguagem. Go é versátil e gera código seguro, especificamente para produzir *API*s de serviço *web* e principalmente, trabalhar bem com concorrência de requisições e *multithreading*. É por meio do módulo escrito em *Go* que se inicia o sistema, se alterna entre as aplicações, executam-se comandos de baixo nível e media-se a interação com o usuário.

Tabela 2 – Teste de desempenho entre *engines*.

Engine	**Número imagens**	**Uso RAM *megabytes***	**FPS**
Irrlicht	1000	58.9	60+
Irrlicht	4000	64.4	38
Irrlicht	27000	93.1	23
Unity	1000	53.0	60+
Unity	4000	60.0	60+
Unity	8000	80.0	60+
Urho3D	1000	35.0	60+
Urho3D	4000	43.8	60+
Urho3D	27000	118.6	60+

Fonte: Autoria Própria
FPS: *Frames per second*, do inglês, quadros por segundo. Significa o número de atualizações em um intervalo de um segundo.

A Tabela 2 mostra os resultados de um teste de desempenho obtido entre as *engines* consideradas para uso do desenvolvimento do *middleware*. Foi medido o uso de memória pela aplicação e a média de taxa de atualização que as *engines* mantiveram por dois minutos, considerando amostras em intervalos de dez segundos. O teste consiste em carregar vários objetos (modelos tridimensionais de cubos) idênticos com uma mesma imagem/textura na superfície de cada um desses objetos. A quantidade de objetos carregados foi mil, quatro mil e vinte e sete mil, em referência ao exemplo de aplicação descrito na subseção 4.2.1, que faz uso da base de imagens do *Visible Human Project*. A quantidade de imagens para o modelo masculino de 1994 tem aproximadamente mil imagens, todas elas de dimensões 1920 *pixels* de largura por 1080 *pixels* de altura, no formato de arquivo *JPEG*. A base do

[21] <https://golang.org> visto em 12/02/2020
[22] <https://github.com/> visto em 08/03/2020

modelo feminino de 1995 tem quatro mil e a recente base feita com outro modelo feminino em 2015 tem aproximadamente vinte e sete mil imagens. Houve uma exceção no caso da *Unity*, em que o editor da *game engine* não permitiu carregar mais de oito mil instâncias de objetos tridimensionais, não sendo possível atingir 27000, como ocorreu para as outras duas *engines*.

Além das tecnologias citadas até então, outros componentes tecnológicos complementares deverão ser empregados para formar o restante do sistema tal como a biblioteca *Chromium Embedded Framework (CEF)*[23], bibliotecas auxiliares em *GoLang* etc.

3.1.3 Banco de Imagens Anatômicas

Em um dos exemplos implementados para demonstração do sistema proposto, foi desenvolvida uma aplicação para a visualização de volume a partir de imagens do projeto do *The Visible Human Project*.

O banco de imagens do *The Visible Human Project*, fornecido para a PUCPR por meio de uma licença da Biblioteca Nacional de Medicina dos Estados Unidos da América (*NLM*) dispõe de conjuntos de imagens de tomografia, ressonância magnética e fotografias das secções criogênicas de cadáveres humanos reais (ACKERMAN, 1997; ACKERMAN; YOO; JENKINS, 2001). O projeto realizou imagens de ressonância, tomografia e fotografias das secções criogênica em dois cadáveres sendo um masculino e outro feminino. O cadáver masculino, é de um detento do estado do Texas condenado a morte, que doou seu corpo para a ciência antes de ser executado. Os cortes criogênicos do corpo do espécime masculino foram feitos em intervalos de 1 milímetro totalizando 1871 imagens. As imagens foram convertidas do filme de setenta milímetros para arquivos digitais de 4096 pixels por 2700 pixels (maior que 4K). O espécime feminino, era de uma cidadã norte-americana que morreu de infarto cardiovascular. Seu corpo foi doado pelo marido. As imagens das secções criogênicas no espécime feminino foram feitas em intervalos de 0.33 milímetros, totalizando 5189 fotografias posteriormente convertidas para imagens digitais de mesma resolução ao do modelo masculino. O fato do projeto do *Visible Human Project* fotografar secções da cabeça aos pés torna-o especial, pois é possível, a partir destas imagens, reconstruir o espécime virtualmente.

3.1.4 Ambiente Computacional de Desenvolvimento

Neste trabalho foi utilizado o compilador *GCC*[24] 6.3, para as linguagens de programação C/C++ e o conjunto de ferramentas para *Go 1.11*[25]. As bibliotecas C/C++ que alicerçam o principal módulo do *sistema* são: *Urho3D* como motor de jogo, o *middleware*

[23] <https://bitbucket.org/chromiumembedded/cef> visto em 12/01/2020
[24] <https://gcc.gnu.org/> visto em 30/01/2019
[25] <https://golang.org/doc/go1.11> visto em 30/01/2019

escrito em *Go* para tratar principalmente comunicação em rede, a biblioteca para interpretação de *JSON* em C++ *nlohmann/json*[26] assim como o *Chromium Embedded Framework (CEF)*[27]. Para o *middleware* em Go, foram mais de dez componentes utilizados. O mais relevante entre eles é o *Gin*[28], usado para tratamento paralelo de requisições no protocolo *HTTP/HTTPS*.

A construção do projeto é totalmente automatizada com *scripts CMake*[29], que trabalham de maneira integrada com a *IDE* de desenvolvimento. O ambiente de programação utilizado é o *Microsoft Visual Studio Code*, que é bastante flexível para funcionar com diversos compiladores além de possuir *plugins* para ajudar no desenvolvimento. Além de ser um bom ambiente de desenvolvimento para *Go*, funciona bem com o compilador *GCC* para C/C++ e gerador de projetos *CMake*. Até 2020, *o Visual Studio Code consta como uma boa IDE de desenvolvimento*[30].

3.2 Classificação e delineamento da pesquisa

Nesta seção é apresentada a classificação da pesquisa quanto a sua natureza; seu objetivo e seus procedimentos.

3.2.1 Classificação da pesquisa

De acordo com Jung (JUNG, 2004) na Figura 5, a pesquisa quanto a sua natureza é pesquisa aplicada, pois objetiva a aplicação dos conhecimentos básicos na geração de novos produtos, processos, patentes e serviços. Quanto aos objetivos, a pesquisa exploratória visa à descoberta de teorias e práticas que modificarão as existentes e estudos iniciais. Quanto aos procedimentos enquadra-se como uma pesquisa experimental, pois busca a descoberta de novos materiais, métodos, técnicas, protótipos de software - ensaios e estudos de laboratório, modelagem, simulação e circuitos etc. Finalmente é uma pesquisa em laboratório que é fundamentalmente a última etapa de todo o processo metodológico. Entende-se como pesquisa em laboratório aquela onde ocorre a possibilidade de se controlar as variáveis que possam intervir no experimento.

3.2.2 Delineamento da pesquisa

Nesta seção é apresentado o modelo proposto para dar resposta ao objetivo geral que é desenvolver um middleware capaz de gerar aplicações interativas para plataformas

[26] <https://github.com/nlohmann/json> visto em 12/01/2020
[27] <https://bitbucket.org/chromiumembedded/cef> visto em 12/01/2020
[28] <https://github.com/gin-gonic/gin> visto em 13/01/2020
[29] <https://cmake.org/> visto em 30/01/2019
[30] <http://www.infigic.com/blog/best-web-development-ide/> visto em 13/01/2020

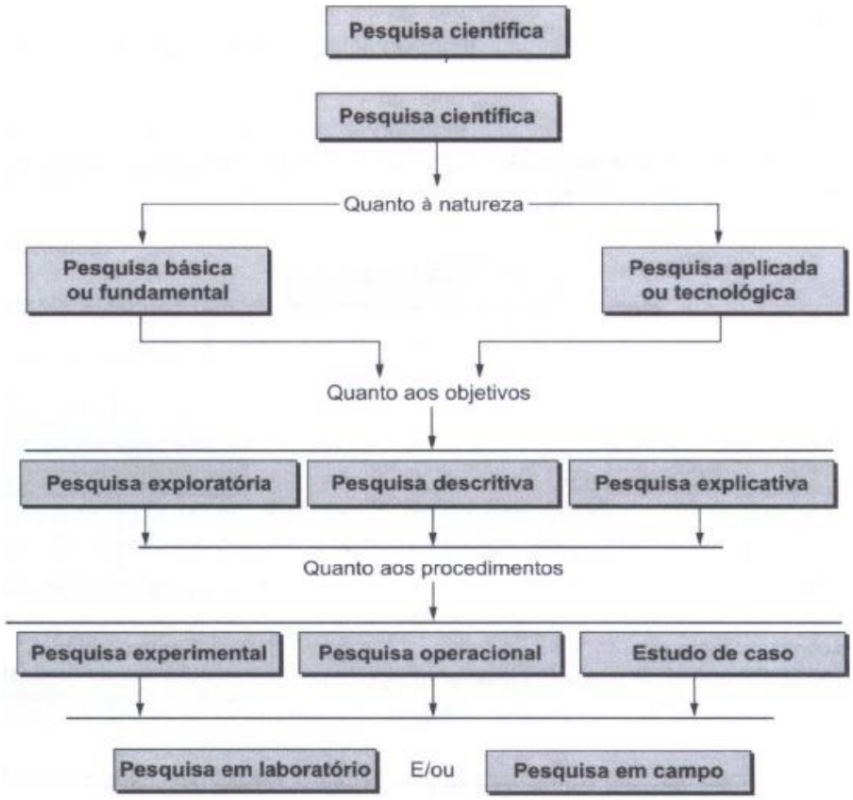


Figura 5 – Tipos de pesquisas científicas.

Fonte: Jung (JUNG, 2004).

imersivas, principalmente *fulldome*, sendo capaz de reproduzir vídeos imersivos, diretamente da *internet*, para o *fulldome*.

Para atender ao objetivo geral foi proposto um modelo que está alinhado com os objetivos específicos detalhado na seção 1.2.2.

O modelo proposto é composto por seis etapas:

Tabela 3 – Etapas do modelo proposto.

Etapa 01 - Arquitetura de infraestrutura
Etapa 02 - Construção do middleware
Etapa 03 - Procedimento para gerar imagens fulldome
Etapa 04 - Calibragem de projeções fulldome
Etapa 05 - Manipulação de Janelas
Etapa 06 - Integração com plataformas da internet

Fonte: Autoria Própria

Na seção 4.1 é apresentado o detalhamento das etapas do modelo proposto.

4 Modelo proposto e Resultados experimentais

Neste capítulo é apresentado o modelo proposto para a solução do problema apresentado, que é composto de seis etapas. Na sequência são apresentados alguns exemplos de aplicação para visualização volumétrica e de aplicação para visualização de objeto de estudo virtual. Algumas considerações sobre o modelo e finalmente são apresentados os resultados experimentais.

4.1 Modelo Hornung

4.1.1 Arquitetura de infraestrutura

Esta seção define como resolver o primeiro objetivo específico. Tratando da arquitetura de equipamentos, atualmente empregada na Arena Digital na PUCPR, fizemos as seguintes pontuações. Mesmo tendo implementado a estratégia de renderização distribuída no sistema proposto, notou-se que um dos motivos pelo qual esse ambiente carece de conteúdo é a arquitetura (de equipamentos) corrente. Desenvolvedores de conteúdo mostram-se descontentes com a dificuldade de trabalhar com ambientes distribuídos. Quando o projeto do sistema foi feito pela *Sky-Skan* (2010-2012), talvez houvesse sentido em distribuir a renderização entre múltiplas máquinas, de modo a garantir melhor desempenho com vídeos de alta resolução, visto que essa tecnologia estava no início de seu uso na época. Mas, atualmente, não é raro encontrarmos por preços acessíveis ao consumidor comum, placas de vídeo que tenham mais de 4 saídas de vídeo, cada uma suportando resoluções de $4K$ ou mais. Sendo assim, não há razão em persistir com essa arquitetura, visto que ela apenas prejudica desenvolvedores de conteúdo e está se tornando obsoleta.

A arquitetura nova que este trabalho propõe, tem o objetivo de simplificar o sistema atual, centralizando toda a aplicação em apenas um único computador. Onde atualmente, tem-se 6 máquinas, uma para cada projetor, uma para o som e mais uma para coordenar o conjunto, será feita a substituição por apenas uma máquina com placa de vídeo de quatro ou mais saídas de vídeo de até 4k de resolução. Dessa maneira, não apenas amenizamos os custos com manutenção de seis máquinas, como também reduzimos o espaço ocupado por tal infraestrutura. O restante da infraestrutura, além da máquina principal e dos projetores, é composto por um equipamento para roteamento de rede (roteador), com finalidade de conectar múltiplos usuários ao sistema. A Figura 6 mostra um esquema simplificado de como ficariam os principais componentes dessa nova arquitetura. Para entender melhor a diferença, basta comparar a Figura 6, que mostra a proposta de nova arquitetura, com a

Figura 2, que mostra a arquitetura atual.

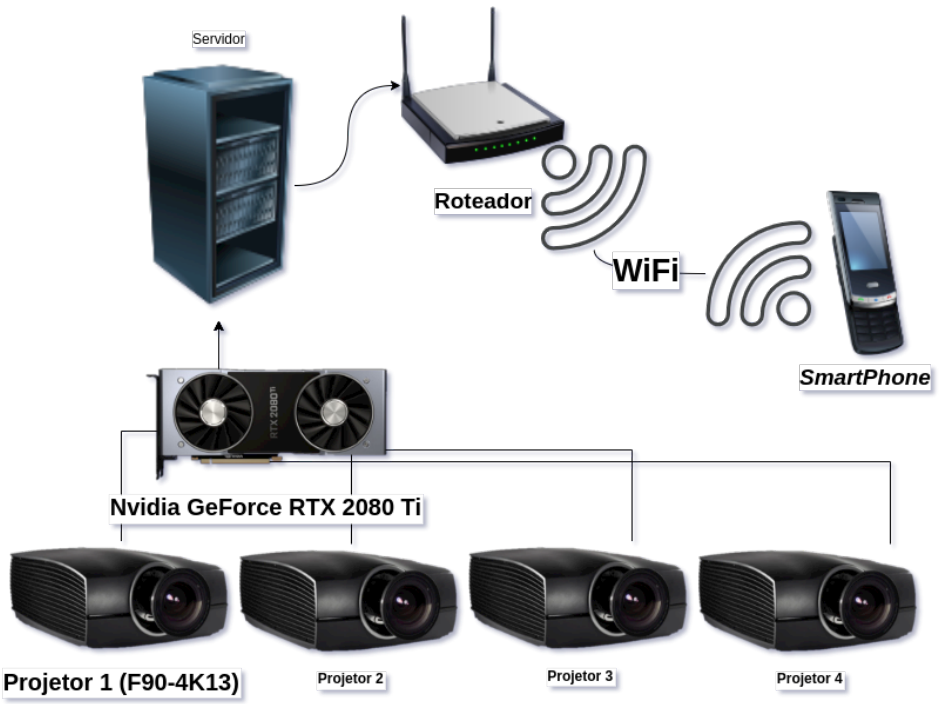


Figura 6 – Esquema simplificado da proposta de uma arquitetura mais simples para a Arena Digital da PUCPR.

Fonte: Autoria própria.

A interação dos usuários será por meio do navegador *web*. Dessa maneira, qualquer dispositivo que possua navegador poderá gerenciar o sistema e interagir com as aplicações. Até então, o sistema era controlado por apenas um operador, que tinha uma estação de controle fixa. A interação do usuário na plateia acontecia através de controles nas poltronas, que ficaram em desuso, pois houve poucas aplicações desenvolvidas que fizessem uso dos mesmos. A nova interface das aplicações poderá ser feita em *HTML* e *JavaScript*, possibilitando assim, facilidade de se desenvolver designs compreensivos e responsivos. Isso tudo é possível, pois o sistema proposto funciona por meio de uma *API Rest*, ou seja, não tem uma interface gráfica propriamente dita. É o núcleo para tal, mas é independente de interface gráfica. Isso possibilita a liberdade de se desenvolver outros clientes (interfaces) baseado em qualquer outra tecnologia e contanto que se comunique com o padrão da API definida, o sistema irá responder em conformidade.

Dessa forma, o operador, considerando a nova arquitetura, poderia controlar o ambiente, tanto com um computador convencional, um *notebook* ou um *smartphone*. Os usuários na plateia também poderiam interagir com a aplicação por meio de seus *smartphones*. Essa *API* também possui "pontes" que permitem a troca de pacotes entre o dispositivo do usuário e a aplicação (derivada) escrita em *script* (*AngelScript*), possibilitando assim, o livre desenvolvimento de aplicações interativas. Ao fim, temos um sistema onde o processo

de renderização da aplicação é apenas um, ou seja, não é distribuído. A aplicação proporciona liberdade ao usuário para interagir com a mesma a partir de qualquer equipamento móvel, em tempo de execução.

De modo a modernizar o ambiente, que conta com equipamentos que têm mais de uma década de uso, estamos propondo a seguinte mudança de equipamentos com tecnologias que são mais adequadas para 2020: substituição dos projetores *Barco F32*, pelos novos *Barco F90-4K13*[1] que são projetores da mesma qualidade e confiabilidade que os projetores atuais, possuem anel baioneta compatível para as lentes grande angular do ambiente fulldome, além de terem resolução 4K e 11800 lúmens cada projetor. Estes novos projetores são baseados em tecnologia de projeção de fósforo a laser, que garante vida útil da lâmpada a até 40000 horas, em contrapartida das 2000 horas dos projetores *LED* atuais. Esse tipo de iluminação apresenta melhor saturação das cores, assim como tonalidades mais fiéis, com menos aberração cromática. Diferentemente do *Barco F32*, o *Barco F90-4K13* possui 3D ativo, ou seja, a tecnologia para estereoscopia é por meio de multiplexação de canais por tempo e não por frequência de cor. Dessa maneira, tem-se contraste de cores fiéis, ao contrário da estereoscopia por método passiva, que divide as frequências de cores para cada canal, gerando aberração cromática na saturação das cores da imagem final projetada. O projetor também tem cotação de preço similar ao equipamento corrente.

Também é sugerido o uso de uma máquina mais moderna, de modo a simplificar a infraestrutura, trocando as máquinas antigas por apenas uma que pode até exceder a capacidade computacional da configuração atual. A infraestrutura original conta com 6 computadores, os quais possuem a mesma configuração de *hardware*, placa de vídeo *NVIDIA Quadro 4000*[2], dois processadores Xeon da arquitetura *Products formerly Gladden*[3](2012) e 12 *gigabytes* de memória *RAM* (máximo suportado pelo processador é de 32 *gigabytes*). O equipamento que poderia substituir todas essas máquinas e com melhor desempenho computacional, poderia ser uma máquina com um processador *Intel i9 9900K*[4] que suporta indexar até 128 *gigabytes* de *RAM*, com uma placa de vídeo *Nvidia Geforce RTX 2080 Ti*[5] com 11 *gigabytes* de memória, têm poder de processamento de 25 *TeraFLOPS* e possui 5 saídas de vídeo.

[1] <https://www.barco.com/en/product/f90-4k13> Acessado em 14/01/2020.
[2] <https://www.nvidia.com.br/object/product-quadro-4000-br.html> Acessado em 14/01/2020.
[3] <https://ark.intel.com/content/www/us/en/ark/products/codename/37504/gladden.html> Acessado em 14/01/2020.
[4] <https://www.intel.com/content/www/us/en/products/processors/core/i9-processors/i9-9900k.html> Acessado em 14/01/2020.
[5] <https://www.nvidia.com/en-eu/geforce/graphics-cards/rtx-2080-ti/>, <https://www.evga.com/products/product.aspx?pn=11G-P4-2281-KR> Acessado em 14/01/2020.

4.1.2 Construção do *middleware*

Nesta seção iremos responder como resolver o segundo objetivo específico. Como visto na subseção 3.1.2, a nível de *software*, o sistema é composto por múltiplas tecnologias. Aqui será descrito como estes componentes se interligam.

São dois os principais módulos do sistema: a instância gráfica e o servidor *backend*. A Figura 7 mostra uma ilustração simplificada disto. A instância gráfica é o modulo que desenha as imagens (renderiza) para serem projetadas. Ela é implementada em C++ e usa a *Urho3D* como alicerce. Esse módulo normalmente recebe o parâmetro de *path* para um *script* (*AngelScript*), que representa uma aplicação que será executada. O *script* é interpretado, possui implementação de lógica e também descreve a cena da aplicação em questão. O fato da aplicação final poder ser escrita em *script* é relevante pois separa o código do sistema e o da aplicação, dando a possibilidade de se desenvolver várias aplicações, de várias temáticas, cada uma implementada em um *script* separado. Além disso, modificações específicas a nível de *script* podem ser alteradas com um editor de texto simples e serem imediatamente executadas, sem necessidade de recompilar todo o sistema ou de ter um ambiente de desenvolvimento para tal.

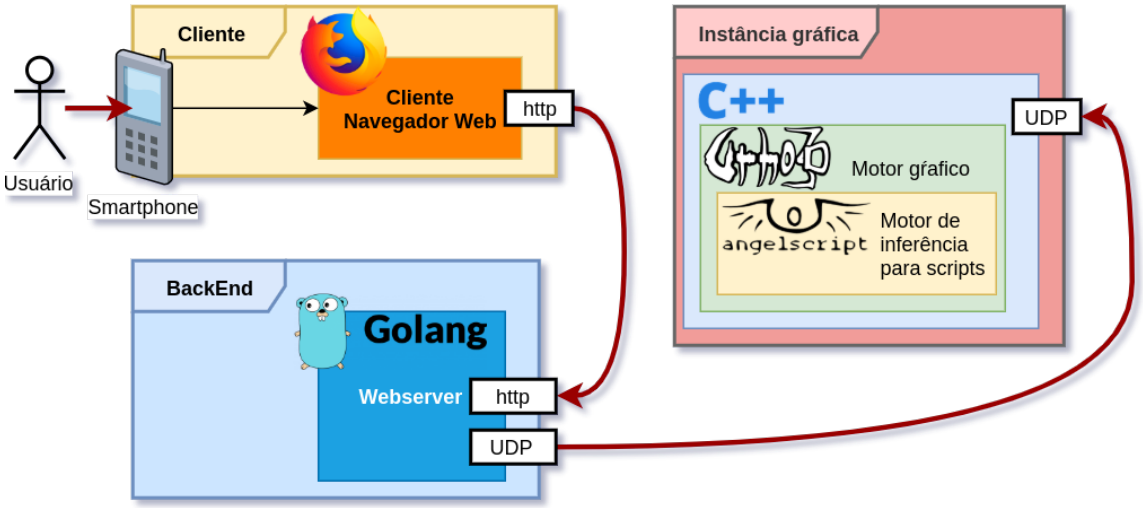


Figura 7 – Representação gráfica dos componentes tecnológicos usados no sistema.

Fonte da ilustração: Autoria própria.

A instância gráfica comunica com o servidor *backend* escrito em *GoLang* por meio de pacotes *UDP*. Como esses dois módulos/processos executam na mesma máquina/computador, não existe risco de pacotes serem perdidos, mesmo sendo transportado pelo protocolo *UDP*. Haveria risco, no entanto, se o pacote tivesse que ser transportado em rede, por mais de uma máquina. A camada de *software* que integra o *backend* com a instância gráfica também é escrita em *GoLang*, porém, transpilado para C++ e compilado com o código do módulo gráfico estaticamente.

O módulo *backend* em *Go* é a parte que orquestra o sistema. Ele é responsável por subir o processo gráfico (com o parâmetro de *path* para um *script*), sob demanda do usuário/controlador. A instância gráfica então, abre um canal de comunicação com o *backend*, para receber demais comandos vindos da rede. Eventualmente, uns destes comandos são direcionados para serem tratados na lógica escrita em *scripts*, dando a possibilidade de se desenvolver aplicações responsivas, com comandos vindos diretamente da rede, ou seja, dos aparelhos móveis dos usuários. A Figura 8 mostra um diagrama de sequência exemplificando uma chamada implementada em *AngelScript* sendo feita a partir de um comando vindo do usuário, usando um navegador *web* como cliente. Adicionalmente, a Figura 9 mostra um diagrama de sequência onde ilustra todas as camadas do sistema proposto e qual o fluxo de chamadas possíveis a serem tratadas pelo sistema. Observe que as invocações dos métodos dos objetos, podem ocorrer em três instantes: no *Backend*, na instância gráfica e no motor de inferência *AngelScript*.

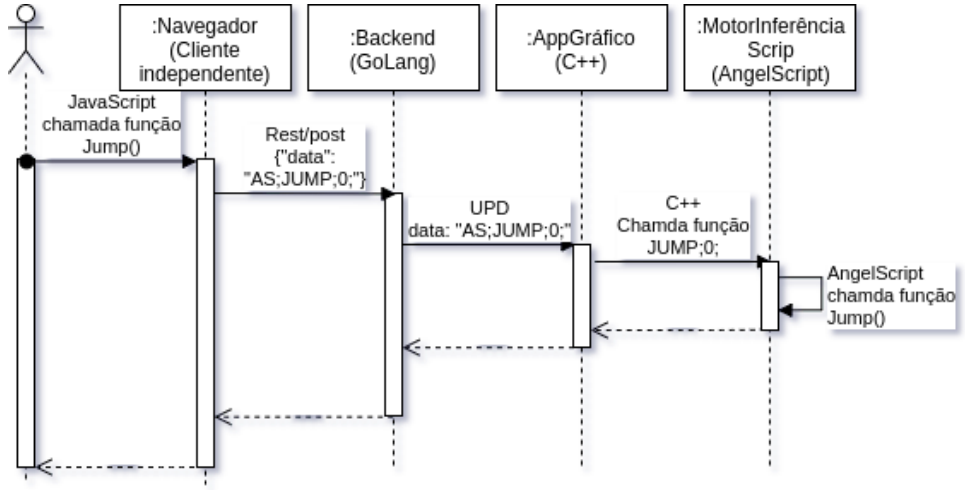


Figura 8 – Diagrama de sequência exemplificando uma chamada de função em *AngelScrip*, originada pelo usuário, via cliente navegador *web*.

Fonte: Autoria própria.

Além destes dois módulos, existe outro processo disparado pelo *backend* para servir páginas *web*. Estas são utilizadas como interface do usuário com o sistema. Dessa forma, o usuário pode abrir painéis de controle com um navegador de *internet*. O fato é que, o módulo *backend* disponibiliza uma pequena *API REST* que permite o controle do sistema como um todo. Nada impede alguém de implementar um cliente para se comunicar com tal *API*, seja por meio de um aplicativo nativo *Android iOS* ou pelo navegador, através da interface *web* similar ao já disponibilizado pelo *middleware* proposto.

4.1.3 Procedimento para gerar imagens *fulldome*

Esta seção responderá como o terceiro objetivo específico é alcançado. Será descrito como é feita a geração e a correção geométrica da imagem final *fulldome* que é projetada

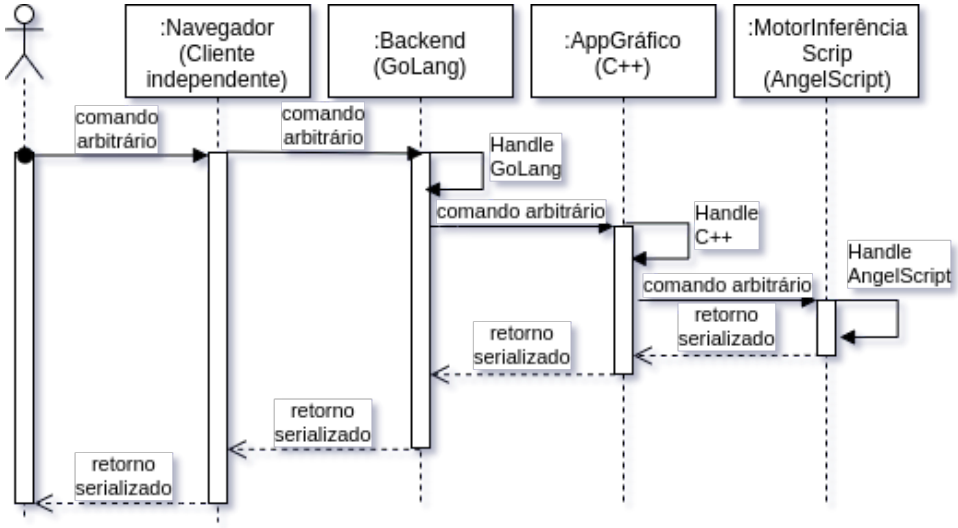


Figura 9 – Diagrama de sequência demonstrando fluxo possível entre todas as camadas do sistema.

Fonte: Autoria própria.

diretamente na cúpula do teatro *fulldome*.

Existe em essência, duas abordagens para se ter a imagem adequada para projeção direta no ecrã *fulldome*. Uma abordagem é usar modelagem matemática aliada a programação com *hardware* gráfico como, por exemplo, programação em *shaders*[6]. O outro método é usar os recursos básicos que se encontram na maioria das bibliotecas gráficas, como por exemplo, as propriedades de mapeamento de textura dos modelos. O segundo método é bem similar ao que Bourke utilizou em uma de suas abordagens com renderização em tempo real usando a *Unity* (BOURKE, 2015). De modo a simplificar a implementação, optou-se pelo segundo método.

Então, o processo para geração da imagem a ser projetada no teatro é definido na sequência. Supondo que temos uma cena virtual em um espaço cartesiano de três dimensões com eixos X, Y e Z; considerando também, um ponto de referência O em um espaço qualquer desta cena, que represente um observador, realizamos a captura de 5 imagens, de modo que, cada imagem seja obtida em direções simultaneamente perpendiculares partindo do ponto O.

A captura de imagem em software gráfico é feita por uma entidade que chamamos simplesmente de câmera. A câmera é um dos componentes primitivos para *engines* gráficas, assim como luzes e modelos tridimensionais, cada uma tem seus parâmetros específicos. Precisamos focar em apenas dois parâmetros da câmera que são o ângulo de abertura da captura e a normal da câmera. A Figura 10 mostra uma representação geral do que é uma câmera na maioria dos softwares gráficos.

[6] <https://www.khronos.org/opengl/wiki/Shader> visto em 08/03/2020

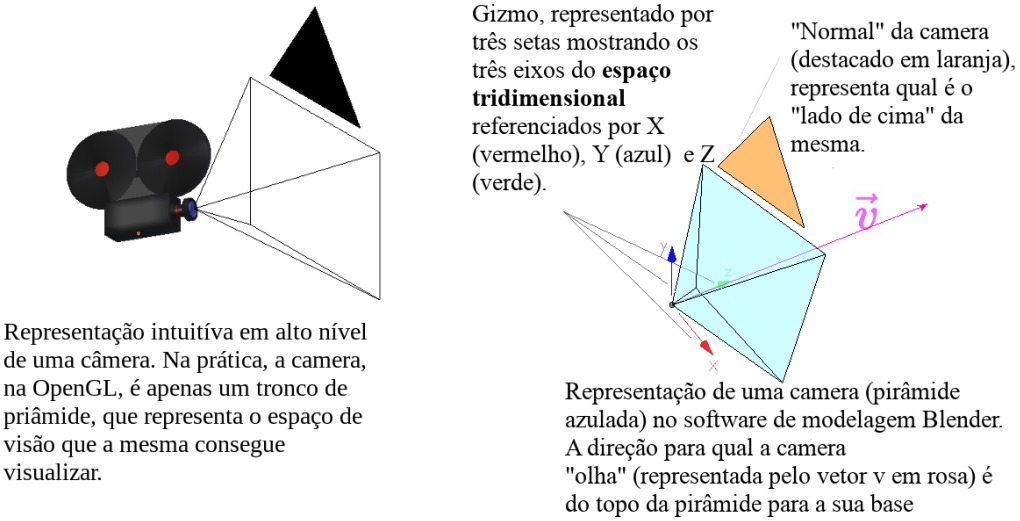


Figura 10 – Representação simplificada do componente câmera em softwares de computação gráfica.

Fonte: Autoria própria.

Auxílio para estudo: <https://relativity.net.au/gaming/java/Frustum.html>

A normal da câmera indica para qual lado é a parte de cima da câmera. Se a normal estiver de ponta cabeça em uma cena, a imagem que a câmera "vê" estará de ponta cabeça também. A Figura 11 ilustra uma cena, onde temos um objeto qualquer (nave espacial) e a câmera em diversas orientações. A normal da mesma é ilustrada por um triângulo adjacente a um dos lados da base da pirâmide, que representa a câmera em si. A representação da câmera como uma pirâmide nas imagens é meramente ilustrativa, devido a maneira como o *Blender* (*software* de modelagem) desenha a mesma. A câmera na realidade é um espaço representado por um tronco de pirâmide.

O ângulo de captura da câmera define o campo de visão (no inglês *FOV* ou *Field of View*). A Figura 12 ilustra uma cena de teste em dois instantes, uma com a câmera com um ângulo de visão de 60º e outra com o ângulo de visão de 90º (ângulo reto).

O observador que citamos anteriormente, representado por O, na realidade é um conjunto de cinco câmeras, cada uma encarregada de capturar uma imagem de cada lado em torno dos trezentos e sessenta graus de latitude do ponto O e uma apontada para cima, cobrindo os noventa graus de longitude. Esse tipo de configuração é usualmente feito para compor imagens do tipo da *fish eye* de maneira virtual. A Figura 13 demonstra a montagem desse composto de cinco câmeras para composição de uma imagem *fish eye*.

A Figura 14 mostra um exemplo do uso desse composto de câmeras para capturar as 5 imagens em uma cena qualquer. Essas imagens foram feitas obedecendo as seguintes regras: cada imagem foi capturada em direções perpendiculares em um dado ponto em comum do espaço da cena; cada imagem tem aspecto quadrado e ângulo de visão de

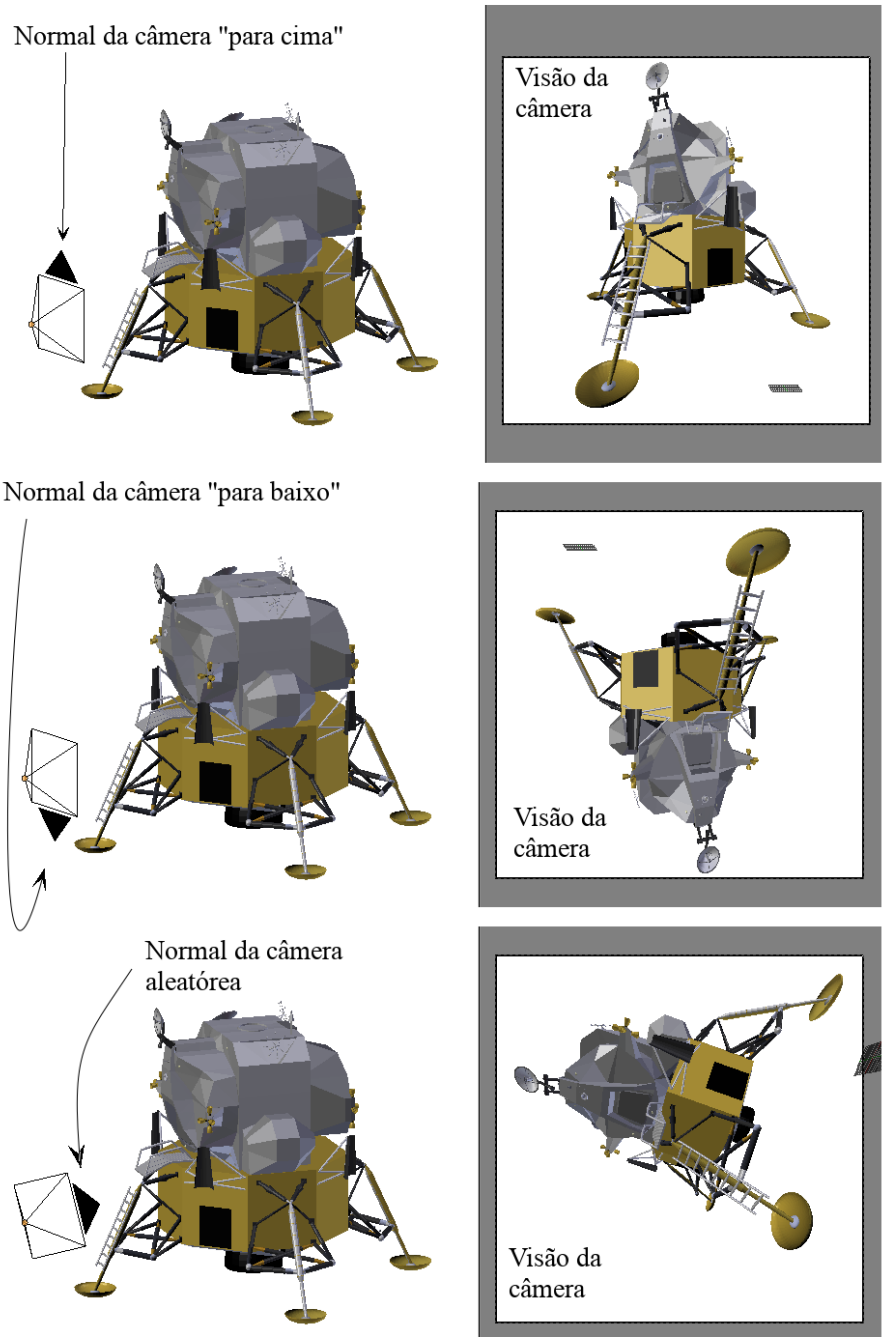


Figura 11 – Demonstração conceitual da normal de um componente câmera.

Fonte da ilustração: Autoria própria. Modelo: NASA *https://nasa3d.arc.nasa.gov/models*

exatamente noventa graus.

Em seguida, as imagens obtidas devem ser aplicadas em um modelo tridimensional especial do domo representado por uma semiesfera oca. Esse modelo tridimensional do domo de semiesfera deve ter uma malha geométrica pentaédrica. Em outras palavras, o modelo é uma semiesfera, na qual, a construção da malha geométrica lembra os lados do cubo, no entanto, com sua superfície em forma de esfera. Esse modelo de semiesfera irá representar o nosso “domo virtual” que será usado posteriormente no processo de correção

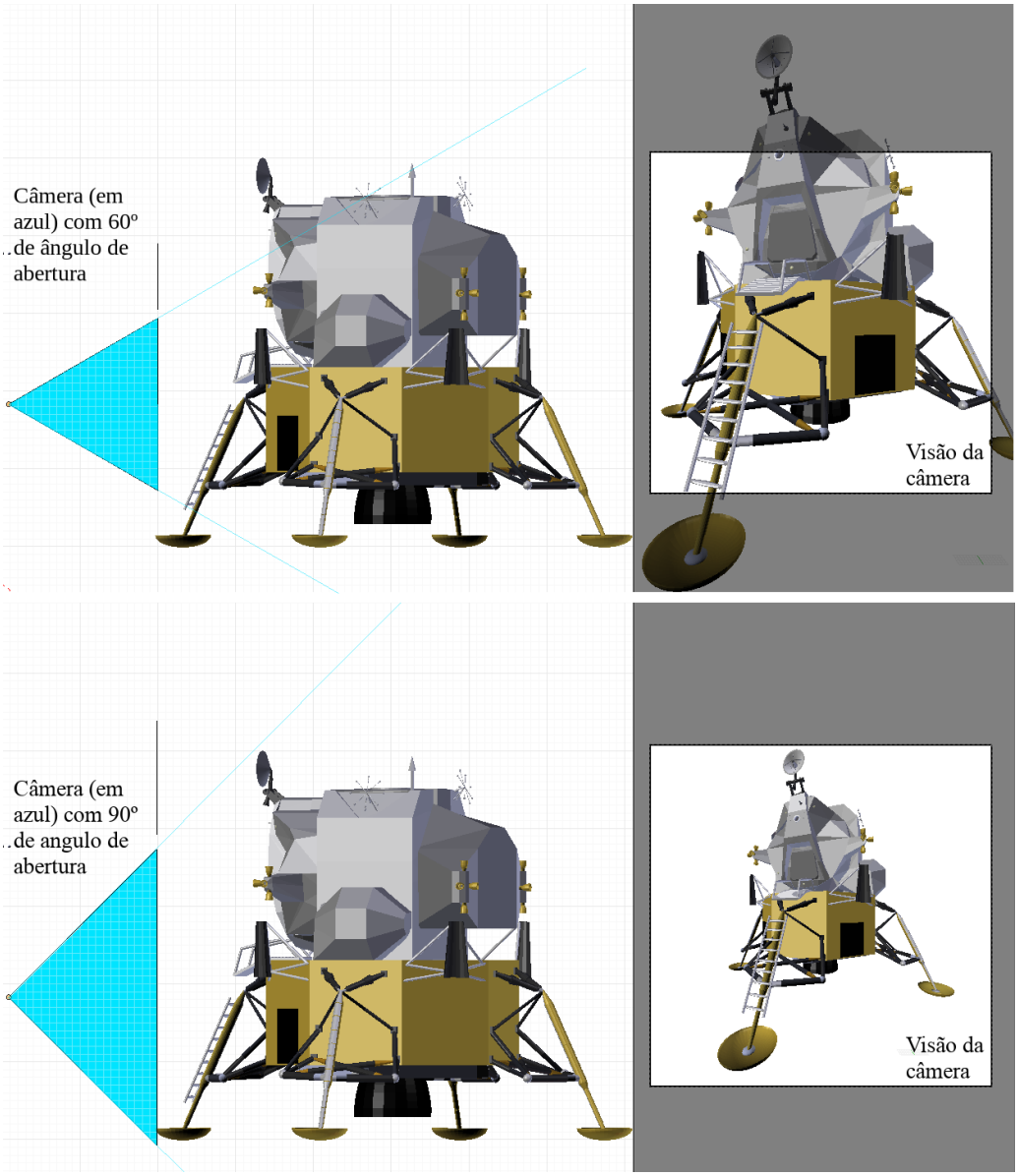


Figura 12 – Demonstração intuitiva do funcionamento do *FOV* de um componente câmera. Fonte da ilustração: Autoria própria. Modelo: NASA *https://nasa3d.arc.nasa.gov/models*

morfológica da imagem a ser projetada na cúpula física real.

A Figura 15 ilustra o que é uma semiesfera de malha geométrica pentaédrica: Ao lado esquerdo, para comparar, uma semiesfera UV tradicional que é usada por padrão nos softwares de modelagem (tal como o *Blender*) e a direita, a semiesfera com malha geométrica pentaédrica.

Em softwares gráficos, modelos tridimensionais podem ter texturas as quais possuem um mapeamento de uma imagem 2D para a superfície tridimensional do modelo. Por convenção, os softwares gráficos referem-se a este mapeamento com o nome de mapeamento UV. A nomenclatura UV é usada para distinguir as referências de eixos do espaço tridimensional x, y e z para o do mapeamento bidimensional das texturas. Ademais, temos

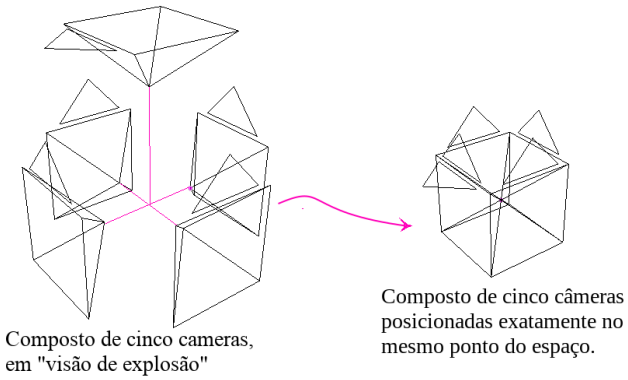


Figura 13 – Ilustração do composto de câmeras para posteriormente formar imagens *fish eye*.

Fonte: Autoria própria.

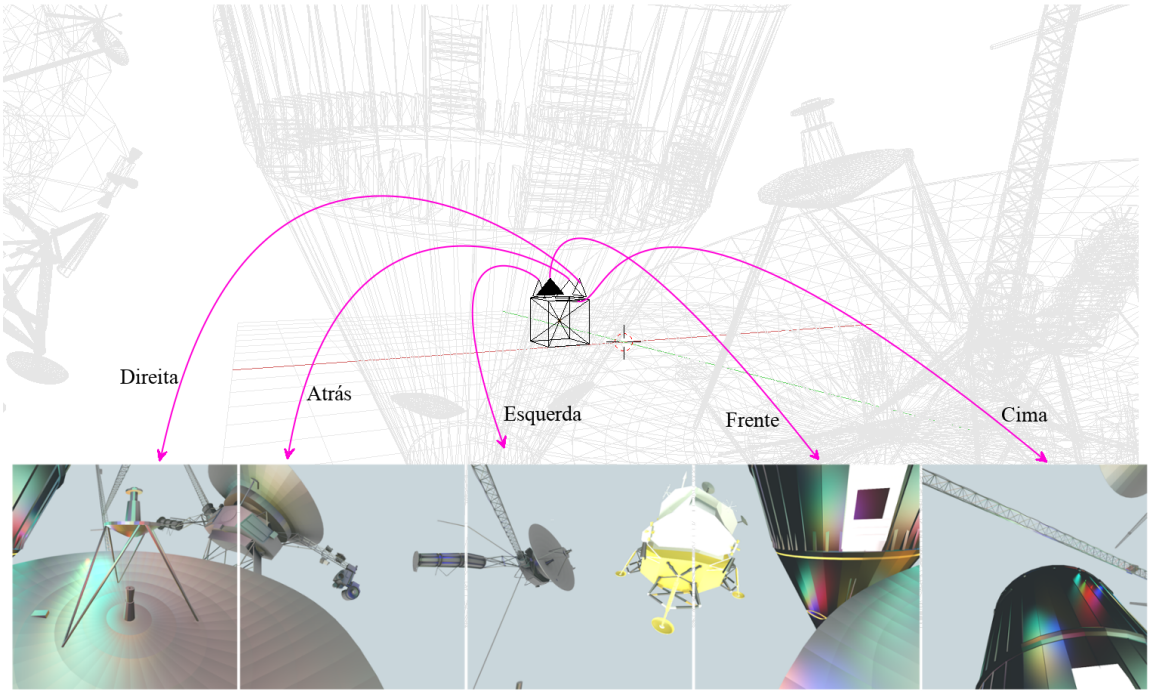


Figura 14 – Demonstração do composto de câmeras capturando imagens em uma cena.

Fonte: Autoria própria.

o conceito de material, que são parâmetros que podem definir características como reflexão, brilho, cor da superfície do modelo a um dado segmento da malha geométrica do mesmo. Normalmente, *engines* gráficas permitem associação de mais de um material para cada modelo geométrico. Para finalidade da nossa aplicação, usamos um material em cada face da malha pentaédrica da semiesfera.

s

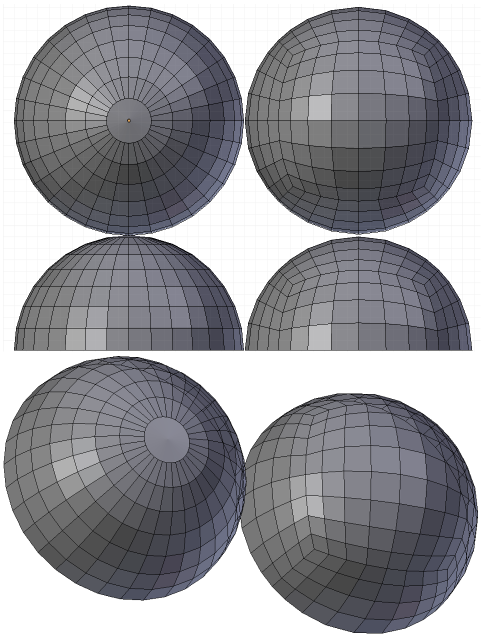


Figura 15 – Visão ortogonal dos modelos para semiesferas, a esquerda, com malha poligonal em disposição UV (polar). A direita, uma malha poligonal, formada por cinco regiões retangulares.

Fonte da ilustração: Autoria própria.

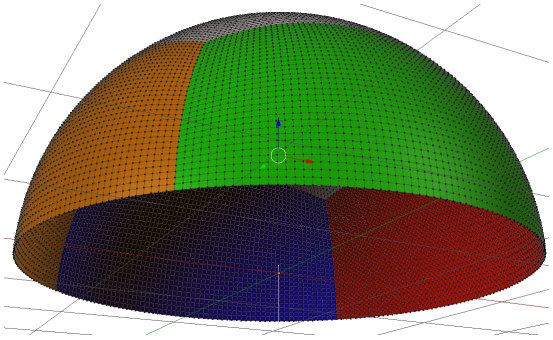


Figura 16 – Visão em perspectiva do modelo, formada por cinco regiões retangulares, na qual, cada uma das regiões possui um material diferente separadas por cor.

Fonte: Autoria própria.

Na Figura 16 é possível observar o modelo da semiesfera com malha geométrica pentaédrica, em que para cada uma das cinco regiões da malha temos um material diferente, representado com uma cor distinta. Nossa intenção em separar as regiões da malha é porque posteriormente iremos aplicar as imagens que coletamos anteriormente com o composto de câmeras em cada um dos lados/regiões da malha pentaédrica.

A Figura 17 ilustra o conjunto de cinco imagens obtidas do composto das cinco câmeras sendo aplicadas no modelo do domo virtual de malha geométrica pentaédrica. Cada imagem situa-se em uma região distinta da malha (definido pelos materiais), formando uma plataforma de visualização para imagens em trezentos e sessenta graus. Essa plataforma

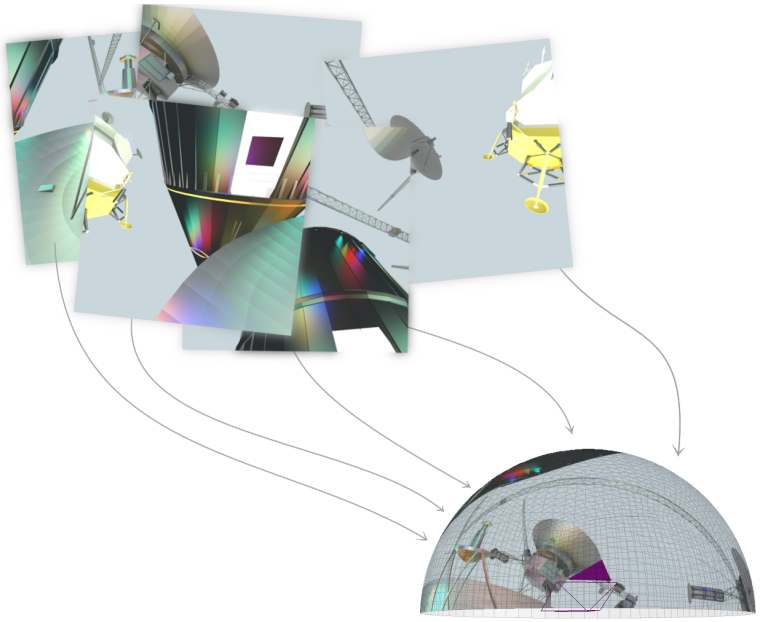


Figura 17 – Ilustração conceitual da aplicação das imagens da cena virtual sobre um modelo de uma semiesfera de malha geométrica pentaédrica.

Fonte: Autoria própria.

pode ser usada para fazer *skyboxes*[7] ou fundos de cenário virtuais, que são usualmente vistos em efeitos especiais para filmes ou videogames. Em nosso caso, estamos usando essa plataforma para gerar a imagem *fish eye*.

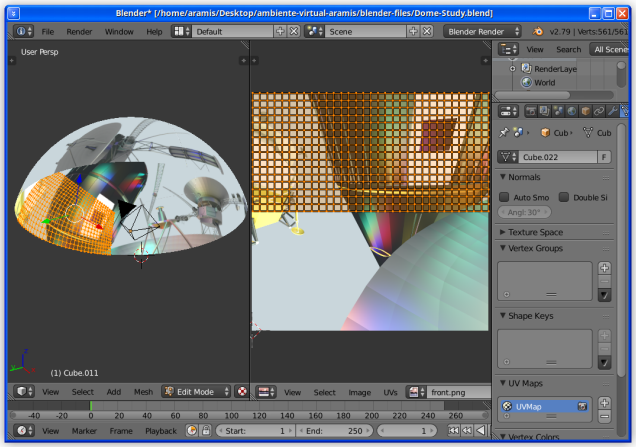


Figura 18 – Captura de tela do suíte de modelagem *Blender*, onde o usuário edita o mapeamento de textura *UV* das regiões da malha pentaédrica.

Fonte: Autoria própria.

A Figura 18 mostra o processo de edição de uma das regiões/lados da malha pentaédrica da semiesfera. Na imagem, o mapeamento UV de um dos cinco lados da malha pentaédrica está sendo sobreposta em cima de uma das cinco imagens obtidas no processo

[7] <https://www.khronos.org/opengl/wiki/Skybox> visto em 03/03/2020

anterior representado na Figura 14. Internamente, nossa ferramenta realiza esse processo sem que o usuário perceba. As Figuras 14, 17 e 18 foram usadas apenas para ilustrar um processo que não é visto em nenhum momento pelo usuário durante o funcionamento do *framework*.

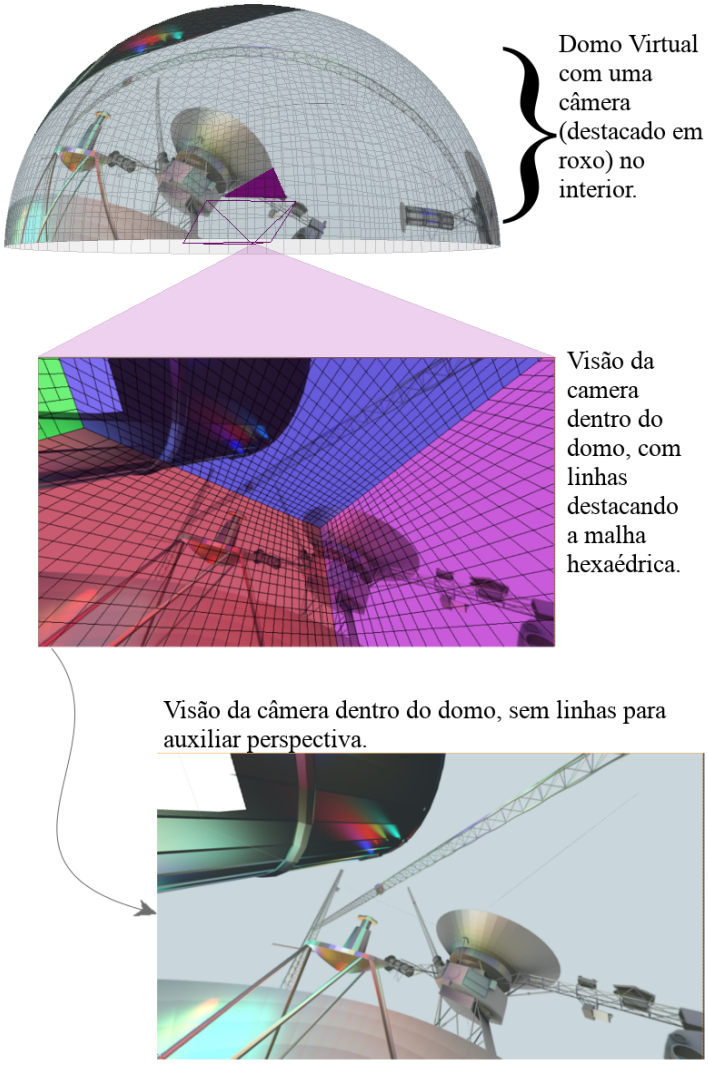


Figura 19 – Demonstração do comportamento das imagens na malha pentaédrica no domo virtual, tomando como ponto de vista, o centro do domo virtual, situado na base do mesmo.

Fonte: Autoria própria

Observando Figura 19 é possível notar que se posicionarmos uma câmera no exato centro da esfera a qual a semiesfera virtual é feita, é possível simular a visão em 360º sem que ocorra distorções da imagem. O resultado da junção das cinco imagens, é a visão em 360º da cena a qual o conjunto de imagens foi originalmente feito. Se houver deslocamento da perspectiva de vista para qualquer outro ponto, referente ao domo virtual, iremos gerar uma distorção da imagem. Isso será usado posteriormente para fazer a correção morfológica da imagem.

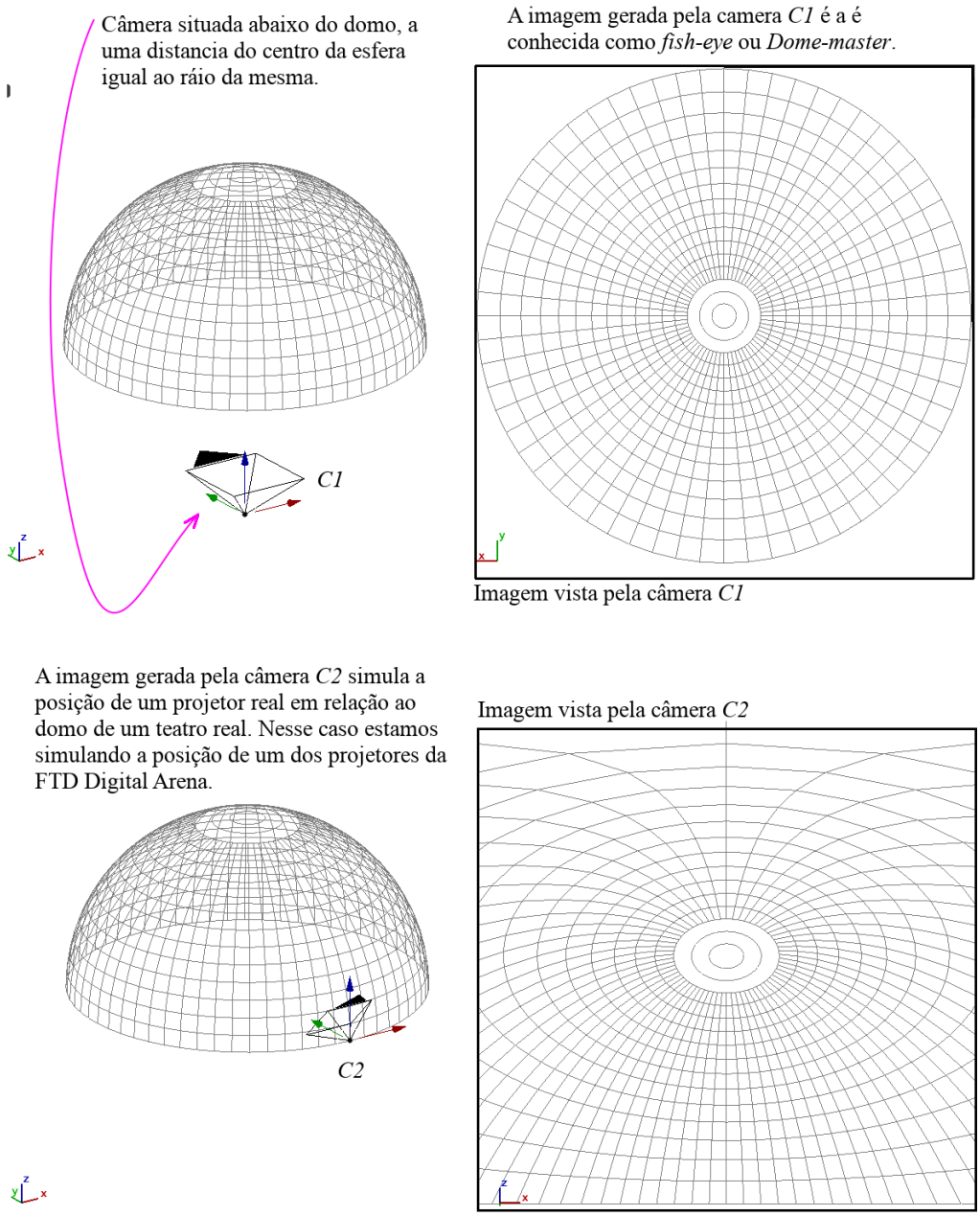


Figura 20 – Demonstração da correção morfológica da imagem para projetores em um teatro fulldome e geração da imagem *dome-master*.

Fonte: Autoria própria.

Cada projetor físico no teatro é simulado como sendo uma câmera que observa uma cúpula virtual. Essa cúpula virtual é a semiesfera pentaédrica com as imagens aplicadas. A posição da câmera virtual (simulando o projetor real) e do domo virtual são coincidentes com seus pares reais. A imagem gerada por essa câmera é a própria correção morfológica da imagem para o projetor físico no teatro. A imagem gerada irá aparentar ser distorcida, mas ao ser projetada, a imagem irá "encaixar" no ecrã.

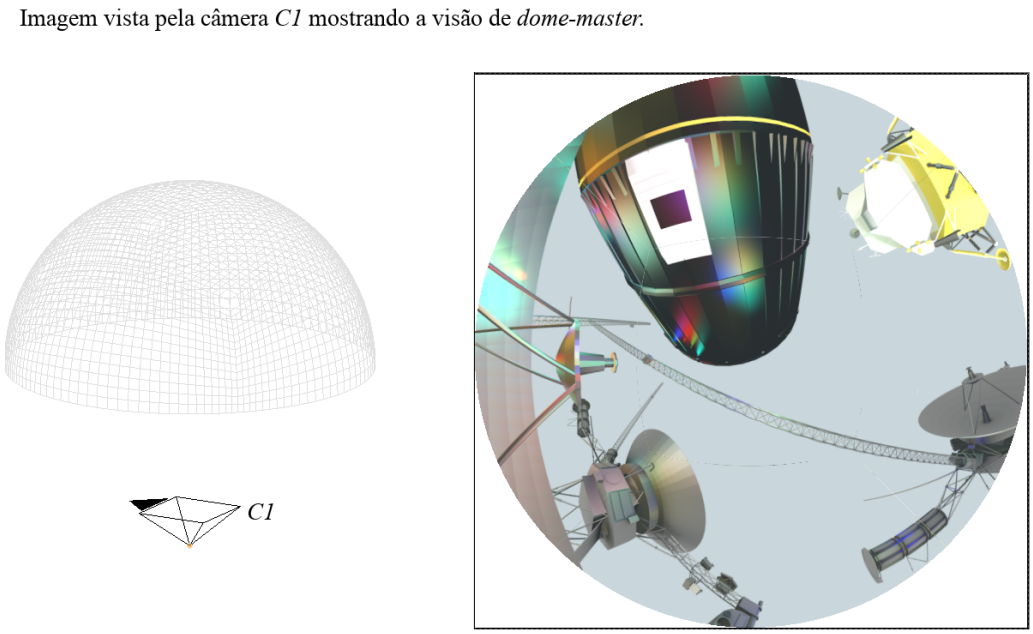


Figura 21 – Demonstração da correção morfológica com imagens geradas para projetores em um teatro fulldome e geração da imagem *dome-master*.

Fonte: Autoria própria.

A Figura 20 mostra essa distorção gerada a partir da simulação da posição de um dos projetores físicos para uma determinada configuração de teatro *fulldome*, nesse caso, similar ao da Arena Digital. A Figura 20 também mostra como, a partir da mesma abordagem, geramos a imagem "*Domemaster*" que também é conhecida como imagem "*fish eye*", que por convenção é o padrão adotado para softwares de edição de vídeo em plataforma *fulldome*.

Por fim, a Figura 21 mostra a correção geométrica para as cinco imagens que capturamos com o composto de câmeras na nossa cena de teste.

4.1.4 Calibragem de projeções *fulldome*

Esta seção irá definir como é resolvido o quarto objetivo específico, sobre calibrar projeções em ecrãs *fulldome*.

Após geradas as imagens do processo descrito na subseção 4.1.3, a simples projeção dessas imagens na superfície do domo não bastará. As lentes dos projetores e até mesmo a própria superfície de projeção possuem deformidades as quais impossibilitam realizar uma costura perfeita sem que haja um processo adicional para correção. Nessa circunstância é necessário um meio para juntar as projeções de maneira localizada. No sistema proposto, dispomos de duas ferramentas para isso: um calibrador manual e outro semiautomático. O calibrador manual é a ferramenta que permite realizar as seguintes ações: Dada a cena virtual, descrita na subseção 4.1.3, a câmera virtual, a qual observa o domo virtual, pode ter todos os seus parâmetros alterados, seja a rotação nos 3 eixos, o translado nos 3 eixos assim como o ângulo da abertura da lente. Dessa maneira é possível alterar a perspectiva das imagens que são projetadas na cúpula real, de maneira a deixá-las bem próximas de terem uma junção contínua. Adicionalmente ao processo de *renderização* dessas imagens, é feita outra passagem de *renderização* e a imagem gerada é renderizada em uma geometria poligonal plana de 9x9 vértices (9 vértices de largura por 9 de altura). Essa geometria poligonal pode ter estes vértices ajustados/deslocados de maneira contida num plano (eixos x e y). Dessa maneira é possível ajustar pontos específicos da imagem como mostra a figura 22. Com essas ferramentas, que compõem o calibrador manual, é possível realizar a junção de múltiplas imagens com perspectiva de projeção distintas.

Apesar de o calibrador manual ser capaz de realizar a junção das imagens, a tarefa é demorada e difícil. Normalmente alguns erros de perspectiva persistem mesmo após as imagens terem juntas contínuas perfeitas. Para tornar a tarefa mais simples, rápida e com perspectiva de projeção mais próxima do adequado, criou-se um método semiautomático. Ele não é completamente automático, pois exige ajustes finos após o término da execução do mesmo, mas parte substancial do problema é resolvido de forma automática. A Figura 23 mostra o processo simplificado utilizado para aquisição dos pontos usados para mapear a perspectiva de cada projetor.

O processo descrito na Figura 23, funciona da seguinte maneira: uma câmera fotográfica, situada em uma sala, registra fotos de marcadores feitas por um projetor ou *display*. São duas etapas. Uma foto sem o marcador e outra com o marcador. Em seguida, o processo calcula a variação mais significativa de cada pixel entre as duas fotos tiradas; se a diferença de cor do pixel for grande, destaca-se o dito pixel, ao contrário, deixa-se o mesmo em branco. O limite que define se a variação é grande ou não, fica a critério

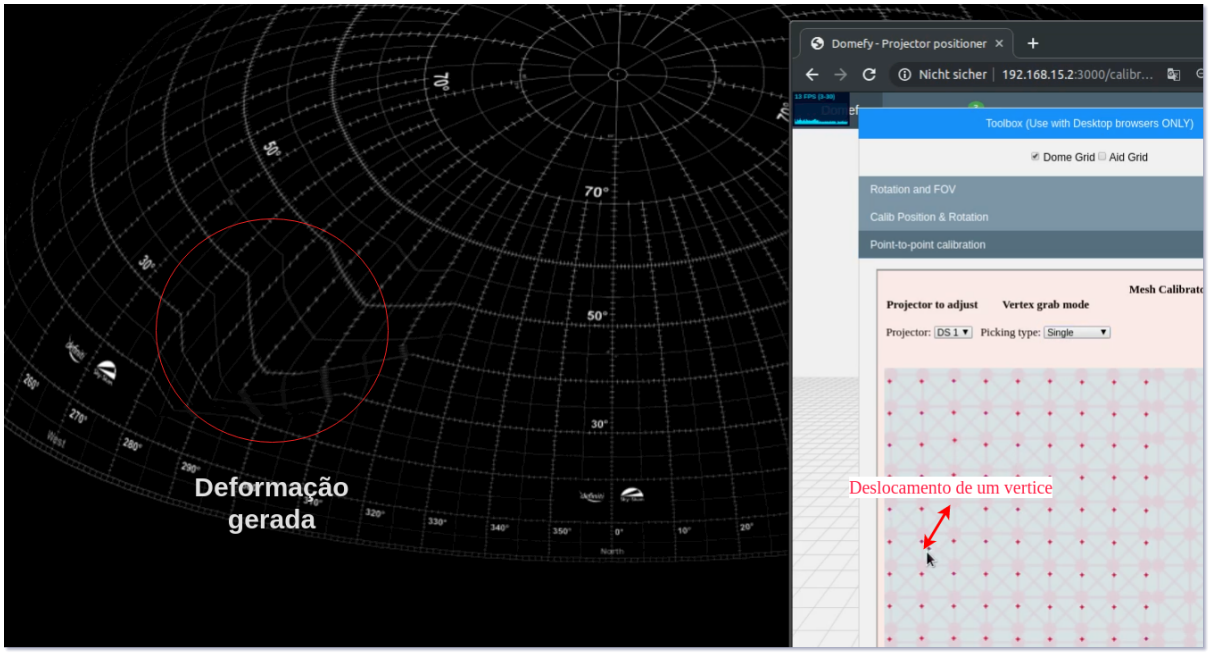


Figura 22 – Demonstração da calibração ponto a ponto. Vídeo: <https://youtu.be/KAlLalKcC7o>

Fonte: Autoria própria.

do usuário, visto que é subjetivo determinar um valor, considerando tantos ruídos que podem existir. Em circunstâncias ideais, o limite pode ser baixo. Com equipamentos menos precisos, o limite deve ser alto. Com essa região destacada, podemos somar a posição dos *pixels* e fazer uma média simples de sua posição em um plano cartesiano, obtendo assim, um valor final x e y. Esse processo é repetido para todos os marcadores. O número de marcadores, por sua vez, pode ser definido no início do processo. Quanto maior, mais suave será o resultado da correção de perspectiva. Para superfícies complexas é ideal um número grande de marcadores. Para superfícies de um plano apenas, um baixo número satisfará. A figura 24 mostra como se parece esse cenário de aquisição de perspectiva. A Figura 25 mostra um exemplo da detecção de marcadores

Assim que o processo de aquisição de pontos termina, podemos gerar uma geometria com mapeamento de textura usando essa informação. Em computação gráfica, temos a habilidade de mapear textura (imagens, fotografias etc.) sobre a superfície de uma geometria poligonal, comumente chamado de *mapeamento UV*[8]. Similar à cartografia, da mesma forma que se pode desenhar os continentes do globo em um mapa plano, na computação gráfica pode-se aplicar uma imagem em uma geometria poligonal. Dito isso, o que será feito na próxima etapa será mapear os pontos coletados (na aquisição da perspectiva de projeção) sobre uma superfície de uma geometria poligonal plana.

Um programa foi escrito para gerar essa geometria plana, recebendo como pa-

[8] <https://docs.blender.org/manual/en/latest/editors/uv/introduction.html> visto em 14/02/2020

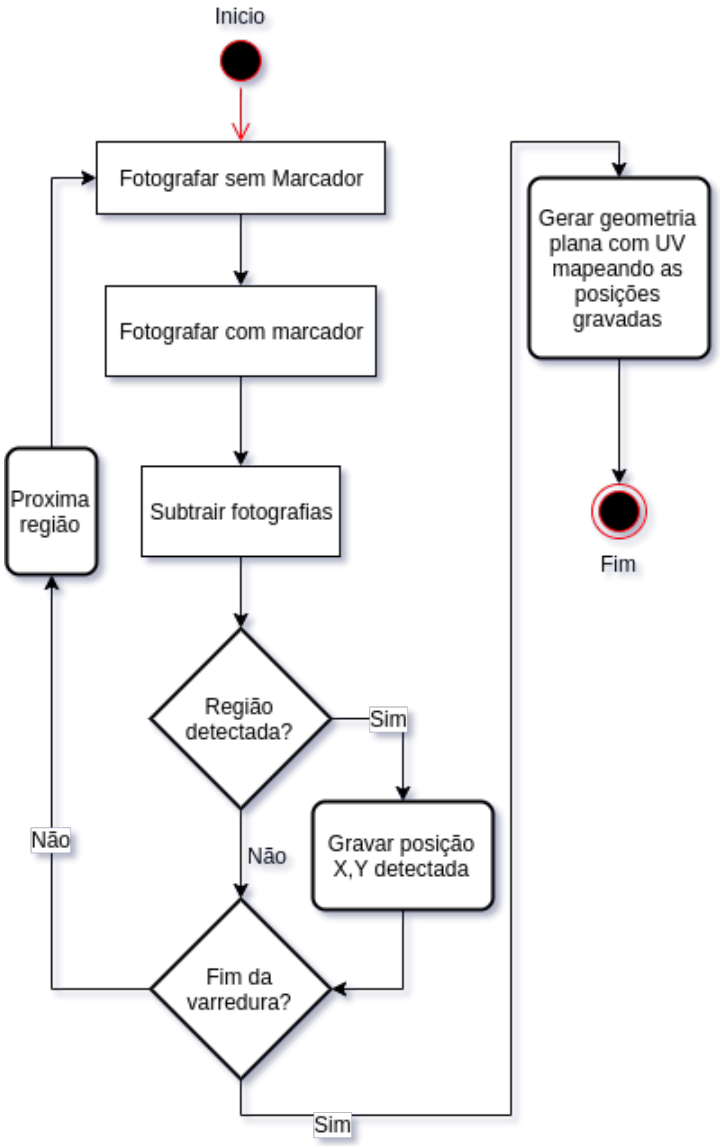


Figura 23 – Algoritmo simplificado do processo para aquisição dos pontos de mapeamento da perspectiva de projeção.

Fonte da ilustração: Autoria própria.

râmetros, a lista de pontos adquiridos no processo anterior e o número de marcadores utilizados. O programa então gerará um arquivo de objeto 3d .obj (*Wavefront*) com o respectivo mapeamento UV referente aos pontos coletados. A figura 26 mostra um exemplo de um mapeamento UV gerado a partir dos pontos coletados no processo de aquisição de perspectiva de projeção. A figura 27 mostra uma imagem mapeada sobre essa malha de pontos gerados. Note que a perspectiva que predomina é o da câmera e não a do projetor.

Tomando a figura 28 como base, o processo executado então, fará a transposição da perspectiva de projeção do projetor b para a perspectiva da câmera a. Desta forma, seria como se estivéssemos projetando a partir da câmera. O processo efetivamente converte a perspectiva do projetor/monitor para o da câmera a qual inicialmente foi usada para

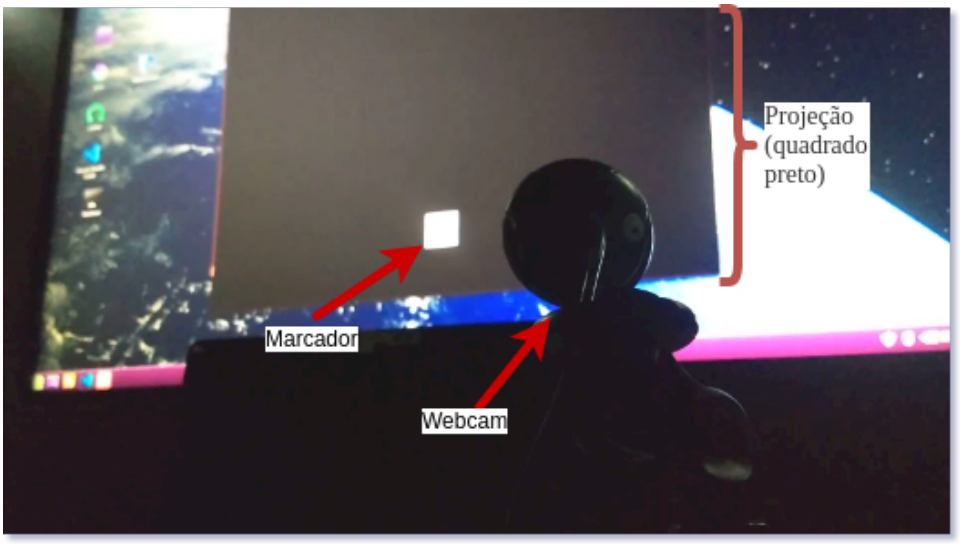


Figura 24 – Demonstração simples do processo de aquisição da perspectiva de projeção.

Fonte: Autoria própria.

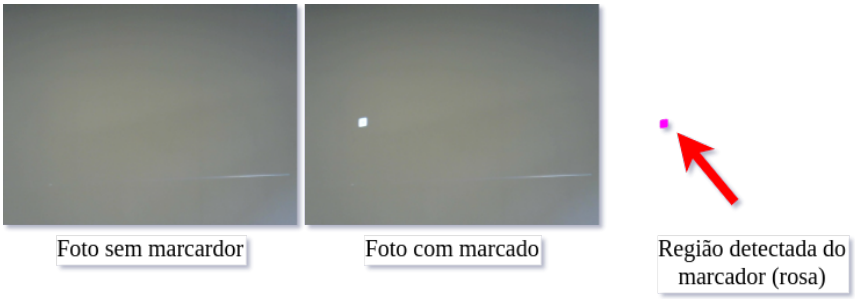


Figura 25 – Resultado obtido com o processo de detecção de um marcador.

Fonte: Autoria própria.

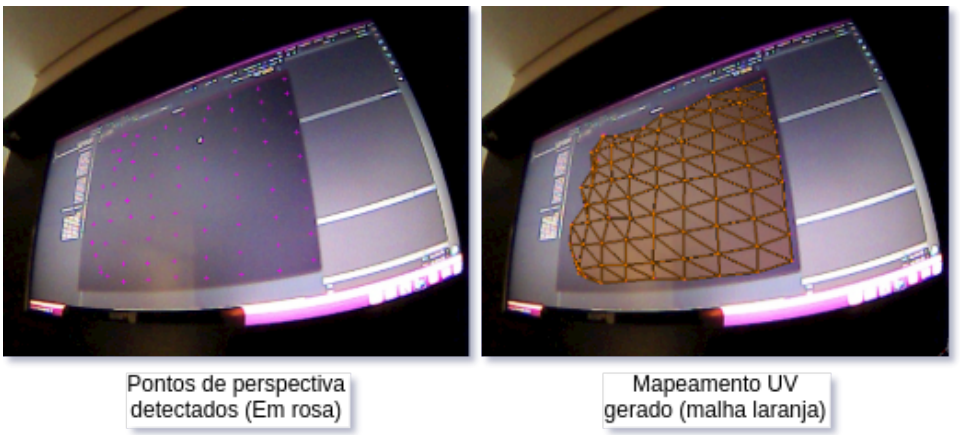


Figura 26 – Demonstração de um mapeamento UV gerado a partir dos pontos de perspectiva adquiridos.

Fonte da ilustração: Autoria própria.

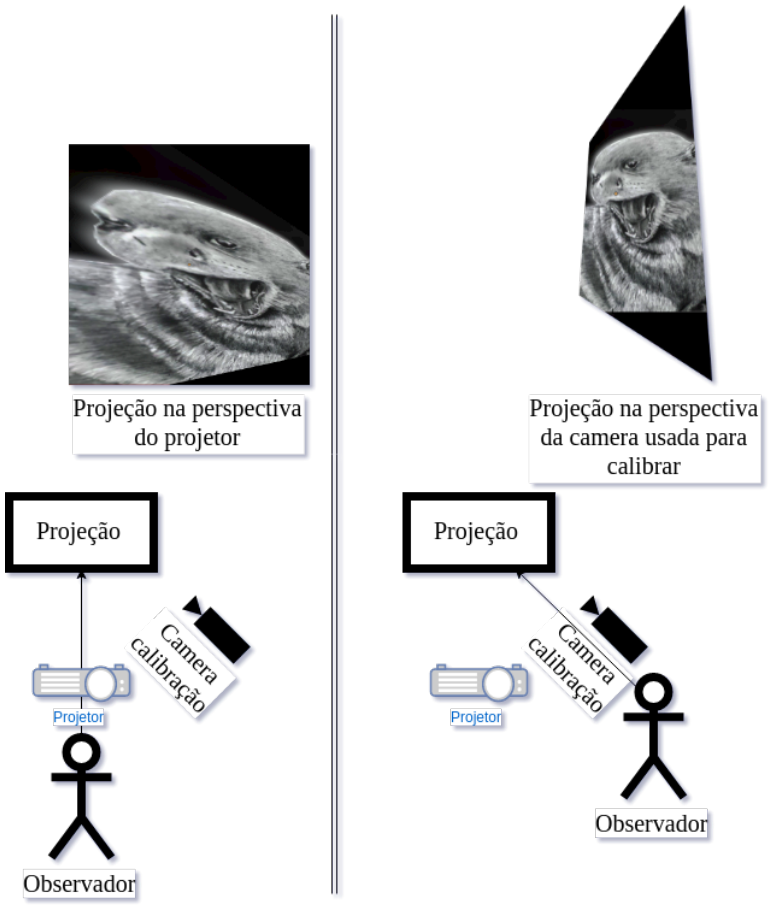


Figura 27 – Exemplo da correção de perspectiva, após aplicada a uma fotografia na malha geométrica de UV gerada com os pontos de perspectiva da câmera de calibração. Vídeo: <https://youtu.be/8GpSU-Ngzlw>

Fonte: Autoria própria.

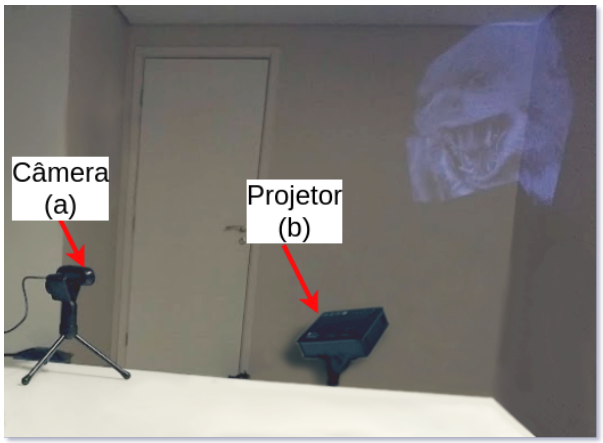


Figura 28 – Organização da cena de teste de calibração.

Fonte: Autoria própria.

capturar as imagens dos marcadores. Dessa maneira, ao se observar uma imagem do ponto de vista da câmera usada para calibrar, a imagem estará sem distorção como mostra o

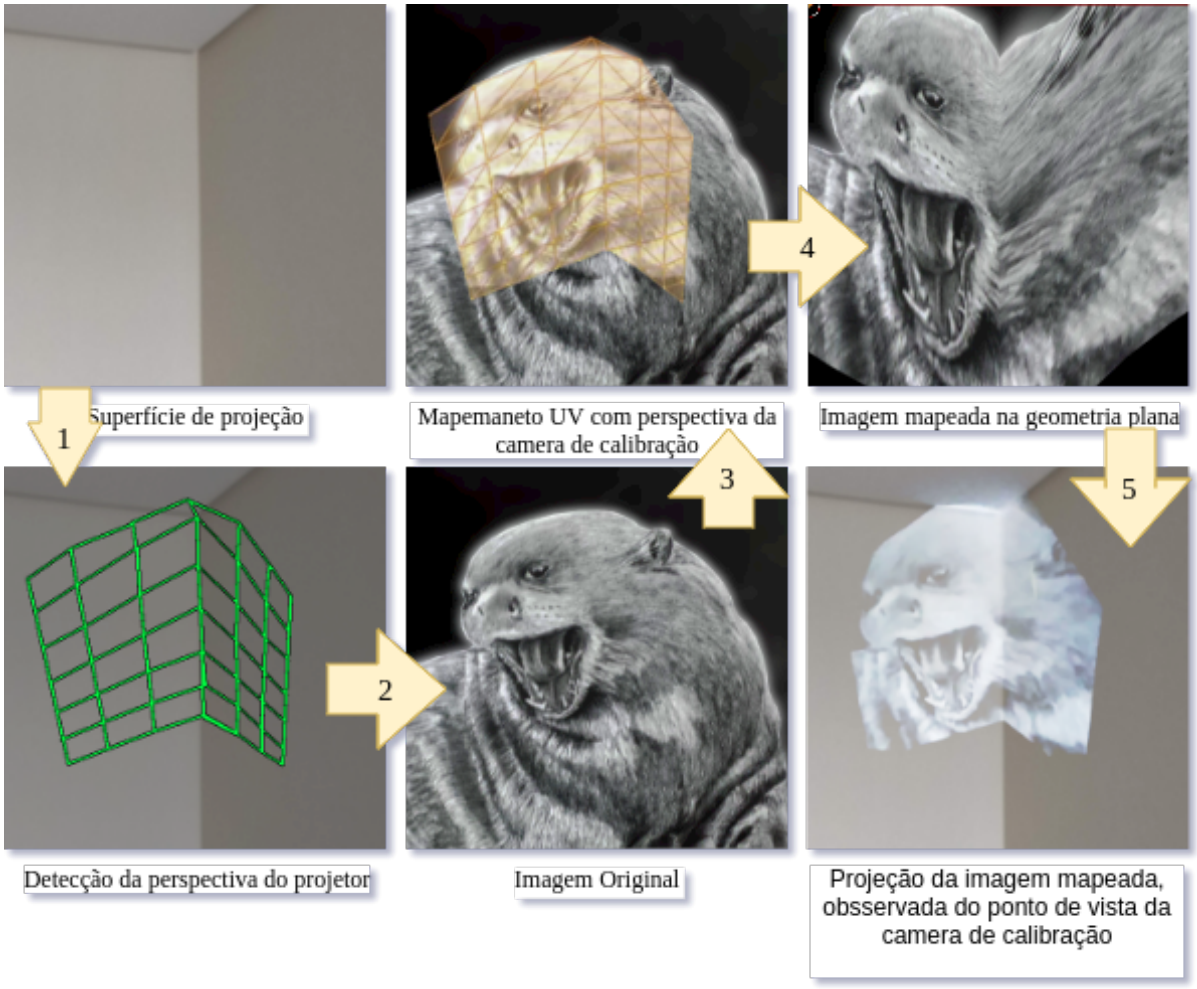


Figura 29 – Exemplo de fluxo do processo de transposição de perspectiva de projeção.

Fonte: Autoria própria.

exemplo na figura 27. Adicionalmente, a figura 30 mostra como seria uma projeção com e sem a transposição de perspectiva.

Uma característica importante desse processo é que funciona para diversos tipos de superfícies de projeções, seja plana, esférica ou cilíndrica, podendo assim ser usado para calibrar tanto ambientes *caves* quanto *fulldome*. A figura 29 ilustra um fluxo de um exemplo mais complexo, usando 3 superfícies planas.

As imagens nos exemplos das figuras 29 e 30, apresentam pequenas falhas, causadas ambas por poucas iterações com marcadores de modo a tornar o processo mais rápido, somado ao fato de a superfície ter curvas abruptas. Para superfícies com curvatura de 90° ou mais, é ideal mais iterações de marcadores, para tornar o mapeamento mais preciso. Apesar disso, esses exemplos, com apenas 7 iterações (malha 7 por 7) e em uma superfície bastante complexa de mapeamento o resultado ainda foi aceitável. Para o objetivo final do trabalho, que é mapear uma cúpula, o processo será mais simples, visto que a superfície da cúpula possui uma curva suave e sem complexidades. O mesmo vale para caves cilíndricas, onde a

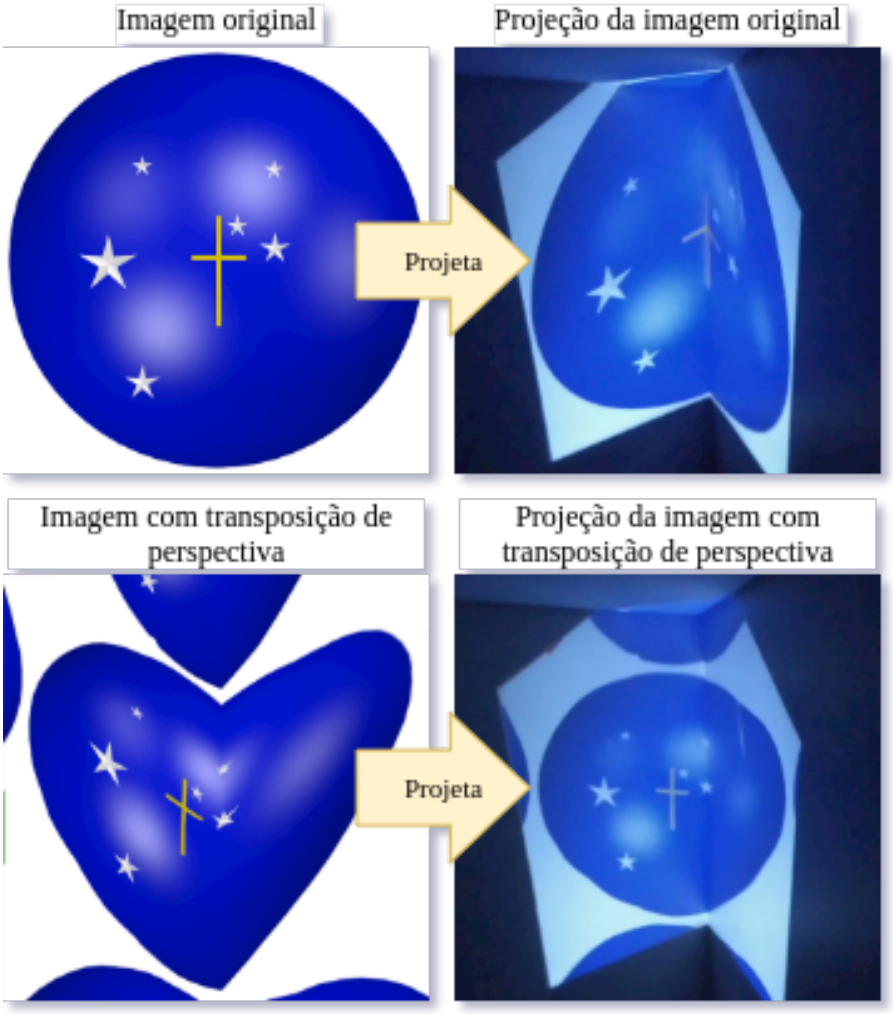


Figura 30 – Diferenciação de uma projeção com transposição de perspectiva e sem.

Fonte: Autoria própria.

curvatura é contínua e suave, não representando grande dificuldade para se determinar a posição dos marcadores.

Ao ser feita uma calibração, as informações sobre a mesma devem ser salvas para que, na próxima vez que o *middleware* iniciar, este retome a configuração da calibração feita anteriormente. Esses parâmetros são escritos em um arquivo *JSON*. Esse mesmo arquivo serve também para distinguir canais de projeção estéreo, possibilitando assim a visualização estereoscópica. Estereoscopia trata sobre a visualização de uma cena com a noção de profundidade. Esse efeito é possível quando o observador visualiza a cena com um ponto de vista distinto para cada olho. Esse deslocamento de ponto de vista é a distância paralaxe.

Além do arquivo *JSON* servir como uma forma de retomar a configuração da calibração, ele permite que o *middleware* trace um perfil para diferentes tipos de ambiente *fulldome* que não o da Arena Digital. Dessa forma o *middleware* consegue funcionar em

diferentes configurações de ambientes imersivos.

4.1.5 Manipulação de Janelas

Esta seção descreve como o quinto objetivo específico, sobre interface, é resolvido. É criado um componente para mover janelas ao redor do observador no ambiente imersivo, que também é utilizado como base para observação da cena tridimensional.

O sistema, terá dentre algumas funcionalidades, mostrar janelas, as quais irão comportar *slides* ou até mesmo aplicações, tais como, navegadores *web*, isso tudo dentro do ambiente imersivo. Para que isso aconteça, é necessário um método para mover as janelas em torno do ambiente, similar a um gerenciador de janelas, mas limitando-se apenas em mover janelas e ampliar ou diminuir o tamanho das mesmas. Nesse sentido, criou-se um componente que usa do sistema de coordenadas esféricas para posicionar um plano, a janela, em torno de um ponto O, o observador. Isso possibilita, criar um sistema baseado em três dimensões, o que é essencial para o nosso sistema, visto que, o mesmo se baseia em plataformas de visualização omnidirecionais tal como o *fulldome*.

Esse componente é reutilizável. Ele está sendo usado no sistema gerenciador de janelas, para mover as janelas em torno do observador. Também é usado na navegação da cena, para mover o observador em torno de um objeto tridimensional na cena 3D.

O sistema de coordenadas esféricas, similar ao cartesiano tridimensional, tem um ponto representado por um terno ordenado mostrado na Equação 4.1:

$$P = (\rho, \theta, \phi) \tag{4.1}$$

A Figura 31 mostra os elementos básicos em um sistema de coordenadas esféricas, onde temos: $\rho = OP$, a distância de O até P. θ a colatitude de P, ou seja, a medida do ângulo que o eixo y forma com OP. ϕ é a longitude ou azimute de P, ou seja, a medida do ângulo que o plano XY forma com o plano de bordo α que contém P.

O componente implementado trabalha basicamente com esses dois pontos, O e P, onde P "gira" em torno de O. Se tomássemos como referência a Figura 31, para simplificar, o sistema de coordenadas esféricas, assume que o ponto P pode ocupar qualquer ponto na superfície da esfera de raio ρ ilustrada (essa superfície está destacada em cores vermelho/cinza na Figura 31).

Usou-se o sistema de coordenadas esféricas pois ele é coerente com o conceito de janelas para o espaço 3D. Suponha o conceito atual de janelas nas interfaces gráficas dos sistemas operacionais para computadores, temos uma representação gráfica para o *mouse*, janelas etc. É trivial o conceito do funcionamento de uma interface gráfica em um monitor convencional. São elementos posicionados em um ponto no plano cartesiano de duas

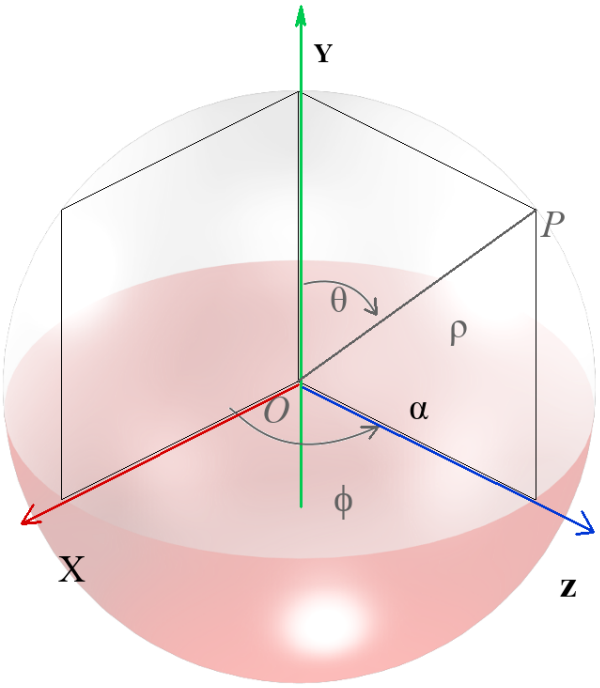


Figura 31 – Demonstração do sistema de coordenadas esféricas.

Fonte da ilustração: Autoria própria.

dimensões, pois a tela é plana. Suponha, no entanto, os mesmos conceitos, mas em uma tela que envolve o usuário em 360º. O sistema que se julga mais apropriado seria o esférico. Se considerarmos apenas a superfície da esfera (no sistema de coordenadas esféricas), podemos afirmar que funcionaria de forma similar ao cartesiano de duas dimensões. Os valores ϕ e θ do terno ordenado da Equação 4.1, tem tradução direta do X e Y do cartesiano de duas dimensões. Eles movem a janela longitudinalmente e latitudinalmente respectivamente. O valor ϕ do terno, altera o raio ou a distância entre O e P. Resulta basicamente na aproximação ou distanciamento do ponto observado, sendo assim, o efeito é de ampliação ou diminuição do elemento que se observa.

Adicionalmente, a *game engine Urho*, assim como as demais outras *game engines*, não trabalham com o terno ordenado do sistema de coordenadas esféricas. Para isso, devemos converter as coordenadas do sistema esférico para o cartesiano de três dimensões. Nesse sentido podemos usar as seguintes fórmulas para converter o terno ordenado do sistema de coordenadas esféricas para o sistema cartesiano de três dimensões:

$$X = \rho \sin\theta \cos\phi \tag{4.2}$$

$$Y = \rho \sin\theta \sin\phi \tag{4.3}$$

$$Z = \rho \cos \theta \tag{4.4}$$

Com relação a *slides*, refere-se aos arquivos de apresentação similares ao *PowerPoint* da *Microsoft*. Integrado com o *middleware*, está uma ferramenta para apresentar slides. Ela é básica e não suporta recursos como reprodução de vídeos nem transições, mas o propósito é garantir o mínimo para que seja possível fazer uma apresentação com slides na plataforma imersiva.

Está fora do escopo desenvolver uma ferramenta para ler arquivos de apresentação *slide*, pois existem mais de um tipo para tal (.pptx, .pptm, .ppt, .pdf). Decidiu-se trabalhar com os *slides* como sendo meramente imagens. As imagens seriam salvas em uma pasta qualquer, um arquivo *JSON* por sua vez definiria um objeto "*Slides*", listando o conjunto de imagens da pasta citada. O arquivo *JSON* é um padrão normalmente utilizado por aplicações *web* para serialização e transporte de objetos e informações pela rede. A partir desse arquivo *JSON* com o objeto de *Slides* definido, escreve-se uma aplicação em *AngelScript* que carrega o arquivo de apresentação *JSON*. Foi necessário implementar na *API AngelScript* do sistema uma funcionalidade para carregar essas apresentações *slides*.

A interface para interação com essa aplicação é uma página *web* que possui botões para passar slides, movê-los em torno da tela do domo e ampliar o tamanho do mesmo como mostra a Figura 32.

O movimento do *slide* ao longo da tela do ambiente imersivo é feito pelo componente de gerenciamento de janelas. Ao usar os controles de movimento mostrados na Figura 32, o *slide* move-se longitudinalmente e latitudinalmente. O controle de *zoom*, altera o raio do sistema esférico, criando a impressão de ampliação ou diminuição do *slide*.

4.1.6 Integração com plataformas da *internet*

Esta seção mostra como o sexto objetivo específico foi feito, sobre integração de plataformas de conteúdo de realidade virtual com o ambiente imersivo. Como mencionado no item 1.3, existe grande quantidade de conteúdo imersivo disponível em plataformas tal como o *YouTube* e um dos objetivos deste trabalho é propor uma forma de integrar essas plataformas com ambientes imersivos.

É fato que, a internet é a principal plataforma para consumo de conteúdo de mídia digital. A conclusão lógica para o trabalho foi integrar a mesma com o ambiente imersivo. Uma das tecnologias que realiza essa tarefa é o navegador *web*. Neste sentido, foi integrado um navegador no sistema, como mostra na Figura 33.

O *Chromium Embedded Framework (CEF)*[9] é uma biblioteca que abstrai funciona-

[9] <https://bitbucket.org/chromiumembedded/cef> visto em 21/02/2020

Figura 32 – Ilustração de como é a interface gráfica do controle de slides.

Fonte da ilustração: Autoria própria.

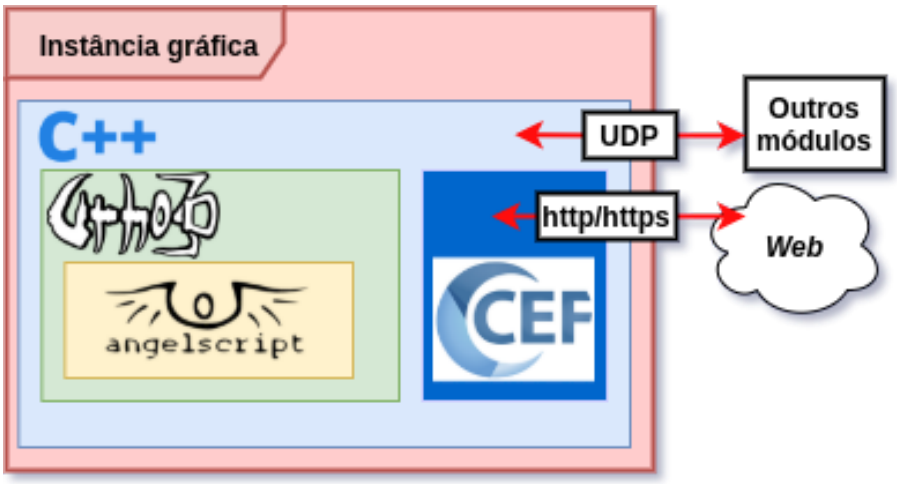


Figura 33 – Representação gráfica dos componentes tecnológicos usados no sistema, destaque de onde situa-se o *CEF*.

Fonte da ilustração: Autoria própria.

lidades do navegador *web Chromium*[10], feita com o propósito de ser integrado em *software*

[10] <https://www.chromium.org/Home> visto em 21/02/2020

existente, fazendo com que o mesmo tenha funcionalidades de um navegador. Geralmente encontra-se o *CEF* embutido em *softwares* convencionais que requerem reprodução de conteúdo da *web* em seu cliente/programa local, ou desejam usar interface gráfica *HTML* ao invés de bibliotecas convencionais tal como *Qt*[11] ou senão, estão convertendo uma aplicação *web* para um aplicativo convencional. Foi feito então a integração do CEF com o sistema proposto.

Com a adição de um navegador *web* pode-se reproduzir a maioria dos arquivos digitais de mídia presentes na *internet*, tais como arquivos de vídeo *.mp4, .wmv, .webm* e áudio *.mp3, .ogg, .flac* etc. O uso de um navegador agrega muito valor ao sistema final, pois se extingue a necessidade de adicionar/implementar *codecs* no sistema para a reprodução de cada um destes arquivos de mídia. Além do sistema possibilitar uma aplicação ser feita em *AngelScript*, com o navegador, aplicações finais poderão ser feitas em *JavaScript* de modo convencional, como qualquer outra aplicação para *web* é feita. Isso adiciona mais possibilidades de se trabalhar com o sistema. Eventualmente, o usuário pode não ver o valor de se desenvolver uma aplicação em *AngelScript* ou C++ por qualquer razão, mas ao mesmo tempo, entende que aplicações *web*, escritas usando *frameworks JavaScript* são uma tendência. Um exemplo de que navegadores *web* são cada vez mais comumente utilizados para desenvolvimento de aplicações convencionais é o *Framework Electron*[12], o qual, também utiliza de forma similar ao *CEF*, tecnologia do *Chromium*. Alguns dos aplicativos mais notórios desenvolvidos com o *Electron* são: A *IDE* de desenvolvimento *Visual Studio Code*[13], os aplicativos de mensagens *Skype*[14], *Dirscord*[15], *Slack*[16] entre outros.

Internamente no sistema, integrado com o *CEF*, múltiplas *threads* pertencem ao *Chromium*, as quais ele mesmo gerencia. O restante do sistema apenas executa algumas chamadas de função da biblioteca *CEF*, para, por exemplo, carregar *URL*, repassar eventos de teclas pressionadas, etc. A respeito da interface com a *Urho3D* é feito uma classe que corresponde a algumas funções do *CEF* para, no mínimo, carregar *URL*s e repassar regiões desenhadas da página *web* para uma textura na *Urho3D*. Isso por sua vez, permite que seja criada uma geometria tridimensional na *Urho3D*, cuja textura corresponde a imagem desenhada pelo processo do *Chromium*. Dessa forma, o navegador renderiza dentro da própria *Urho3D* como mostra na Figura 34.

O plano que representa o navegador pode mover-se ao redor do ambiente, ampliar e diminuir de tamanho, similar ao do componente de *slides* (subseção 4.1.5). O movimento do navegador ao longo da tela do ambiente imersivo é feito pelo componente de gerenciamento de janelas, descrito na subseção 4.1.5. A interface de controle também é bastante similar a

[11] <https://www.qt.io/> visto em 21/02/2020
[12] <https://www.electronjs.org/> visto em 21/02/2020
[13] <https://code.visualstudio.com/> visto em 21/02/2020
[14] <https://www.skype.com/> visto em 21/02/2020
[15] <https://discordapp.com/> visto em 21/02/2020
[16] <https://slack.com> visto em 21/02/2020

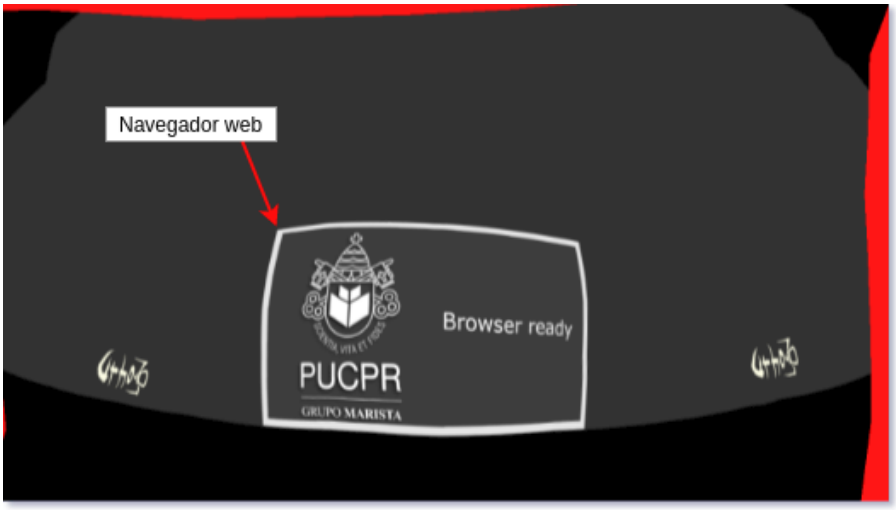


Figura 34 – Navegador integrado na *Urho3D*.

Fonte: Autoria própria.

interface do controle de *slide* mostrado na Figura 32.

A partir desse ponto, o navegador está integrado com o sistema. Ele é mostrado dentro da própria *Urho3D*, como se fosse uma textura animada. Foi reaproveitado o mecanismo de visualização descrito no item 4.1.5 para que fosse possível ancorar o navegador em um elemento, que pudesse ser deslocado ao longo da tela de visualização *fulldome*.

Figura 35 – Um *frame* de um vídeo 360º do YouTube mostrando como o mesmo é armazenado pela plataforma.

Fonte: Ryan Whitehead (Canal YouTube); <https://www.youtube.com/watch?v=1_ifgJqLqTY>

A primeira função testada com o navegador foi abrir vídeos diretamente do YouTube, uma tarefa simples, visto ser necessário apenas executar uma *URL* com o endereço do vídeo no YouTube. O navegador faz todo o restante do trabalho. Funciona como em um

navegador convencional.

Mapeamento cilíndrico equidistante

Mapeamento tipo Cube Map utilizado pelo youtube

Figura 36 – Comparação entre uma imagem 360º em ambos os mapeamentos cilíndrico equidistante e *cube map* do YouTube.

Fonte: Ryan Whitehead (Canal YouTube); <https://www.youtube.com/watch?v=1_ifgJqLqTY>

Como o foco do trabalho são ambientes imersivos, mais especificamente *fulldome*, a segunda função testada no navegador foi a de visualizar um vídeo 360. Propositalmente executou-se um vídeo 360º, na superfície de visualização plana, a fim de entender como o mesmo se mostraria. A Figura 35 mostra como é mapeado um vídeo 360º no *YouTube*. Repare que, apesar do vídeo ser 360º, o mesmo deve ser armazenado de alguma forma convencional. Um vídeo 360º é apenas um vídeo normal, uma sequência de imagens bidimensionais, cartografadas de uma maneira específica, não importando exatamente como é o mapeamento pois existem diversas maneiras de fazer isso.

Os vídeos de 360º do YouTube possuem relação 2:1, ou seja, a largura é o dobro da altura, independentemente da resolução. É dividido em 6 regiões: frente, esquerda, direita, para cima, para baixo e atrás. Cada segmento é uma imagem de 90º de ângulo de perspectiva. A junção delas forma a imagem panorâmica. Em termos menos formais, é uma espécie de *Cube map*[17]. Resumidamente, *Cube Mapping*, em computação gráfica, é o processo de mapear o ambiente (cena), sobre as faces de um cubo, resultando assim, em 6 imagens cada uma com direções simultaneamente perpendiculares entre si. O YouTube

[17] <https://learnopengl.com/Advanced-OpenGL/Cubemaps> visto em 21/02/2020

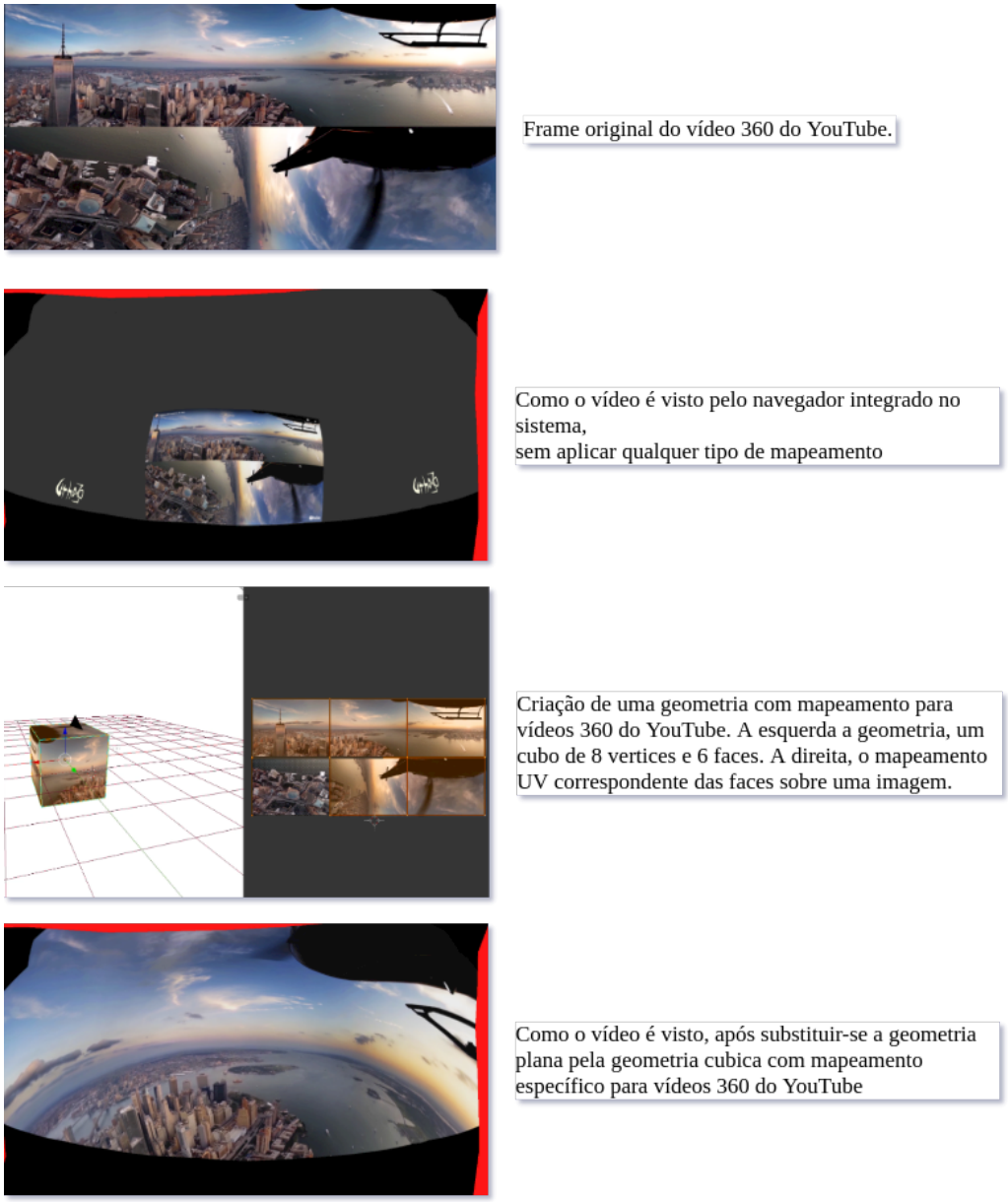


Figura 37 – Visualização simplificada de como obteve-se visualização 360º a partir do uso de uma geometria de mapeamento UV apropriado.

Fonte da ilustração: Autoria própria.

usou deste artifício para mapear vídeos 360º. Outro formato normalmente utilizado é o mapeamento cilíndrico equidistante, como mostrado na Figura 36. Uma razão para se usar o mapeamento cúbico ao contrário da cilíndrica equidistante, seria uma melhor distribuição de *pixels*. No mapeamento cilíndrico equidistante, os polos concentram poucos *pixels*. Os mesmos são esticados para haver preenchimento.

Levantou-se duas formas de mapear a imagem para a visualização em 360º. Uma era usar dos recursos do *driver* gráfico (*OpenGL*, *Direct3D*) e modificar a matriz de visualização da câmera para formar a visualização 360º. A outra era criar uma geometria com mapeamento *UV* correspondente. Optou-se pela segunda, por ser mais simples de

manter o código e mais flexível, visto que não exige conhecimento de baixo nível sobre *drivers* gráficos. Além disso, já está sendo feito o uso de um *Game Engine* (*Urho3D*). A primeira solução seria na circunstância de não haver outro recurso senão o *driver* gráfico. Dito isso, criou-se um cubo, com mapeamento UV correspondente ao utilizado pelo *YouTube*. Todo o processo é similar ao descrito na subseção 4.1.3. Ao invés de um domo virtual, tem-se um cubo virtual, com uma câmera observando no centro da geometria como mostra a figura 37, com o respectivo resultado.

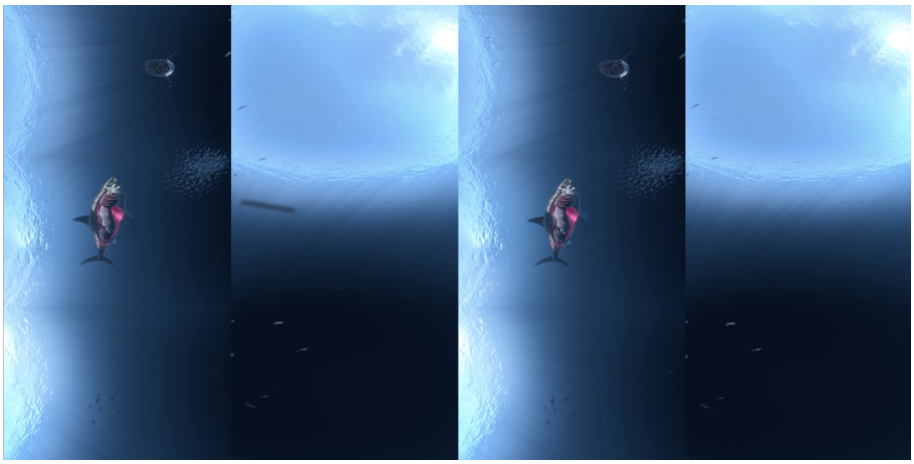


Figura 38 – Um *frame* de um vídeo 360º 3D do YouTube mostrando como o mesmo é armazenado pela plataforma.

Fonte: Curiscope (Canal YouTube); <https://www.youtube.com/watch?v=HNOT_feL27Y>

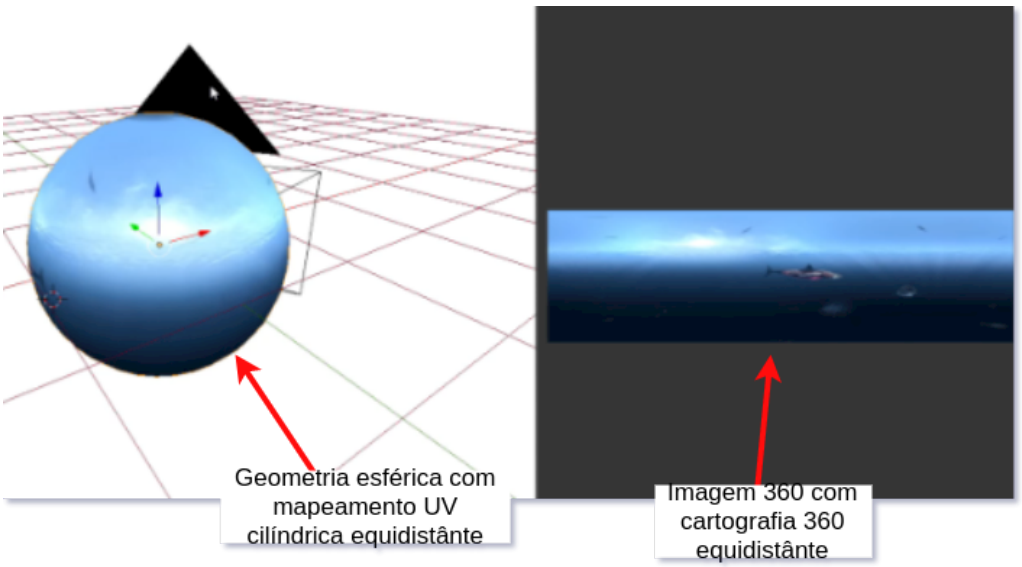


Figura 39 – Geometria criada para mapear vídeo de cartografia cilíndrica equidistante.

Fonte da ilustração: Autoria própria.

A mesma abordagem pode ser utilizada para os demais formatos de vídeo 360º,

como é o caso dos vídeos de mapeamento cilíndrico equidistante. Criou-se uma geometria esférica, com mapeamento cilíndrico, como mostra a Figura 39.

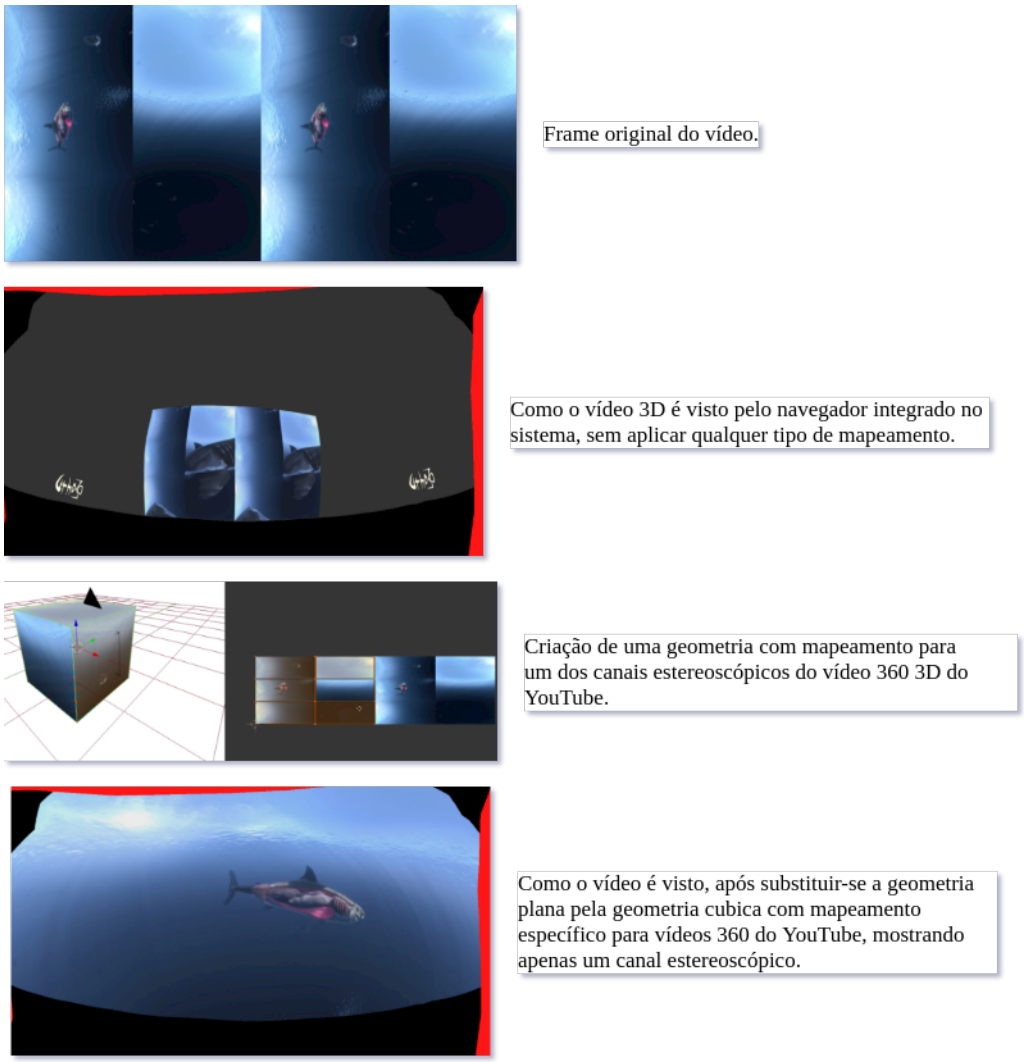


Figura 40 – Visualização simplificada de como obteve-se visualização de um dos canais do vídeo 360º 3D do YouTube, a partir do uso de uma geometria de mapeamento UV apropriado.

Fonte da ilustração: Autoria própria.

Por fim, no *YouTube* existem vídeos 360º 3D. Ou seja, vídeos imersivos estereoscópicos. O seu mapeamento é o mesmo do mapeamento cúbico utilizado, reorientado para acomodar 2 canais, como mostra a Figura 38. Criou-se duas geometrias, uma mapeando cada um destes canais separadamente, de modo a termos a opção de separar uma da outra para projetar separadamente. A Figura 40 mostra como é mapeado o *frame* do vídeo 360º 3D do YouTube. Dessa maneira consegue-se criar *viewports* com canais distintos. Esse é o único pré-requisito para um ambiente com capacidade de reproduzir mídia estereoscópica poder funcionar. O sistema de projeção estereoscópico precisa apenas das imagens separadas em *viewports* diferentes. O restante do trabalho é feito pelo *hardware*

do ambiente imersivo. Dessa forma, o sistema reproduz vídeos convencionais, 360º e 360º 3D, diretamente do YouTube, ou arquivo de mídia suportados pelo *Chromium*.

4.2 Exemplo de aplicação para visualização

4.2.1 Exemplo de aplicação para visualização volumétrica

Este item refere-se ao sétimo objetivo específico. Será mostrado um exemplo implementado com o *middleware*, com propósito de demonstração técnica. O exemplo trata da visualização de um volume 3D, criado a partir de imagens do *Visible Human Project*. É similar a uma visualização de volume de ressonância magnética (RM) 3D ou tomografia computadorizada (TC) 3D. Como as imagens do *Visible Human Project* são fotografias de criosecção axiais, o volume resultante tem coloração natural, ao contrário de um volume de RM/TC colorido artificialmente. O resultado é visualmente mais fiel ao objeto real.

Inicialmente, não é possível trabalhar com as imagens originais do *Visible Human Project*, visto que a base é muito grande e não poderia ser carregada em memória para executar o programa. Como detalhado na subseção 3.1.3, cada imagem de corte do *Visible Human Project*, possui tamanho aproximado de 33.2 *megabytes* quando em formato *Bitmap* (arquivo *.bmp*) codificado em 24 *bits*. Na codificação de 24 *bits* do *Bitmap*, para cada pixel da imagem, um byte é usado para armazenar o valor de cada canal de cor, usando o espaço de cor *RGB* (*Red, Green* e *Blue*). Ou seja, em um *pixel* na imagem, o valor da intensidade de cada um dos três canais de cores estará armazenado em um *byte*. Para o computador, um *byte* é a combinação de oito *bits*, que por sua vez é um valor binário representado por 0 ou 1. O número de combinações que podem ser feitas com o sistema binário pode ser calculado pela Equação 4.5.

$$C = N^2 \tag{4.5}$$

Na Equação 4.5, C é o número de combinações possíveis com N dígitos binários. Nesse sentido, um *byte*, que é um conjunto de 8 *bits*, possui 256 combinações. Cada canal de cor no espaço de cor *RGB* do *Bitmap* pode variar sua intensidade de 0 a 255. Considerando que existem três canais de cores 8 bits, somando 24 *bits*, se usarmos a Equação 4.5 usando $N = 24$, verificamos que é possível representar 16777216 cores nessa codificação.

O principal problema de usar o arquivo *Bitmap* codificado em 24 *bits* é o tamanho por arquivo, que fica grande, se considerarmos que deverão ser carregadas 5189 (quantidade de imagens na base de imagens feminina do *Visible Human Project*) imagens na aplicação, o que resultaria aproximadamente 173 *gigabytes*.

Nesse sentido, é feito o seguinte procedimento para reduzir o tamanho do conjunto de imagens do *Visible Human Project* para o espécime feminino.

Converte-se o arquivo do formato *Bitmap* para *JPEG*. É possível apenas com esse procedimento, reduzir o arquivo, antes de 33.2 *megabytes* para 18 *megabytes*, o que soma 94 *gigabytes* de tamanho.

Em seguida, para cada imagem do conjunto, é reduzida sua dimensão, de 4096 *pixels* por 2700 *pixels* (resolução *4K*), para 1920 *pixels* por 1266 *pixels* (*Full HD*). A imagem resultante fica com tamanho de 2,7 *megabytes*, somando 15 *gigabytes* de tamanho para o conjunto todo. Esse processo, no entanto, resulta na perda de resolução das imagens, com diminuição dos detalhes na base resultante.

Nesse ponto do processo, já é possível trabalhar com esse volume de dados no *hardware* da Arena Digital. Porém, não haveria memória suficiente para carregar todas as 5189 imagens de uma só vez. Seria necessário ocultar algumas de modo a economizar memória. Se desejássemos reduzir ainda mais esse volume de dados para poder visualizar todas as imagens, mantendo uma resolução *High Definition* (*HD*), poderíamos redimensionar a imagem até o mínimo de 1440 *pixels* por 950 *pixels*, que é maior que o *form factor HD*. Esse dimensionamento resultaria em imagens de 1,5 *megabytes*, somando 7,8 *gigabytes* para o conjunto todo.

Adicionalmente, pode ser feita uma otimização na imagem, de modo a reduzir ainda mais o tamanho (em *megabytes*) da mesma, sem haver perda de detalhes. Esse procedimento é feito através do recorte de imagens, como uma forma de "destacar" uma região contida na imagem original. A Figura 41 ilustra a diferença entre o processo de recorte e redimensionamento de imagens.

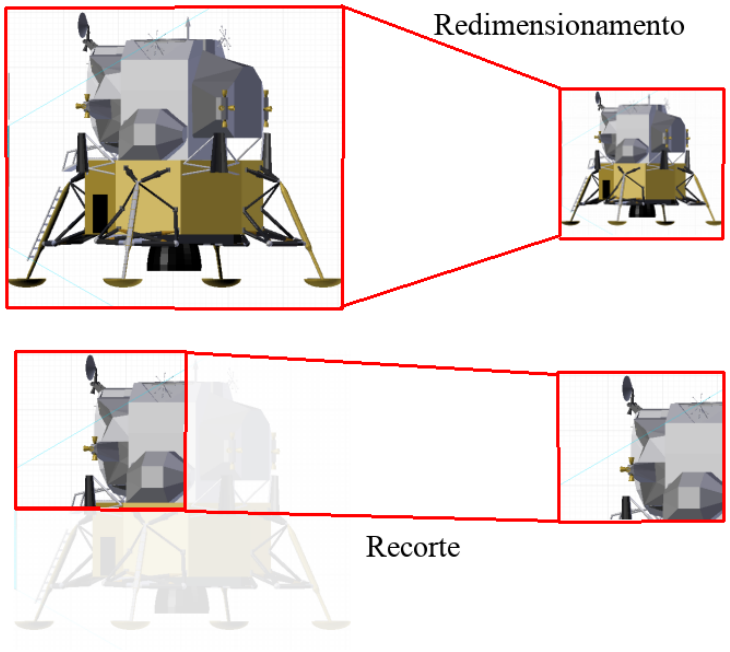


Figura 41 – Representação da diferença entre recorte de imagens e redimensionamento de imagens.

Fonte: Autoria própria.

Algoritmo utilizado para recorte de imagens:

Algoritmo 1: Enquadramento do conjunto

```
1  X1 ← Y1 ← infinito();
2  X2 ← Y2 ← 0;
3  for img <= numImagensConjunto(); ++img do
4      for i <= larguraImagem(); ++i do
5          for j <= alturaImagem(); ++j do
6              if pixel(img, i, j) != Vazio() then
7                  if i « X1 then
8                      X1 ← i;
9                  if i » X2 then
10                     X2 ← i;
11                 if j « Y1 then
12                     Y1 ← j;
13                 if j » Y2 then
14                     Y2 ← j;
```

O Algoritmo 1 mostra de maneira simplificada como podemos realizar um enquadramento da informação contida nas imagens do *Visible Human Project*. Ele determina uma caixa delimitadora mínima (no inglês, *Bounding Boxes*) absoluta considerando todo o conjunto de imagens do modelo do *Visible Human Project*. Essa caixa delimitadora é o espaço com a menor medida (área, volume, ou hiper volume) possível que englobe todos os pontos (*pixels*).

O Algoritmo 1 funciona iterando todas as imagens do conjunto. Para cada uma, é varrido cada um dos pixels da mesma, de maneira que, é armazenada a informação do pixel mais à esquerda ($X1$) e o mais à direita ($X2$) (no eixo das abscissas). Também é armazenado o pixel mais acima ($Y1$) e o mais abaixo ($Y2$) (no eixo das ordenadas). Com essas quatro variáveis, é possível determinar dois pontos $P1 = (X1, Y1)$ e $P2 = (X2, Y2)$, uma sub-região. A sub-região determinada pelos dois pontos então é recortada em todas as imagens do conjunto que forma modelo do *Visible Human Project*.

O resultado é um novo conjunto de imagens, de menor dimensão, mas que preservam a qualidade da informação das imagens antes de aplicar o processo de enquadramento.

A construção do volume tridimensional do espécime feminino da base de imagens do *Visible Human Project* foi feita por meio do "empilhamento" das imagens (cortes).

Tomando como base uma representação cartesiana do espaço em três dimensões: X (largura), Y (altura) e Z (profundidade), carrega-se em ordem na memória as imagens dos cortes de criosecção, já processadas pelo procedimento descrito anteriormente. Para cada iteração soma-se o intervalo entre as fotografias das secções criogênicas: 0,33 milímetros para o espécime feminino.

Algoritmo 2: Reconstrução Modelo Virtual

```
1 for Zi = 0; Zi <= QuantidadeImagensVHP(); Zi ← ++Zi do
2     Mp ← criaModeloPlano() ;
3     Mp ← atribuiTextura(VHP[Zi]) ;
4     Mp ← posiciona(0, Zi * intervaloSecção, 0) ;
```

O Algoritmo 2 mostra como é feito para carregar cada uma das fotografias de cortes na aplicação. Na linha 1 itera-se as imagens do conjunto. Nas linhas 2 e 3 são usadas as funções do motor de jogo *Urho* para carregar um modelo de um plano e em seguida, atribuir uma imagem da base do *Visible Human Project*. A linha 4 posiciona o modelo do plano em uma altura proporcional ao intervalo do corte de criosecção (*intervaloSecção*). O resultado do processo é um modelo tridimensional com todos os cortes, representados por planos com a textura (fotografia) das criosecções "empilhados", de maneira que cria o objeto volumétrico em três dimensões como visto na *Figura 42*[18].

[18] Fonte: <https://www.youtube.com/watch?v=MgEEJWsn2o8> visto em 01/01/2019

Figura 42 – Simulação da reconstrução do espécime feminino do *Visible Human Project* com uma sequência de duas mil fotografias de cortes.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: *National Library of Medicine* <https://www.nlm.nih.gov/>

A partir do volume construído é possível realizar cortes anatômicos no mesmo. Como visto em livros de anatomia, existe uma formalidade com relação aos planos de corte para a visualização das estruturas anatômicas. Os planos de corte normalmente empregados são o sagital, coronal, axial e oblíquo. Estudantes da escola de medicina usam referências de cortes para estudar anatomia humana, por isso é importante reproduzir os mesmos neste exemplo.

Dado o sistema de coordenadas cartesiano em três dimensões, definido por 3 retas simultaneamente perpendiculares entre si: X, Y, Z, onde X é a largura representado pelas abscissas, Y é altura -ordenada- e Z é a profundidade -cota-. Posicionamos um modelo humano "olhando" para o sentido do eixo da cota Z. O plano de corte sagital seria o plano YZ, o coronal seria o plano XY e o axial o plano XZ, como visto na Figura 43.

Para os cortes sagital e coronal são utilizadas *transformações*[19] nos modelos tridimensionais dos planos com as imagens de criosecções. Transformação são utilizadas em computação gráfica para alterar a posição, rotação, tamanho e forma dos objetos. Em cenas tridimensionais, os objetos são representados no computador por pontos, que formam

[19] <https://open.gl/transformations> visto em 8/03/2020

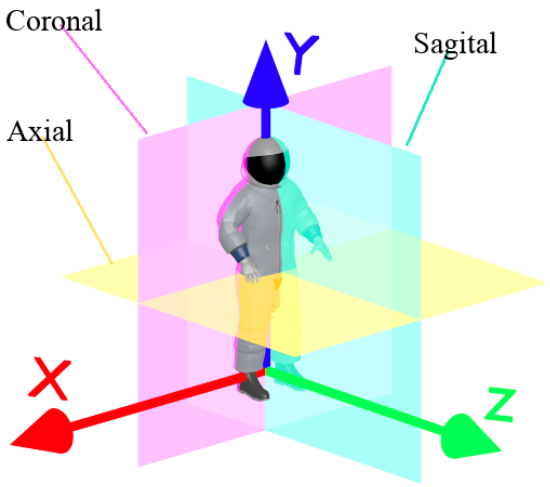


Figura 43 – Referência representando um espaço cartesiano com os planos de cortes anatômicos.

Fonte da ilustração: Autoria própria. Modelo: <https://nasa3d.arc.nasa.gov/models>

retas e faces. Ao executar um processo de transformação em um objeto da cena, são feitos cálculos para cada um dos pontos do objeto, alterando assim, as características do mesmo. Internamente, as transformações são resolvidas na *OpenGL* através de operações matriciais.

Como está sendo feito o uso de uma *game engine* (*Urho3D*) em nosso *middleware*, a mesma já realiza esses procedimentos de maneira automática, bastando apenas executar as funções implementadas na própria *game engine*. Para criar o efeito dos cortes anatômicos, foram utilizadas apenas as funções para translado, escala e mapeamento de imagens. Na *Urho3D* essas funções são *SetPosition*[20], *SetScale*[21] e *SetUVTransform*[22] as quais alteram a posição e tamanho do objeto, assim como o mapeamento da textura do mesmo. *SetScale* e *SetPosition* recebem como parâmetro três valores, referentes ao fator da transformação nas três dimensões. A função *SetUVTransform* tem parâmetros para alterar a rotação, posição e fator de repetição que uma textura é desenhada na superfície de um objeto na cena.

Supondo que será feito um corte sagital, é realizada uma transformação de escala e translado, ao longo do eixo X, em cada plano com imagem de criosecção. Simultaneamente, é alterado o mapeamento da textura na superfície de cada plano, na mesma proporção da transformação de escala e translado. A operação de escala irá alterar as dimensões dos planos com as imagens e a operação de translado irá compensar a operação anterior, pois

[20] <https://urho3d.github.io/documentation/1.4/class_urho3_d_1_1_node.html#aa2fd960b4c5641cd38771280a6b84fb8> visto em 8/03/2020
[21] <https://urho3d.github.io/documentation/1.4/class_urho3_d_1_1_node.html#ace951aa924dd417a731746cbf92ead06> visto em 8/03/2020
[22] <https://urho3d.github.io/documentation/1.6/class_urho3_d_1_1_material.html#a2470fc628c1bbdb6b52054076714ff41> visto em 8/03/2020

dependendo de onde está a *origem do objeto*[23] (referência para cálculo da transformação), pode comprometer o efeito. É alterado na mesma proporção o mapeamento das imagens dos objetos com as imagens de criosecção, de modo a alterar o fator de repetição. Caso isso não seja feito, o resultado visual se assemelharia a um "achatamento" do volume (formada pelas imagens de criosecção empilhadas) e não um corte. Alterando-se o sentido das transformações ao longo do eixo X irá criar o efeito do corte sagital em sentido oposto. O mesmo processo é feito para o corte coronal, mudando apenas o fator das transformações para o eixo Y.

Na base de imagens do *Visible Human Project*, os cortes criogênicos são axiais. Nesse sentido, para simular o corte anatômico axial do volume, basta ocultar os planos com as imagens das criosecções na medida da altura do corte (h_{AXI}) desejado, sendo que ($h_{AXI} \in \mathbb{N}|1 < Axi < nImagens$), onde $nImagens$ é a quantidade de imagens axiais e Axi é o valor do corte axial.

Com relação ao corte oblíquo, na *OpenGL*, a característica da projeção da câmera é definida por parâmetros em uma matriz, os quais, descrevem uma região em forma de tronco de pirâmide, chamada de *espaço da câmera*[24]. Esse espaço (dentro do tronco de pirâmide) determina o que será mostrado ao ser renderizado a cena, de maneira que, tudo o que estiver dentro dessa região irá aparecer na renderização e tudo o que estiver fora, não será mostrado na renderização da cena pela câmera.

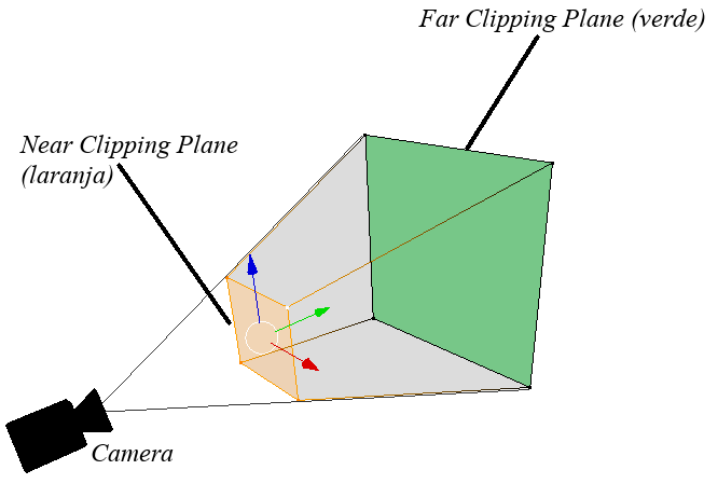


Figura 44 – Demonstração do tronco de pirâmide que representa o campo de visão para renderização de uma cena por uma câmera no *OpenGL*.

Fonte: Autoria própria.

[23] <https://docs.blender.org/manual/en/latest/scene_layout/object/origin.html> visto em 8/03/2020
[24] <http://ogldev.atspace.co.uk/www/tutorial13/tutorial13.html/> visto em 28/02/2020

A Figura 44 mostra uma representação do tronco de pirâmide que representa a área de renderização de uma câmera (espaço da câmera). Em destaque estão os planos de delimitação da distância de renderização mínima e máxima, representados pelo *Near Clipping Plane* e *Far Clipping Plane*. Esses planos basicamente definem uma distância mínima que uma câmera consegue renderizar um determinado objeto na cena, assim como a distância máxima que o objeto pode estar para que ainda possa ser renderizado. Esse mecanismo existe para controlar o uso de recursos pela aplicação gráfica. Existe a função *SetNearClip*[25] na *Urho3D* a qual altera esses valores da matriz de projeção de perspectiva da câmera.

Para simular o corte oblíquo do volume na *Urho3D* basta executar a função *SetNearClip* para alterar o *Near Clipping Plane* (distância mínima para haver a renderização) da câmera que observa o volume. Dessa maneira, se parte do conjunto de planos (objetos com imagens de criosecção) que constituem o volume do *Visible Human Project* interseccionam com o plano de distância mínima para renderização, a parte fora da região do espaço da câmera (tronco de pirâmide) não será renderizado pela *OpenGL*, criando assim o efeito do corte, perpendicular a direção de visualização da câmera como visto na Figura 45.

Figura 45 – Visualização do modelo seccionado em plano oblíquo.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: *National Library of Medicine* <https://www.nlm.nih.gov/>

A interface gráfica desse exemplo consiste apenas de dois elementos na tela do domo: o modelo virtual do objeto de estudo e uma área reservada para visualização de conteúdo auxiliar tal como o navegador *web* (descrito na subseção 4.1.6) ou *slides* (descrito na subseção 4.1.5).

[25] <https://urho3d.github.io/documentation/1.7/class_urho3_d_1_1_camera.html#a4ae07f2e2794c60a1a10f16931d9fca4> visto em 8/03/2020

O painel de controle para cortes anatômicos é um composto de cinco *sliders*[26] que possibilitam realizar os cortes axial, coronal, sagital e oblíquo assim como o ajuste da transparência do modelo. A Figura 46 mostra em detalhe os controles desse painel.

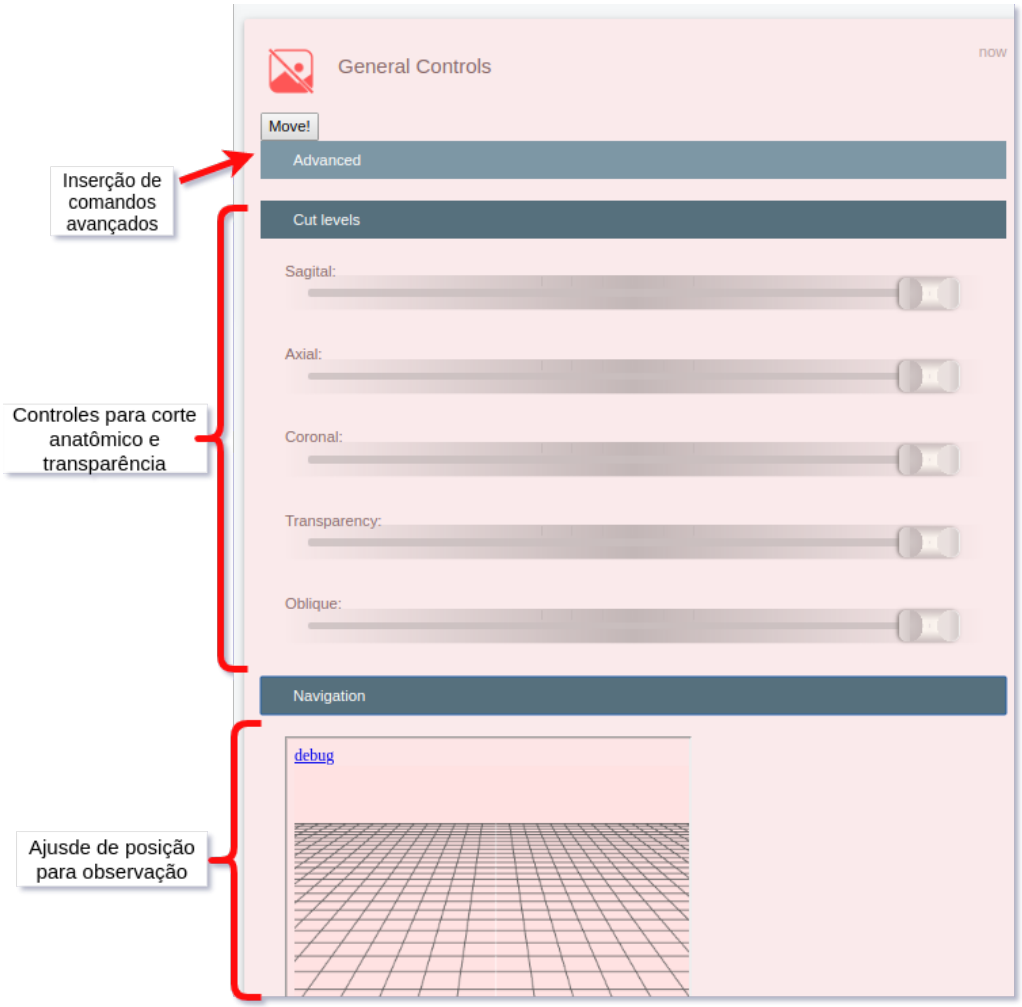


Figura 46 – Interface gráfica mostrando o painel de controle para cortes anatômicos da aplicação de exemplo de aplicação do *Visible Human Project*.

Fonte: Autoria própria.

A visualização em torno do volume do *Visible Human Project* se dá utilizando o componente que ajuda na órbita da câmera principal obedecendo ao sistema esférico de coordenadas como descrito na subseção 4.1.5. O modelo é tridimensional e, por ser construído por um processo de empilhamento de imagens, faz dele um objeto volumétrico, ou seja, o interior não é oco. Por esse motivo conseguimos realizar os cortes anatômicos e enxergar o interior do cadáver como se fosse um corpo real sendo dissecado. Adicionalmente é possível ajustar a transparência de cada plano com a foto de criosecção, de modo a tornar todo o modelo do objeto de estudo transparente como mostra na Figura 47

[26] <https://docs.microsoft.com/en-us/windows/desktop/uxguide/ctrl-sliders> visto em 29/01/2019

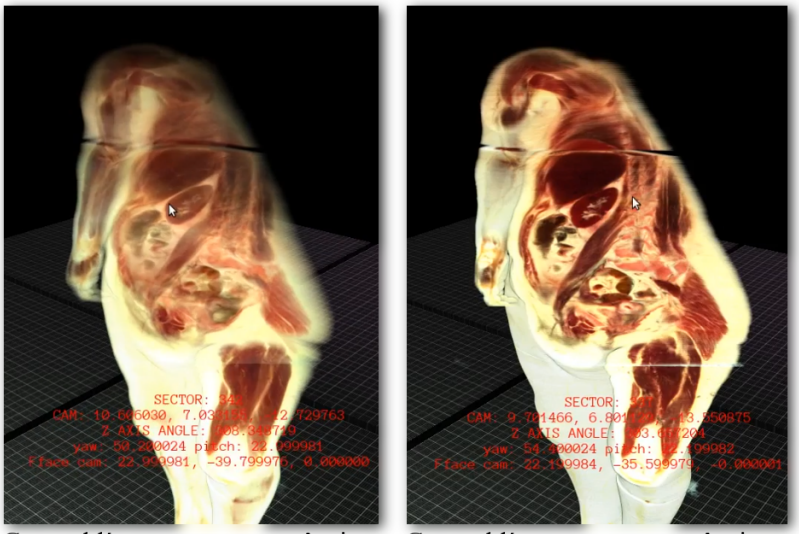


Figura 47 – Visualização do funcionamento da transparência para o modelo virtual.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: *National Library of Medicine* <https://www.nlm.nih.gov/>

O navegador ou *slider* por sua vez, seria usado principalmente como a área para apresentação de conteúdo auxiliar tal como *slides* ou vídeos do *YouTube*.

A Figura 48 mostra a interface gráfica para controle de cortes do modelo virtual através de um aparelho *smartphone* ou similar, acessando o painel de controle pelo navegador, similar ao processo já descrito na subseção 4.1.6.

4.2.2 Exemplo de aplicação para visualização de objeto de estudo virtual

Este item refere-se ao sétimo objetivo específico. Será mostrado outro exemplo de aplicação implementado com o *middleware*.

O exemplo baseia-se no uso de um conjunto de imagens as quais são feitas por meio da aquisição de fotografias ao redor de um objeto. Semelhante a um processo de escaneamento, este é um escaneamento bidimensional, em que uma foto é tirada em intervalos angulares em torno do objeto, formando uma base de imagens em 360º do mesmo. O escâner *F2S2* (SILVA et al., 2018) realiza tal aquisição de imagens. A partir de uma aplicação, é possível visualizar o objeto como se o mesmo fosse um objeto tridimensional, apesar deste ser apenas um conjunto de imagens. Os passos a seguir descrevem como realizar esse processo de visualização do objeto formado por um conjunto de imagens.

Carrega-se em ordem as imagens do escaneamento em um conjunto C; dada uma representação cartesiana do espaço em três dimensões: x (largura), y (altura) e z (profundidade), tomando como referência o plano cartesiano na imagem 43, define-se um ponto P onde iremos carregar o objeto de estudo.

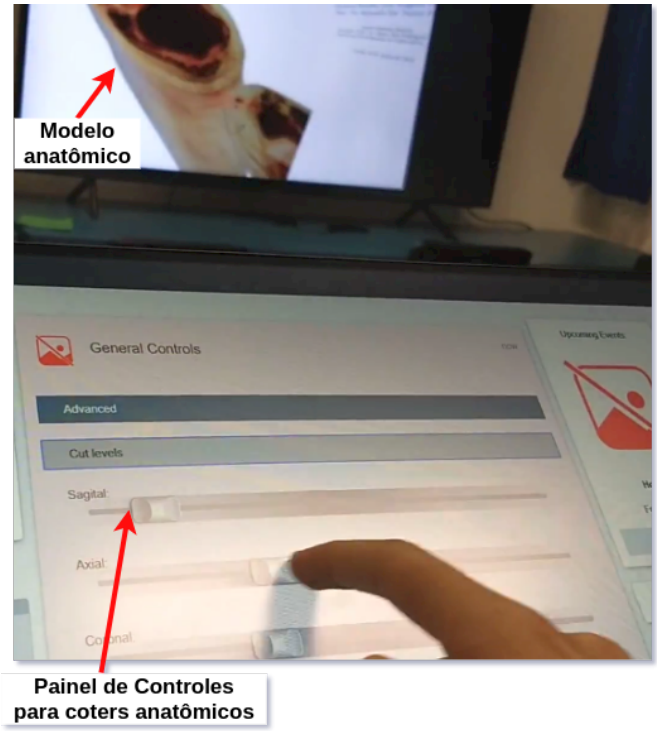


Figura 48 – Demonstração da interface gráfica do usuário para controle da aplicação de exemplo do *Visible Human Project*.

Fonte da fotografia: Autoria própria. Base de imagens anatómicas: <https://www.nlm.nih.gov/>

É necessário determinarmos qual imagem do conjunto C deve ser mostrada. Isso é feito referente a posição do observador O. Para isso, é calculado o ângulo θ entre o ponto P e o observador O, no plano XZ. A partir do ângulo θ obtido, calcula-se o índice I, que representa o índice da imagem no conjunto C.

$$\theta = tan^{-1}(\frac{x}{y}) \tag{4.6}$$

$$angulo = rad * (\frac{180}{\pi}) \tag{4.7}$$

Para simplificar, tome o seguinte exemplo: Um objeto de estudo escaneado de 400 imagens; o intervalo entre cada imagem é $360/400 = 0,9$, ou seja, tem-se uma imagem do objeto a cada 0,9º. Nesse sentido, supondo que um objeto e o observador respectivamente, estejam nas seguintes posições do plano cartesiano 3D (0,0,0) e (1,0,1), como mostra a figura 49, o ângulo θ formado pelas retas OP e o eixo X é deduzido por trigonometria básica como visto na fórmula 4.6. O resultado obtido do arco tangente é em radianos; converte-se então o resultado de radianos para graus com a fórmula 4.7, substituindo rad pelo valor de radianos. No exemplo, o valor do θ seria 0.785398163 radianos, convertendo

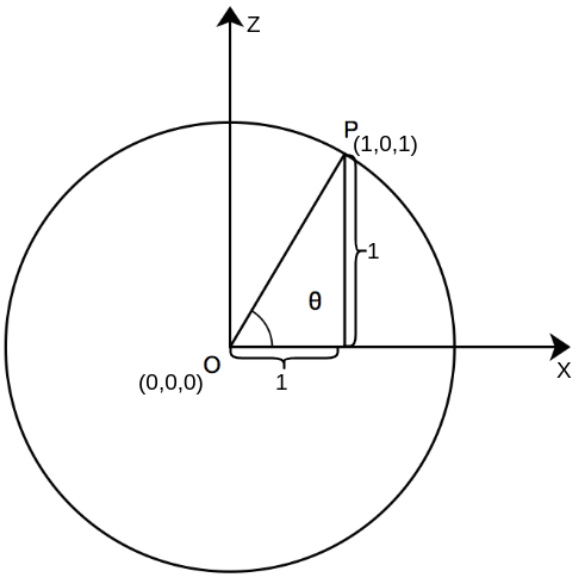


Figura 49 – Ilustração de como é obtido o ângulo theta formado pela reta OP e o eixo X.

Fonte da ilustração: Autoria própria.

para graus seria aproximadamente 45º. Nesse sentido, o índice I da imagem é calculado pela fórmula 4.8, onde N é a quantidade de imagens no conjunto C.

$$I = \left\lceil \frac{360}{360/N} \right\rceil \tag{4.8}$$

A partir do índice I obtido, mostra-se a imagem referente a ele no conjunto ordenado C. O resultado desse processo é a imagem condizente ao ângulo formado entre o observador e o objeto. É importante ressaltar que, o conjunto C deve ser ordenado, no sentido de que, tomando um elemento do mesmo, seus valores adjacentes devem representar a foto seguinte (ou anterior) ao dado elemento.

A Figura 50 mostra um crânio humano, feito com o escâner *F2S2*, sendo desenhado por meio do processo descrito.

No próximo item, serão apresentadas as considerações sobre as etapas apresentadas nessa seção 4.1.

4.3 Considerações sobre o modelo

A *Urho3D*, apesar de não estar à altura de *game engines* comerciais tais como a *Unreal* e *Unity*, possui funcionalidades tão sofisticadas quanto. Um exemplo é a possibilidade de poder funcionar em praticamente qualquer plataforma conhecida atualmente, inclusive *web*. Em contrapartida, diferentemente das *game engines* comerciais (*Unity* e *Unreal*), a

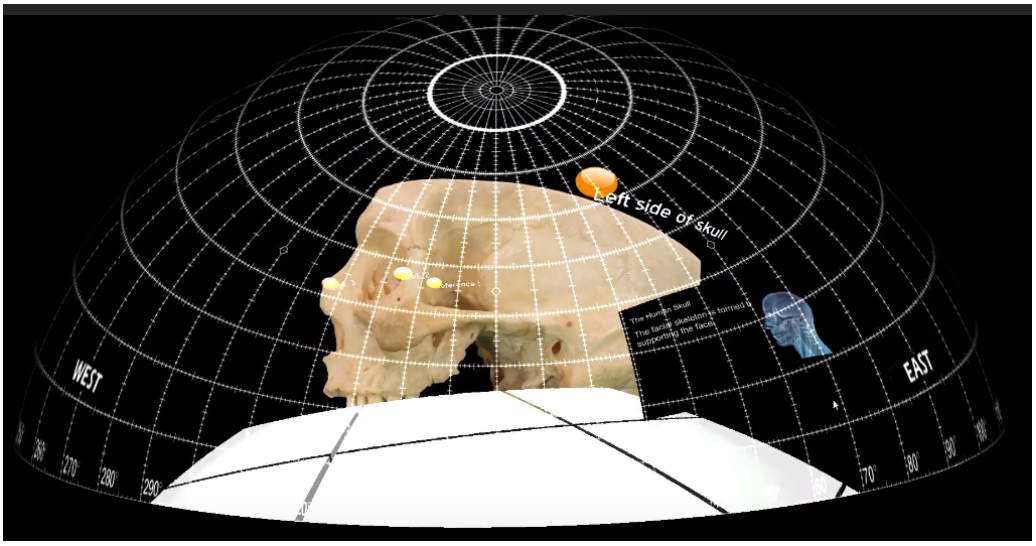


Figura 50 – Ilustração de objetos de aprendizagem, crânio humano sendo usado juntamente com *slides*.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: PUCPR

Urho3D possui código fonte aberto, possibilitando modificações em baixo nível, ou a nível de código da própria *Urho3D*.

Foi descartado o uso da *Unity*. em um primeiro momento, porque a mesma exigia licença para usar *RTT* (*Render To Texture*[27], componente que bibliotecas gráficas utilizam para renderizar a cena em uma textura que pode ser reutilizada posteriormente em outra ou na mesma cena virtual). Esse recurso é necessário para implementar o processo de geração da imagem *fulldome* (mostrado na subseção 4.1.3).

A *Unreal*, assim como demais opções não mencionadas, foram descartadas por não possibilitarem o manuseio a nível de código fonte. Eventuais modificações mais elaboradas não seriam possíveis. Para mitigar esses problemas, decidimos utilizar a tecnologia de código fonte aberto. Consideramos que quanto maior o grau de flexibilidade para modificar o *software* melhor, visto que isso possibilita adaptar a ferramenta para que a mesma possa executar tarefas complexas e específicas. *Unity* e *Unreal* não foram projetadas pensando ser diretamente conectadas com outras aplicações a nível de C++ por desenvolvedores de conteúdo. É possível fazer isso, porém, demanda ajustes inconvenientes por parte do desenvolvedor. O código fica complexo e difícil de manter posteriormente. O foco destas *engines* está mais voltado para artistas do que para programadores. Tanto a *Unreal* quanto a *Unity* poderiam ser usadas como componentes base da nossa ferramenta, mas o foco destas está em fazer jogos eletrônicos. Elas não foram projetadas pensando em ser reusadas por outros *frameworks* peculiares.

A *Urho3D* por outro lado é escrita em C++ e é projetada de modo a possibilitar a

[27] <http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-14-render-to-texture/> visto em 30/01/2019

programação tanto por linguagens de script como *AngelScript*[28] e *Lua*[29] quanto a própria *API* em C++. Isso facilita a integração da *Urho3D* (a nível de código nativo da própria *Urho3D*) com outras bibliotecas em C++ tais como *Chromium Embedded Framework (CEF)*[30] e *FLTK*[31]/*Qt*[32] (interface gráfica).

Foi considerado o uso da biblioteca gráfica *Irrlicht*[33]. Esta por sua vez não seria a escolha ideal, devido as tecnologias antigas da qual ela faz uso. A partir de testes feitos com a *Irrlicht*, executando os exemplos técnicos e examinando o código fonte, observamos que ela usa métodos ineficientes para renderização de cenas complexas e não possui técnicas de sombreamento e iluminação tão boas quanto aos da *Urho3D*. A tecnologia da *Irrlicht* é, em sua maioria, remanescente de técnicas utilizadas em *game engines* de *1997*[34].

Adicionalmente, para comunicação em rede, entre os aparelhos celulares dos usuários e o sistema, decidiu-se utilizar uma tecnologia moderna e estável, tal como o *framework Gin*[35], tecnologia utilizada em *Go-Lang* para desenvolvimento de *API*s de serviço *web*. Go também foi usado para escrever um código posteriormente transpilado para C/C++, para integrar as duas tecnologias, especificamente em funcionalidades de comunicação em rede.

Com relação a interface gráfica, julga-se ter encontrado uma solução satisfatória. O fato é que, ambientes imersivos, principalmente *fulldome* e caves, são plataformas de visualização omnidirecional. Nesse sentido é inconveniente uso de dispositivos convencionais com os utilizados em um computador de mesa para criar uma aplicação interativa para esses ambientes. A princípio, pensou-se em basear toda a interação com a aplicação por meio de um apontador 3D, similar aos controles de *smartTV* atuais. Apesar de funcional, uma interface baseada apenas nesse tipo de controle não seria ideal para aplicações fora do contexto de navegação de interface gráfica, ou interação de múltiplos usuários com sistema, ou aplicações mais elaboradas. Além disso, existe a impressão do controle ser algo desajeitado, pois se baseia em navegação espacial, algo não muito comum de ser utilizado.

Sempre se teve a ideia de que integrar o *SmartPhone* com o sistema seria a opção ideal para interface do usuário com o mesmo. Uma razão seria de que, atualmente, é comum as pessoas terem um *SmartPhone* com interface *touchscreen* e estarem habituadas a usarem tal aparelho. Não se julgava, no entanto, desenvolver aplicações nativas para sistemas operacionais *mobile* específicos, visto que isso demandaria muito tempo e aumentaria a complexidade do trabalho como um todo. Optou-se pelo uso de páginas *web* como interface gráfica, visto que, é normal que *SmartPhone*s vendidos atualmente possuam navegadores

[28] <https://www.angelcode.com/angelscript/> visto em 30/01/2019
[29] <https://www.lua.org/> visto em 30/01/2019
[30] <https://bitbucket.org/chromiumembedded/cef> visto em 10/02/2020
[31] <https://www.fltk.org/> visto em 30/01/2019
[32] <https://www.qt.io/> visto em 30/01/2019
[33] <http://irrlicht.sourceforge.net/> visto em 30/01/2019
[34] <http://irrlicht.sourceforge.net/?page_id=45> visto em 03/03/2020
[35] <https://github.com/gin-gonic/gin> visto em 30/01/2019

web, sempre atualizados com as últimas versões, possuindo, no mínimo, funcionalidades encontradas na primeira versão do *HTML5*.

Foi criado um serviço *web* com uma *API REST* juntamente de uma interface *frontend* em *HTML* básico (sem *framework* de *frontend*), apropriado para dispositivos *mobile*. A solução atendeu bem as expectativas, pois a interface *HTML* é flexível e existem diversos componentes de interface gráfica para tal tecnologia. Versões mais recentes do *HTML* e *JavaScript* possuem suporte para eventos de toque(*touchscreen*), tornando aplicações em *HTML* tão intuitivas quanto as nativas. Além disso, é possível redesenhar toda a interface gráfica *HTML*, sem alterar o núcleo do sistema, pois o serviço de interface gráfica é separado do *backend*. Isso possibilita preservar a *API*. Outro diferencial de se usar uma *API REST web* é que podem ser feitos outros clientes, para *desktops*, ou até mesmo aplicativos nativos para *Android* ou *iOS*, utilizando a mesma *API*.

Adicionalmente, as versões mais recentes dos navegadores trazem funcionalidades que antes eram apenas disponíveis para aplicações nativas, como acesso aos sensores do dispositivo: acelerômetro, giroscópio, magnetômetro, câmera entre outros. Essa é até uma razão pela qual não se desenvolvia aplicativos *mobile* como um serviço *web*, pois não se tinha acesso a, por exemplo, câmera ou GPS. Existem vantagens em se desenvolver uma aplicação *mobile* nativa. A principal seria maior velocidade de resposta da aplicação. A vantagem de uma aplicação *mobile web*, também conhecida como *PWA* (*Progressive Web App*), é que, não exige ser instalada no dispositivo do usuário. Visto que se tem acesso aos sensores do *smartphone* pelas *APIs* dos navegadores atuais, é possível implementar uma interface *web*, que funcionasse de maneira similar ao apontador 3D, porém, usando apenas o *smartphone*.

A respeito do navegador *CEF*, integrado no sistema, como mostra o item 4.1.6, o mesmo poderia ser atualizado para a versão mais recente, de modo a ter acesso aos últimos recursos tais como comunicação *WebRTC* e acesso a dispositivos de captura de vídeo (câmeras conectadas no computador). Dessa forma seria possível conectar placas de capturas via *USB*, essas por sua vez, conectadas em dispositivos de *cast* tais como *Chromecast* e *AppleTV* possibilitando o espelhamento ou reprodução de conteúdo por meio dos protocolos de *cast* da Google, Apple e similares diretamente para o sistema proposto. Efetivamente, estar-se-ia adaptando o *ChromeCast*, *AppleTV* ou similares, com o sistema. Seria possível, fazer *casting* de conteúdo, de aparelhos compatíveis, diretamente para o ambiente imersivo por meio destes dispositivos de *cast*. Adicionalmente, seria possível espelhar as telas dos dispositivos no ambiente imersivo via *WiFi*, sem necessidade de cabos anexados. Também seria possível abrir aplicações de *desktop* convencionais, tais como *Skype*, *Google Meet* e apresentações *slides* ou arquivos PDF e mostrá-los diretamente no ambiente imersivo. Tomando como referência a solução de mapeamento utilizada no item 4.1.6, seria possível utilizar da mesma abordagem para mapear aplicações VR, diretamente

do dispositivo *SmartPhone* do usuário que faz o espelhamento por *casting*. O fato é que, existem aplicações para *smarphones* que divide a tela em 2, podendo haver ou não estereoscopia. O usuário então, anexa esse celular em um suporte com conjunto de lentes e o veste na cabeça, fazendo-o assim um óculos de realidade virtual improvisado. A ideia seria espelhar a tela do celular para o sistema e mapear a imagem da mesma, usando abordagem similar ao descrito no item 4.1.6. Seria então adaptado o conteúdo de aplicações VR de *smartphone* para serem reproduzidas diretamente no ambiente imersivo.

4.4 Resultados experimentais

Os experimentos até o momento ocorreram majoritariamente, em duas configurações de *hardware*. Inicialmente o *hardware* utilizado, coincidentemente o menos potente, foi um *notebook Dell Vostro 14-5480* equipado, em um primeiro momento com 8 *gigabytes* e em outros testes com 16 *gigabytes* de memória *RAM*. O computador de teste possui *Chip* gráfico *NVIDIA GeForce 830M* que tem desempenho bruto de 554 *GigaFLOPS* e processador *Intel(R) Core(TM) i7-5500U CPU 2.40GHz* com sistema operacional *Linux Mint 19.1 Tessa*. A segunda máquina possui especificações similares, porém com uma placa de vídeo *Nvidia Quadro* de 4 saídas de vídeo *DVI*, executando sobre o sistema operacional *Windows 10*.

A *Figura 51*[36] mostra um dos exemplos implementado com o sistema, de visualização de objetos de aprendizagem virtuais. O objeto de aprendizagem em questão é um crânio humano. O objeto é formado por um conjunto de 400 imagens de resolução *UHD*. A captura de tela mostrada na Figura 51 foi feita durante testes em uma máquina com 16 *gigabytes* de memória *RAM*. É possível ver no centro da figura, um gerenciador de recursos que mostra o uso da memória em quase 100%. O conjunto das 400 imagens tem tamanho em disco de 478 *megabytes* e ao carregar esses dados na ferramenta, o tamanho em memória *RAM* não é o mesmo do tamanho em disco. Nesse caso para cada imagem de 1.19 *megabytes*, a ferramenta usa aproximadamente 25 *megabytes* em memória *RAM*. Apesar da alta demanda por memória *RAM*, a aplicação roda com média de 60 *FPS*.

A *Figura 52*[37] mostra uma captura de tela do sistema executando outro exemplo de aplicação feito com o mesmo. A ferramenta está mostrando o modelo do espécime feminino do *Visible Human Project*, simplificado até a região abdominal para economizar recursos.

Vale ressaltar que ambas as Figuras 51 e 52, mostram a versão do sistema para arquitetura distribuída, a qual foi descartada em favor de uma arquitetura centralizada como descrito em 4.1.1. Nessa infraestrutura, haveria uma instância mestre, que ditaria a cena para as instâncias gráficas, que apenas desenhariam as cenas para serem projetadas.

[36] Autoria própria: <https://www.youtube.com/watch?v=ZJTMn4nTwao> visto em 09/01/2019
[37] Autoria própria: <https://www.youtube.com/watch?v=JmlqDRLmoUM> visto em 09/01/2019

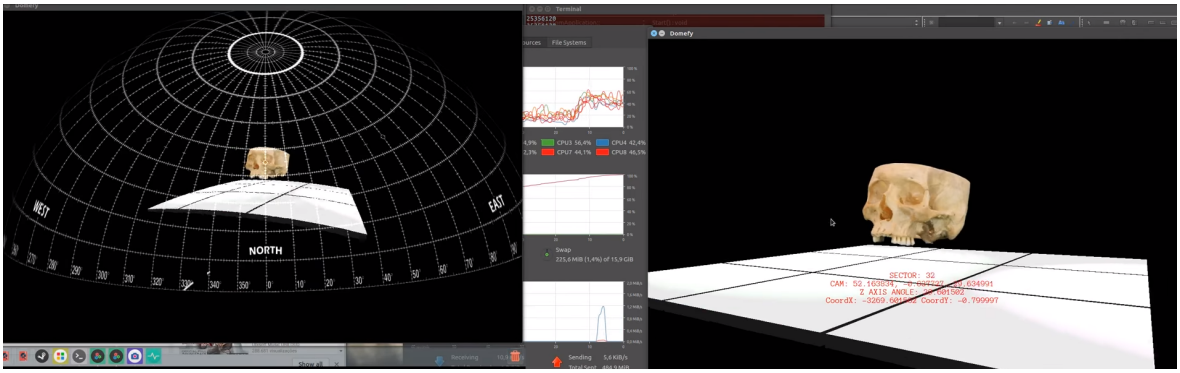


Figura 51 – Objeto de aprendizagem sendo visualizado em fulldome na ferramenta proposta.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: PUCPR

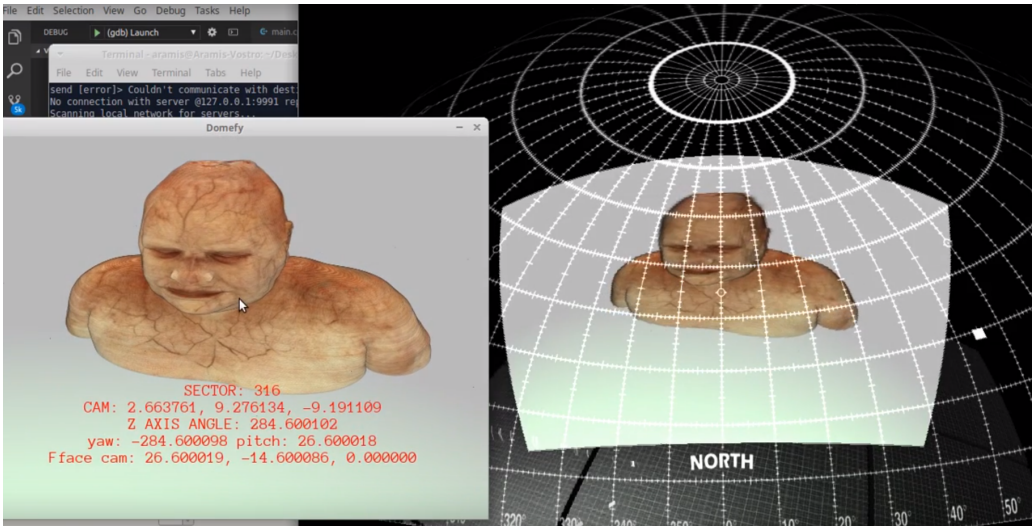


Figura 52 – Imagens de cortes do projeto do *Visible Human Project* sendo visualizadas em fulldome na ferramenta proposta.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: *National Library of Medicine* <https://www.nlm.nih.gov/>

A versão final proposta do sistema, como visto no item 4.1.2, sugere um sistema não distribuído. Apesar disso, o sistema provou ser capaz, em versões anteriores, de trabalhar múltiplas instâncias de forma distribuída.

A *Figura 53*[38] refere-se ao exemplo de aplicação descrito na 4.2.2, onde mostra o objeto de aprendizagem (crânio) junto de elementos auxiliares para ajudar na compreensão do estudo. É possível observar, sobreposto ao objeto de estudo, *hyperlinks*, representados por marcadores em amarelo, com textos e na parte inferior direita da figura está o *slide* para apresentação. Todos estes elementos foram feitos por meio de *scripts AngelScript*.

Nas figuras 54 e 55 percebemos artefatos na textura dos cortes, que se formam

[38] Autoria própria: <https://www.youtube.com/watch?v=s-ZRLc1CRw8> visto em 09/01/2019

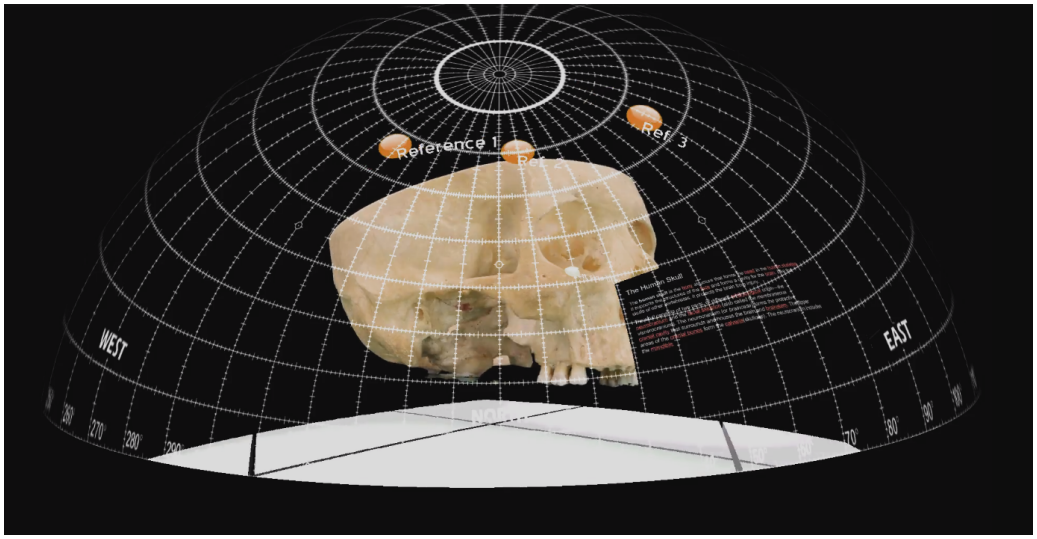


Figura 53 – Objeto de aprendizagem sendo visualizado em *fulldome* junto de *slides*.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: PUCPR

em decorrência da deficiência da *game engine* em filtrar a textura. Para evitar esse erro, limitou-se o ângulo de inclinação entre a câmera e os cortes de modo a evitar que esse problema ocorra.

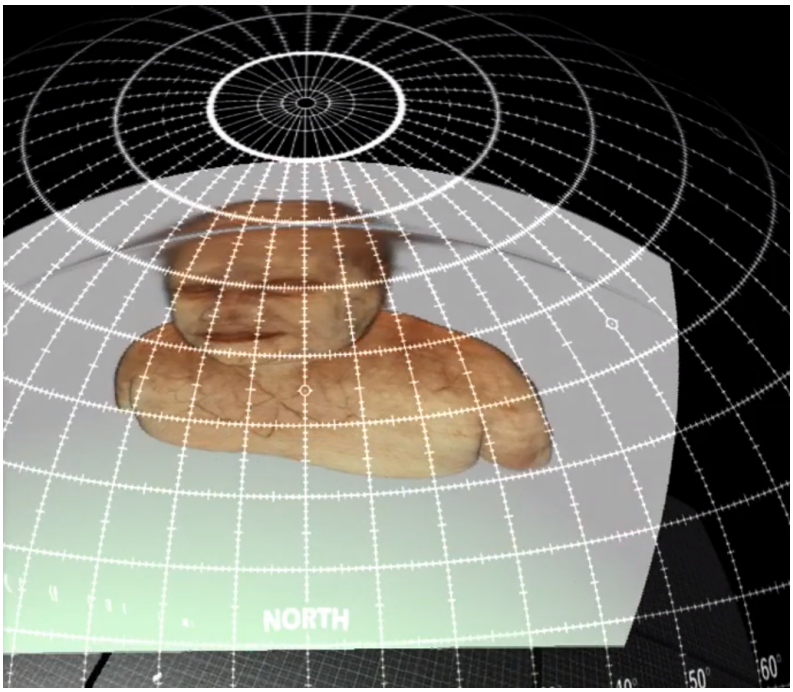


Figura 54 – Captura de tela da aplicação em plataforma *fulldome* em que mostra aberração na textura dos cortes causada por insuficiência de filtragem *anti aliasing*.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: *National Library of Medicine* <https://www.nlm.nih.gov/>

As Figuras 56 e 57 isolam outro tipo de anomalia não vista nas Figuras 54 e 55 . A renderização se distorce, de maneira similar entre as *engines*. A anomalia é conhecida

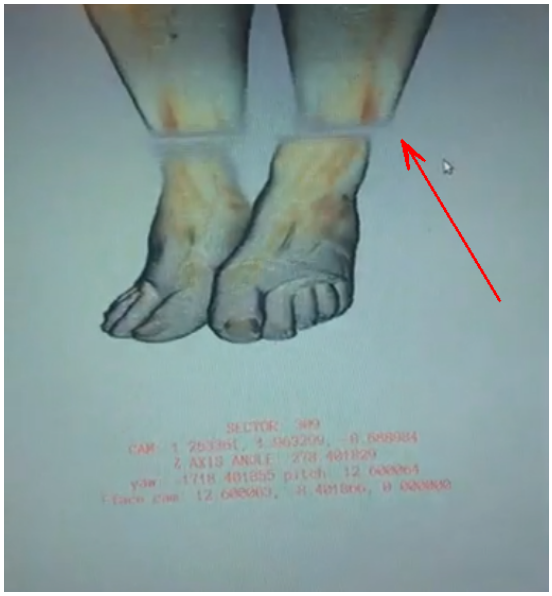


Figura 55 – Captura de tela da aplicação mostrando a aberração na textura dos cortes causados por falta de filtragem *anti aliasing*.

Fonte da ilustração: Autoria própria. Base de imagens anatómicas: *National Library of Medicine* <https://www.nlm.nih.gov/>

Figura 56 – Anomalia de renderização na *Urho3D* causada pelo "empilhamento" de geometrias planas.

Fonte da ilustração: Autoria própria.

Figura 57 – Anomalia de renderização na *Irrlicht* causada pelo "empilhamento" de geometrias planas.

Fonte: Autoria própria.

como efeito *Moirè*, que é o surgimento de interferência entre padrões. No caso da nossa aplicação, o padrão que se forma são os planos de cortes, que quando vistas em ângulo quase paralelo ao observador, formam o padrão similar a traços ou linhas. A *engine* pode mesclar pixels quando próximos uns dos outros, similar ao nosso caso. É isso que está

gerando a interferência. Outra anomalia é o desaparecimento dos planos. Por se tratar de representações planas, sem espessura, quando observados em ângulos quase que paralelos à direção de observação da câmera virtual faz com que a *engine* não renderize nada.

Alguns testes puderam ser feitos no ambiente *fulldome* próximo ao término desse trabalho. Foram feitos testes do calibrador manual, da funcionalidade para carregar instâncias via *script*, da apresentação de slides, com o sistema de troca de slides integrado e por fim, foi testado o navegador *web*.

O calibrador manual foi utilizado para fazer um mapeamento rápido, apenas para juntar as imagens dos projetores. Não foi gasto mais que duas horas para calibrar com perfeição todas as juntas entre as projeções. O calibrador semiautomático não estava disponível para teste até então. Mesmo assim, foi possível obter uma projeção final aceitável. O resultado deste trabalho pode ser visto na Figura 58, onde mostra claramente o ponto de junção entre as projeções do ambiente usado para teste.

Figura 58 – Demonstração da junção das projeções no ecrã *fulldome*.

Fonte: Autoria própria.

Após feita uma calibração básica, testaram-se as aplicações *script* do sistema. Foram convertidos 3 exemplos técnicos da *Urho3D* para executarem no *fulldome* através do *middleware*. As imagens 59, 60, 61 e 62 mostram esses exemplos técnicos da *Urho3d* convertidos para funcionar no *fulldome* por meio do *middleware*. A conversão dos exemplos é um processo simples. Apenas se encapsula o código em uma classe especial, que o sistema instancia durante a execução. Os exemplos funcionaram sem nenhum problema.

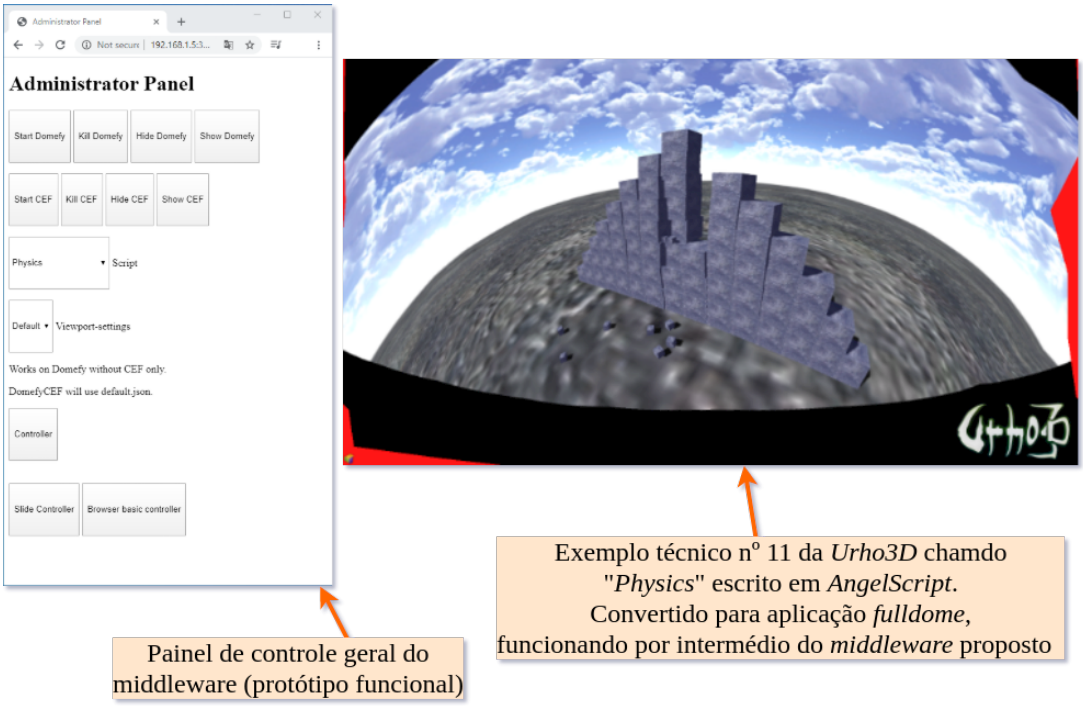


Figura 59 – Aplicação de exemplo (chamada de "*Physics*") da *Urho3D* convertida para executar no *fulldome*.

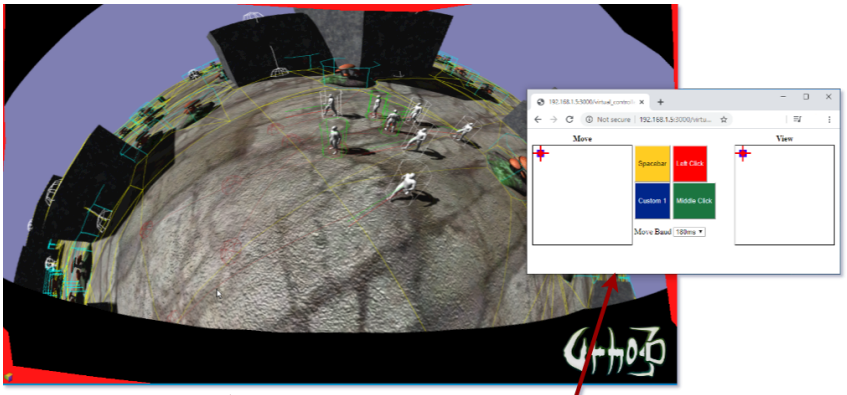
Fonte da ilustração: Autoria própria.

Fonte do exemplo interativo na *web*: <https://urho3d.github.io/samples/11_Physics.html>.

Adicionalmente, foram feitos controles em *HTML* e *JavaScript*, para que o usuário pudesse interagir com os exemplos por meio de um *smartphone* (Figura 61), ao invés de um teclado e mouse. Os comandos por sua vez, partiam do *smartphone* do usuário, eram roteados pelo *middleware*, redirecionados para a instância gráfica do mesmo e finalmente tratados na lógica escrita em *AngelScript*, segundo explicação feita no item 4.1.2.

A Figura 59 mostra um exemplo técnico da *Urho3D*, adaptado para funcionar no *fulldome* por meio do middleware. Essa aplicação foi escolhida para demonstrar a capacidade da *Urho3D* com simulação de física para jogos. Esse mesmo exemplo destacou-se, pois possibilitou a melhor experiência para percepção de profundidade, durante visualização estereoscópica. É possível ver à esquerda na Figura 59, uma janela, com um navegador web aberto. Esse navegador está acessando o painel de controle do *middleware*. Através do mesmo é possível escolher qual aplicação executar, ativar estereoscopia, e abrir os controles para interação com a aplicação.

A Figura 60 mostra outro exemplo técnico da *Urho3D*, adaptado para funcionar no *fulldome*. Foi utilizado este exemplo pois é um dos mais elaborados exemplos técnicos da *Urho3D*. Na mesma figura podemos ver à direita, uma janela de navegador, com os controles virtuais. A interação do usuário com aplicação é feita por meio desses controles. O



Exemplo técnico nº 39 da *Urho3D*, chamado de "Crowd Navigation", escrito em *AngelScript*. Convertido para platadorma *fulldome*, funcionando por intermédio do *middleware*

Interface de controle para aplicação. Uma página *web* provida pelo próprio *middleware*. O usuário pode abrir ela em um *smartphone* e interagir com a aplicação.

Figura 60 – Aplicação de exemplo (chamada de "*Crowd Navigation*") da *Urho3D* convertida para executar no *fulldome*.

Fonte da ilustração: Autoria própria.

Fonte do exemplo interativo na *web*: <https://urho3d.github.io/samples/39_CrowdNavigation.html>.

mesmo foi projetado para ser utilizado em uma tela *touchscreen*, como mostra a Figura 61.

Devido ao fato destas interfaces serem implementadas em *HTML* e *JavaScript*, é possível acessar de qualquer equipamento que tenha um navegador de *internet* instalado. Na Figura 61, o usuário controla a aplicação *Crowd Navigation* com um *smartphone*, por meio do controle virtual acessível via *browser*. A Figura 62 mostra a mesma aplicação executando em ambiente *fulldome*.

A Figura 63 mostra uma apresentação sendo feita usando o *middleware*. Foi utilizado o recurso integrado do *middleware* para apresentação de *slides* (descrito na subseção 4.1.5). O próprio apresentador faz as trocas das imagens, por meio do seu *smartphone*.

A Figura 64 mostra o navegador *CEF* integrado no sistema executando um vídeo 360 diretamente do YouTube, segundo processo descrito no item 4.1.6. Em todos os testes realizados, o navegador, depois de inicializar não apresentou nenhum problema. A taxa de atualização ficou em torno de 20 quadros por segundo (*FPS*). Não é o ideal, mas é aceitável. Não foram feitas modificações posteriores a integração do *CEF* para que o mesmo apresentasse melhor desempenho, tal como, melhor resolução ou taxa de atualização. Os testes foram todos realizados com a mesma configuração (resolução 1366 por 768, 20 quadros por segundo de atualização), de modo a garantir a estabilidade.

Figura 61 – Aplicação de exemplo (chamada de "*Crowd Navigation*") da *Urho3D* sendo controlada por meio de um *smartphone*.

Fonte da ilustração: Autoria própria.

Fonte do exemplo interativo na *web*: <https://urho3d.github.io/samples/39_CrowdNavigation.html>.

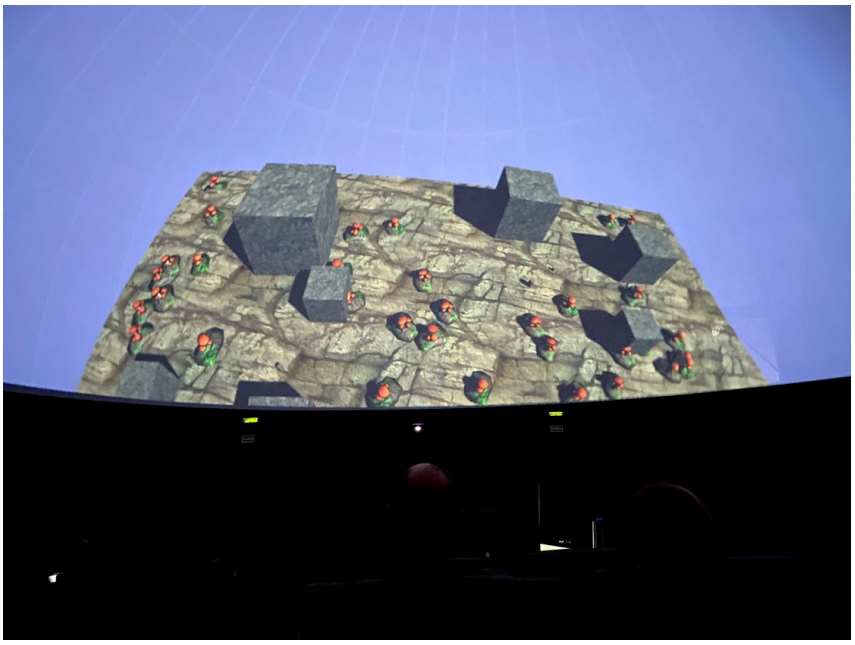


Figura 62 – Aplicação de exemplo (chamada de "*Crowd Navigation*") da *Urho3D* convertida para executar no *fulldome* por meio do *middleware*. Na imagem, o exemplo interativo convertido da *Urho3D* está executando na Arena Digital.

Fonte da ilustração: Autoria própria.

Fonte do exemplo interativo na *web*: <https://urho3d.github.io/samples/39_CrowdNavigation.html>.

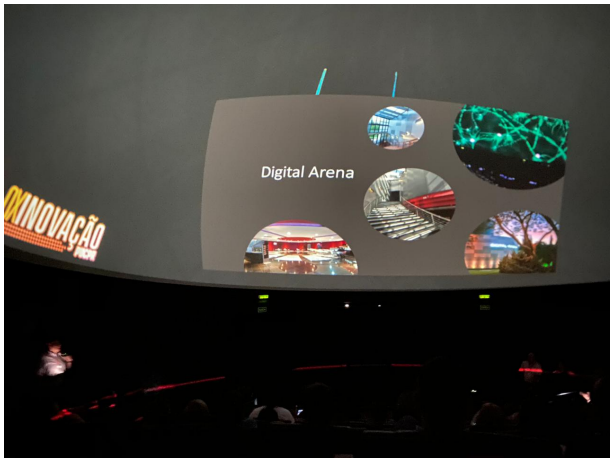


Figura 63 – Fotografia de uma apresentação *fulldome* sendo feita com o *middleware*.

Fonte: Autoria própria.

Figura 64 – Fotografia de uma apresentação em *fulldome*. No domo está sendo executado um vídeo 360º diretamente do *YouTube* por meio do *middleware*.

Fonte: Autoria própria.

5 Considerações finais

Durante o desenvolvimento desse trabalho, o *middleware* foi utilizado em apresentações reais, em eventos internos da PUCPR. O mesmo funcionou dentro da expectativa, atendendo seu propósito e nesse sentido, julgou-se que o sistema já é passível de ser utilizado, mesmo em um estado próximo a de um produto viável mínimo, seu desempenho foi aceitável.

Sobre a arquitetura da infraestrutura: os testes foram feitos na Arena Digital da PUCPR, que ainda conta com o *hardware* original antigo. Uma modificação parcial no *hardware* foi implementada de maneira temporária, para que o sistema proposto fosse testado.

A respeito da navegação pelo sistema, decidiu-se pelo uso de uma aplicação *web*, com interface feita com *HTML/JavaScript* e uso de uma *API* de *backend* executando em um *webserver*. Julga-se a *API REST* uma boa solução pelos seguintes aspectos: não fica obsoleta tão facilmente. Se eventualmente a interface gráfica com o usuário ficar desinteressante, por qualquer razão, é possível adaptar a interface de maneira mais simples, mantendo o *Core* do sistema intacto, apenas modificando o cliente *frontend*; uma interface *web* usando *HTML* provou-se versátil. Além das razões mostradas na seção 4.3, tem-se disponível muitos componentes para essa tecnologia de código aberto; não existe a necessidade de desenvolver um *hardware* novo para ter interação com a plataforma imersiva, visto que, é possível produzir uma interface gráfica compressível com um cliente *web* para *smartphones*, que são bastante acessíveis atualmente.

Inicialmente, pensou-se em utilizar uma interface convencional como teclado e mouse ou um dispositivo de controle remoto similar aos de *Smart TV* e dispositivos *apresentadores de slides*[1], porém, após experimentos feitos com *SmartPhones*, descartou-se a ideia, visto que o uso de aparelhos com tela *touchscreen* se mostrou mais intuitivo e versátil.

A reprodução de vídeo 3D 360º do YouTube, não chegou a ser testada no ambiente imersivo propriamente dito. O conceito apenas foi testado e validado em ambiente simulado. Os vídeos 360º sem estereoscopia foram testados no ambiente imersivo, como mostram os resultados na seção 4.4.

Sobre ao calibrador, em sua primeira versão, que se baseia em procedimentos manuais, notou-se que era trabalhoso de se obter uma boa calibração, tomando como base, uma calibração "do zero". Nesse sentido, desenvolveu-se o calibrador semiautomático, que serve para disponibilizar ao usuário, uma calibração inicial, não necessariamente perfeita,

[1] <https://www.logitech.com/en-us/product/wireless-presenter-r400> visto em 30/01/2019

mas que fosse utilizada como base para uma calibração mais precisa e fácil de se manter. A versão semiautomática do calibrador foi feita muito ao fim do período deste trabalho e não foi testada de maneira satisfatória, apesar de o conceito ter sido validado.

A respeito dos objetivos específicos, pontua-se que: o primeiro, sobre definir uma arquitetura para executar o *middleware* foi descrito na subseção 4.1.1. Foi proposta uma arquitetura de infraestrutura que atualiza o sistema do teatro Digital Arena. É sugerido o uso de uma arquitetura centralizada, dentre outras razões, por apresentar menor complexidade de se desenvolver aplicações para tal arquitetura. Centralizar o processamento da aplicação em apenas um computador atualmente é viável, visto que o *hardware* disponível em mercado tem potência para tal e o custo de manutenção para uma máquina é menor.

A subseção 4.1.2 refere-se ao segundo objetivo específico. Foi mostrado como os componentes de software (citados na subseção 3.1.2) foram usados para a montagem do *middleware* proposto. O *middleware* é escrito em *GoLang*, possui uma instância gráfica que basicamente é um *game engine* chamada *Urho3D*. A *game engine* foi modificada para comportar um *browser* de *internet* e adaptada para executar aplicações escritas em *AngelScript*. O usuário do sistema, interage com o mesmo via aplicações *browser* de *smartphones* ou computadores, sem haver necessidade de instalar aplicativos ou software adicional no dispositivo do usuário. Toda a comunicação entre o usuário e a aplicação gráfica é mediada pelo *middleware*.

A subseção 4.1.3 diz respeito ao terceiro objetivo específico. Foi mostrado como gerar uma imagem para projeção em ecrã *fulldome*. O procedimento mostrado faz uso de funcionalidades comumente encontradas em *game engines*. Não é feito uso de programação com *shaders* ou de operações matemática complexas. As funcionalidades utilizadas foram *render textures*[2] e *mapeamento UV*[3].

A subseção 4.1.4 fala sobre o quarto objetivo específico. Nela foi mostrado como é feita a calibração da imagem *fulldome* gerada pelo processo descrito na subseção 4.1.3. A calibração é importante para garantir a costura entre as projeções, visto que o teatro utilizado para teste (Arena Digital) usa um sistema de projeção composta (mais de um projetor). Somente gerar a imagem *fulldome* para projeção não basta, pois sempre haverá pequenas descontinuidades entre as projeções. Esses desencontros entre as projeções podem ser causados por deformações na superfície de projeção (ecrã *fulldome*), distorção das lentes dos projetores, desalinhamento dos mesmo durante longos períodos de tempo devido a vibrações na sala etc. É mostrado dois tipos de calibradores, um que se baseia por meio de ajustes manuais pelo usuário e outro automatizado, que utiliza um computador e uma câmera para auxiliar na calibração.

[2] <https://docs.unity3d.com/Manual/class-RenderTexture.html> visto em 08/03/2020
[3] <https://docs.blender.org/manual/en/latest/editors/uv/introduction.html> visto em 14/02/2020

O quinto objetivo específico, sobre interface, é tratado na subseção 4.1.5. Nela é mostrado como um sistema de janelas poderia funcionar em uma tela omnidirecional como a do *fulldome*. Foi feito na *Urho3D* um componente para implementar essas janelas. Posteriormente esse componente foi utilizado para mostrar apresentações *slides* e um navegador *web*.

A subseção 4.1.6 trata do sexto objetivo específico, sobre integração de plataformas de conteúdo de realidade virtual, principalmente vídeos, com o ambiente *fulldome*. Nesta subseção foi mostrado como se integrou um navegador *web* (*browser*) na *game engine Urho3D*. Modificou-se a *Urho3D* para integrar o *CEF*. Com essa integração foi possível abrir páginas *web* e visualizar vídeos da *internet* na *Urho3D*. Usando o componente de janelas descrito na subseção 4.1.5 foi possível projetar esse conteúdo no *fulldome*. É descrito também como adaptar vídeos 360º para visualização no *fulldome*. Adicionalmente, é possível considerar que esta subseção (4.1.6) também pode contar como sendo um exemplo de aplicação interativa do *middleware*.

Nas subseções 4.2.1 e 4.2.2 foram mostradas simples aplicações interativas, escritas em *AngelScript* e executadas pela *Urho3D* através do *middleware*. A subseção 4.2.1 trata de um exemplo mais complexo e extenso se comparado com o da 4.2.2. Ambos os exemplos se baseiam em visualização de objetos de estudo da área médica. O objetivo destas subseções foi apenas de mostrar a versatilidade do *middleware* para produção de aplicações interativas de qualquer gênero para o ambiente *fulldome*.

Para finalizar, pontuamos algumas sugestões para trabalhos futuros. Pode ser aprimorado o calibrador manual já existente, para que o mesmo fique com uma interface mais simples. Integrar no mesmo (no início do fluxo de execução), o calibrador semiautomático, tornando o processo de calibração mais simples e preciso.

A respeito do navegador: foi integrada uma versão de 2016 do *CEF*, pois a documentação disponível até o momento para a integração com a *Urho3D* era referente a versão 3.2 de 2016. Apesar disso, essa versão do *CEF* funciona bem e atende o propósito do trabalho. O mesmo pode ser atualizado para a versão mais recente e com isso, funcionalidades para comunicação via *WebRTC* poderiam ser utilizadas para abrir dispositivos de captura de vídeo (câmeras) diretamente no navegador, dentre outros. Apesar do *CEF* ser compatível com sistemas *GNU/Linux*, a integração testada foi feita apenas para sistema operacional *Windows*, em função do cronograma de trabalho.

Por fim, o item 4.3 sugere uma solução de como integrar diversas funcionalidades ao sistema, como espelhamento de área de trabalho de computadores convencionais e telas de dispositivos *smartphone*, habilitando uso de aplicativos de videoconferência e aplicativos de RV para *smartphone* serem reproduzidos no ambiente imersivo.

Referências

ACKERMAN, M. J. The visible human project/sup tm/: a resource for anatomical visualization. In: *Information Technology Applications in Biomedicine. ITAB '97. Proceedings of the IEEE Engineering in Medicine and Biology Society Region 8 International Conference*. [S.l.: s.n.], 1997. p. 29–31. Citado na página 48.

ACKERMAN, M J; YOO, Terry; JENKINS, D. From data to knowledge – the visible human project continues. *Studies in health technology and informatics*, v. 84, p. 887–90, 02 2001. Citado na página 48.

BOURKE, Paul. Spherical mirror: a new approach to hemispherical dome projection. In: . [S.l.: s.n.], 2005. p. 281–284. Citado na página 36.

BOURKE, Paul. Omni-directional stereoscopic fisheye images for immersive hemispherical dome environments. In: PRAKASH, E. (Ed.). *CGAT09 Computer Games, Multimedia and Allied Technology 09 Proceedings*. [S.l.]: Research Publishing Services, 2009. v. 1, p. 136–143. ISBN 9789810831653. Citado na página 40.

BOURKE, Paul. *Creating fisheye image sequences with Unity3D*. 2015. Disponível em: <http://paulbourke.net/dome/unityfisheye/>. Citado na página 56.

FELINTO, D.; ZANG, A. R.; VELHO, L. Production framework for full panoramic scenes with photorealistic augmented reality. In: *2012 XXXVIII Conferencia Latinoamericana En Informatica (CLEI)*. [S.l.: s.n.], 2012. p. 1–10. Citado 3 vezes nas páginas 31, 32 e 35.

GODDARD, William; MUSCAT, Alexander; MANNING, James; HOLOPAINEN, Jussi. Interactive dome experiences: designing astrosurf. In: *MindTrek*. [S.l.: s.n.], 2016. Citado 2 vezes nas páginas 37 e 40.

JUNG, C. F. *Metodologia para pesquisa & desenvolvimento: aplicada a novas tecnologias, produtos e processos*. [S.l.]: Axcel Books do Brasil Editora, 2004. ISBN 9788573232332. Citado 2 vezes nas páginas 49 e 50.

KELSO, J.; ARSENAULT, L. E.; SATTERFIELD, S. G.; KRIZ, R. D. Diverse: a framework for building extensible and reconfigurable device independent virtual environments. In: *Proceedings IEEE Virtual Reality 2002*. [S.l.: s.n.], 2002. p. 183–190. ISSN 1087-8270. Citado na página 41.

LANTZ, Ed. A survey of large-scale immersive displays. In: *Proceedings of the 2007 Workshop on Emerging Displays Technologies: Images and Beyond: The Future of Displays and Interacton*. New York, NY, USA: ACM, 2007. (EDT '07). ISBN 978-1-59593-669-1. Disponível em: <http://doi.acm.org/10.1145/1278240.1278241>. Citado 7 vezes nas páginas 30, 32, 33, 35, 36, 37 e 38.

LI, Sophia; HUANG, Yazhou; TRI, Vinh-Sang; ELVEK, Johan; WAN, Samuel; KJALLSTROM, Jan; ANDERSSON, Nils; JOHANSSON, Mats; LEJERSKAR, Dan. Interactive theater-sized dome design for edutainment and immersive training. In: *Proceedings of the 2014 Virtual Reality International Conference*. New York, NY, USA: ACM, 2014. (VRIC '14), p. 8:1–8:5. ISBN 978-1-4503-2626-1. Disponível em:

<http://doi.acm.org/10.1145/2617841.2620693>. Citado 4 vezes nas páginas 33, 36, 38 e 40.

SCHUCHARDT, Philip; BOWMAN, Doug. The benefits of immersion for spatial understanding of complex underground cave systems. In: . [S.l.: s.n.], 2007. p. 121–124. Citado na página 33.

SILVA, Flávio; OLSEN, Diogo Roberto; PIERIN, Lucas Murbach; MORAES, Aramis Hornung; JUSTINO, Edson José Rodrigues. Generation of stereoscopic interactive learning objects true to the original object. In: . [S.l.: s.n.], 2018. Citado 2 vezes nas páginas 43 e 92.

SIMON, A.; PRAGER, M. G.; SCHWARZ, S.; FRITZ, M.; JORKE, H. Interference-filter-based stereoscopic 3d lcd. *Journal of Information Display*, Taylor and Francis, v. 11, n. 1, p. 24–27, 2010. Disponível em: <https://doi.org/10.1080/15980316.2010.9652114>. Citado na página 45.

TREDINNICK, J.; RICHENS, P. A fulldome interactive visitor experience a novel approach to delivering interactive virtual heritage experiences to group audiences in fulldome projection spaces, evaluated through spatial awareness and emotional response. In: *2015 Digital Heritage*. [S.l.: s.n.], 2015. v. 1, p. 413–414. Citado 3 vezes nas páginas 33, 37 e 40.