Universidade Federal do Maranhão
Curso de Ciência da Computação

Petterson Sousa Diniz

Visualização de Modelos de Elementos Finitos em Realidade Virtual

São Luís - MA
2019

Petterson Sousa Diniz

Visualização de Modelos de Elementos Finitos em Realidade Virtual

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, **como parte dos requisitos necessários** para obtenção do grau de Bacharel em Ciência da Computação.

Universidade Federal do Maranhão

Orientador: Prof. Msc. Giovanni Lucca França da Silva

Coorientador: Prof. Dr. Anselmo Cardoso de Paiva

São Luís - MA

2019

Petterson Sousa Diniz

Visualização de Modelos de Elementos Finitos em Realidade Virtual

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, **como parte dos requisitos necessários** para obtenção do grau de Bacharel em Ciência da Computação.

Aprovada em 11 de julho de 2019

Prof. Msc. Giovanni Lucca França da Silva
Orientador

Prof. Dr. Anselmo Cardoso de Paiva
Coorientador

Prof. Dr. Tiago Bonini Borchartt
Examinador

Prof. Dr. Daniel Lima Gomes Júnior
Examinador

Prof. Dr. Alex Martins Santos
Examinador

Prof. Msc. João Otávio Bandeira Diniz
Examinador

São Luís - MA
2019

À todos que contribuíram com essa jornada.

Agradecimentos

Agradeço primeiramente e acima de tudo a Deus, Aquele que me sustenta e que me abençoa com todas as oportunidades, mesmo aquelas que não tenho tanta fé.

Agradeço a minha família, que me sustentou, me ensinou valores valiosos que levarei para toda a vida e me proporcionaram os alicerces para que eu chegasse até esse momento.

Agradeço ao meu coorientador, professor, mentor e maior incentivador, Anselmo Cardoso de Paiva, que acreditou em mim até em momentos em que eu mesmo estava desacreditado. Agradeço por todos os conselhos e a insistência, pois sem estes, eu não estaria concluindo este trabalho.

Agradeço ao meu orientador Giovanni Lucca por estar me ajudando na conclusão desse trabalho e pela amizade que temos cultivado desde quando nos conhecemos.

Agradeço aos meus professores e companheiros de curso.

Agradecimentos especiais ao professor Geraldo Braz, que ainda quando eu estava no início do curso, me deu uma oportunidade de desenvolver meu potencial. Agradeço por ter sido aquele que me deu a primeira oportunidade, que foi um divisor de águas em minha vida.

Agradeço também ao professor Aristófanes Corrêa, que de forma direta e muitas vezes ásperas, me abriu os olhos para as oportunidades que eu tinha à minha frente.

À professora Simara, coordenadora do curso, que me auxiliou nos momentos críticos da minha carreira acadêmica.

Agradeço aos professores Renato e Marco e ao Bruno, da Faculdade de Engenharia da Universidade do Porto, onde comecei a desenvolver esse trabalho. Agradeço também as minhas amigas Polyana Bezerra e Daniela Costa, companheiras de viagem e de projetos.

Agradeço aos meus amigos Nigel, Roberto e Lucas, pelos momentos que compartilhamos e coisas que aprendemos juntos no inicio da nossa jornada.

Agradeço aos meus amigos Paulo Jansen e Tiago Ribeiro por terem visto potencial em mim e me apresentado ao professor Anselmo. Agradeço também por todos os anos que trabalhamos e convivemos juntos. Ao Tiago, agradeço pelos Subways que comemos nas voltas pra casa. Ao Paulo, agradeço pelos jogos, pelas músicas, pelas conversas, pelos projetos e acima de tudo, por ter sido uma das maiores amizades que cultivei durante os meus anos de graduação.

Agradeço aos meus amigos Johnatan Carvalho e Domingos Dias, aqueles com com quem convivo maior parte do meu tempo e compartilho das minhas dificuldades e alegrias.

Agradeço pelo apoio e incentivo nos diferentes segmentos de minha vida. Espero que seja apenas o início de uma longa amizade e uma enriquecedora parceria e que nós cresçamos juntos e nos tornemos cada dia mais fortes.

Agradeço aos meus amigos Artur Pinheiro e João Victor, pelos brainstorms, game-jans, pizzas e demais momentos que passamos juntos.

Agradeço ao meu amigo João Bandeira, que em meio à todas as brincadeiras, sempre esteve torcendo para que eu me formasse um dia.

Agradecimentos ao meu amigo e então professor Ruy Oliveira, que sempre em nossas conversas, me encorajava a não desistir.

Finalmente, agradeço a todos os que contribuíram de forma direta ou indireta para este momento.

"Don't let your dreams be dreams!"
Just...
DO IT!
(Shia Labeouf)

Resumo

A Realidade Virtual tem se popularizado bastante nos últimos anos, bem como sua utilização em diversas áreas, como odontologia, biologia e engenharia. Essa tecnologia tem o objetivo de proporcionar uma experiência imersiva em um ambiente externo ao real. Dessa forma, é um terreno fértil para que simulações computacionais sejam experimentadas de uma maneira mais natural. O Método de Elementos Finitos é uma análise matemática que consiste na discretização de um meio contínuo em pequenos elementos, mantendo as mesmas propriedades do meio original. O resultado desse processo é um modelo 3D com informação temporal de tensão e deformação. Esses modelos apresentam dados obtidos do resultado da simulação de aplicação de restrições e cargas sobre uma geometria 3D em um ambiente computacional. O objetivo desse trabalho é utilizar a Realidade Virtual para visualização de modelos de elementos finitos. A aplicação proposta proporciona uma imersão do usuário em um ambiente virtual, possibilitando que este manipule e visualize o modelo 3D de forma mais intuitiva. Foram utilizadas duas formas de avaliação, uma de desempenho e outra de usabilidade. Na primeira, obteve resultados muito bons, pois obteve medidas de quadros por segundo de 106 para modelos leves e 67 para modelos pesados, excelentes para visualização do ambiente em realidade virtual. A segunda obteve resultados satisfatórios, com média de avaliação superior a 3,5 escala de 1 a 5, porém pode melhorar com base no *feedback* recebido dos usuários. Esta aplicação proporciona uma forma diferente e incrementada de análise do resultado de simulações obtidas a partir do Método dos Elementos Finitos.

Palavras-chaves: Realidade Virtual, Simulações Computacionais, Modelos de Elementos Finitos, HTC Vive, Unity 3D.

Abstract

Virtual Reality has become very popular in recent years, as well as its use in several areas, such as dentistry, biology and engineering. This technology aims to provide an immersive experience in an environment outside the real world. In this way, it is a fertile ground for computational simulations to be tried in a more natural way. The Finite Element Method is a mathematical analysis that consists of discretizing a continuous model into discrete elements, preserving the initial model's same properties. The result of this method is a 3D model with tension and deformation temporal data. These models provide data produced in a computing environment from simulation results for restricting and charging a 3D geometry. The objective of this work is to use Virtual Reality for finite-element model visualization. The proposed application provides to the user a immersive experience in a virtual environment, allowing the user to manipulate and visualize the 3D model more intuitively. Two forms of evaluation were used, one about performance and one about usability. In the first one, it obtained very good results, as it obtained measurements of frames per second of 106 for light models and 67 for heavy models, excellent for visualization of the environment in virtual reality. The second one had satisfactory results, with an average of more than 3.5 scale from 1 to 5, but it can be improved based on the feedback received from users. This application provides a different and incremental way of analysis of the result of simulations obtained from the Finite Element Method.

Keywords: Virtual Reality, Computational Simulations, Finite Elements Model, HTC Vive, Unity 3D.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

3D	Tridimensional
FEUP	Faculdade de Engenharia da Universidade do Porto
FPS	Quados por Segundo (*Frames per Second*)
HMD	*Head Mounted Display*
HTC	*High-Tech Computer Corporation*
IHC	Interface Humano-Computador
LED	Diodo Emissor de Luz (*Light Emmissor Diode*)
LCD	Tela de Cristal Líquido (*Liquid Crystal Display*)
MEF	Modelos de Elementos Finitos
RA	Realidade Aumentada
RV	Realidade Virtual

Sumário

1 Introdução

Realidade Virtual (RV) é uma interface avançada para o usuário acessar aplicações executadas no computador, proporcionando a visualização, movimentação e interação do usuário, em tempo real, em ambientes tridimensionais (3D) gerados no computador. Esta interface avançada, ao contrário da Realidade Aumentada (RA), que busca enriquecer a realidade adicionando informações, efetua a substituição de toda a realidade (KIRNER; SISCOUTTO, 2007). Nos últimos anos, a RV tem se popularizado e a tecnologia tem evoluído de tal modo que novos dispositivos proporcionam cada vez mais elevados níveis de imersão através da visão e rastreamento de movimentos (BOAS, 2013).

Essa tecnologia tem por objetivo proporcionar ao usuário uma experiência imersiva em um ambiente externo ao real, podendo ou não replicar a realidade como a conhecemos, possibilitando a criação de um ambiente que permite que o usuário participe ativamente do espaços artificiais criados por computador (MATHEW, 2014). Para isso, faz uso de sensores (dispositivos de entrada) e dispositivos de saída projetados de modo a proporcionar uma sensação de estar inserido no ambiente virtual, a partir da imersão visual, sonora e tátil (DESAI et al., 2014).

O desenvolvimento de novos, melhores e mais acessíveis dispositivos de RV, em especial os *head mounted displays* (HMDs) impulsionaram a popularização das tecnologias de RV e com isso proporcionaram um maior investimento e desenvolvimento de aplicações do tipo, desde simuladores, passando por aplicativos de reunião virtual e até jogos (PINHEIRO, 2016). Empresas como a *Sony* (SONY, 2018) e a *High-Tech Computer Corporation* (HTC) (HTC, 2018) têm contribuído bastante para o aumento no uso dessa tecnologia, possuindo um *hardware* próprio para uso de RV, a *Sony* com o *Morpheus* e a *HTC* com o *Vive*. Entretanto a RV não possui aplicações apenas na área de entretenimento, mas também tem aplicações na área de educação, turismo, *marketing*, planejamento, entre outros (GUTTENTAG, 2010).

O Método dos Elementos Finitos (MEF) é uma análise matemática que consiste na discretização de um meio contínuo em pequenos elementos descritos por equações diferenciais e resolvidos por modelos matemáticos, mantendo as mesmas propriedades do meio original (LOTTI et al., 2006). O MEF tem por objetivo determinar o estado de tensão e de deformação de um sólido de geometria arbitrária em seu estado original após sofrer ações externas (AZEVEDO, 2011). Essa técnica consiste em uma simulação feita por computador com o intuito de prever o comportamento de estruturas, que podem ser simples, como um parafuso, ou complexas, como o alicerce de um prédio, sob um conjunto de restrições, como densidade, atrito, elasticidade e ações externas à estrutura. O resultado

desse processo é um modelo 3D com informação temporal de tensão e deformação.

O uso da RV tem grande potencial em aplicações que envolvam visualização e interação com ambientes 3D, sejam essas aplicações para entretenimento, treinamento ou apenas uma experiência em um ambiente virtual que não podem ser proporcionados na vida real. Esse potencial se dá pelo fato de que a RV proporciona ao usuário uma experiencia imersiva e intuitiva, através dos dispositivos de entrada e saída especialmente desenvolvidos para essa tecnologia.

1.1 Objetivos

O objetivo geral deste trabalho é desenvolver uma aplicação em realidade virtual que permita ao usuário visualizar e interagir com o resultado de uma simulação mecânica realizada através do Método dos Elementos Finitos.

Os objetivos específicos são:

- Compreender a estrutura de arquivos usados no MEF;
- Interpretar e renderizar a geometria e as forças de um MEF usando RV;
- Incorporar a simulação a um ambiente virtual;
- Permitir que o usuário interaja com a simulação.

Este trabalho está organizado de uma forma que preza pela boa compreensão do assunto abordado. No Capítulo 2, são abordados os principais conceitos necessários para a compreensão do problema, como RV, simulações computacionais, MEF e as tecnologias utilizadas para o desenvolvimento da aplicação. No Capítulo 3, é apresentada a metodologia aplicada para o desenvolvimento da aplicação. No Capítulo 4, são apresentados os resultados e discussão. Por fim, no Capítulo 5, tem-se a conclusão do trabalho e proposta de trabalhos futuros.

2 Fundamentação Teórica

Neste capítulo serão apresentados os temas de maior relevância para o desenvolvimento deste trabalho de monografia, que visa implementar uma aplicação em RV que permita ao usuário visualizar e interagir com o resultado de uma simulação de um MEF.

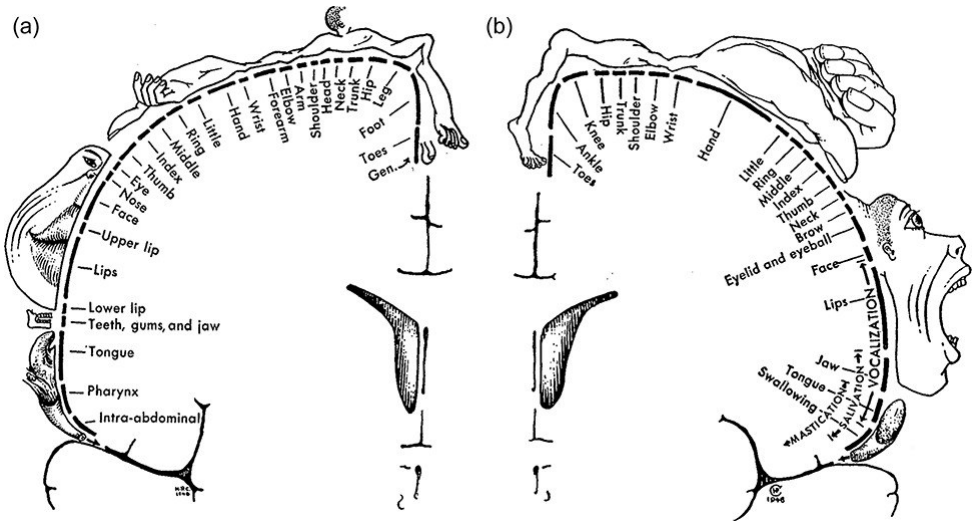
2.1 Realidade Virtual

Realidade Virtual, segundo (RIZZO et al., 2013), pode ser definida como uma interface humano-computador (IHC) que permite que o usuário se veja imerso em um ambiente virtual onde pode interagir com este de maneira mais natural do que quando usa uma IHC regular. Devido a essa característica, a RV pode ser utilizada em diversas áreas, como aplicações militares, educacionais, médicas, de telecomunicações, de entretenimento, de engenharia, entre outros (MATHEW, 2014).

Para que se alcance experiências cada vez mais imersivas, a RV utiliza diversos dispositivos de hardware, responsáveis por estimular um ou mais sentidos do corpo humano, como por exemplo, *Head Mounted Displays* (HMDs), sensores de movimento, sensores de rastreamento do corpo (RIZZO et al., 2012). Dentre as combinações possíveis, uma das mais usadas é o uso de um HMD e um sistema de rastreamento de cabeça, que permitem ao usuário visualizar e ouvir os componentes de uma cena virtual em tempo real de acordo com a sua movimentação no mundo real, de forma que o indivíduo tem a sensação de estar se movendo pela própria cena virtual (RIZZO et al., 2012).

Recentemente, dispositivos de rastreamento de mãos estão cada vez mais comuns. Empresas como *Sony* (SONY, 2018), *HTC* (HTC, 2018) e *Oculus* (OCULUS, 2019), que são empresas que possuem grande expressividade na área de RV, disponibilizaram uma alternativa a esse rastreamento junto de seu HMD, *Morpheus (Sony), Vive (HTC) e Rift (Oculus)*, que consiste em dois controles, um para cada mão, que podem ser rastreados junto com o HMD. A introdução desse rastreamento tem o potencial de aumentar bastante a imersão do usuário no mundo virtual. O trabalho de Schott (1993), em seu mapa sensorial e motor, denominado de Homúnculo de Penfield (Figura 1), mostra que em algumas partes do corpo humano há uma maior sensibilidade por parte do cérebro quando estimulados do que outras. Esse homúnculo, que é representado por um homem desproporcional, onde as partes do corpo mais sensíveis são maiores, enquanto as menos sensíveis são menores. Observa-se que as mãos tem uma grande influência na percepção do meio que estamos inserido, dessa forma, a introdução de um sistema de rastreamento de mãos tem uma grande contribuição na imersão do usuário de RV.

Figura 1 – Homúnculo sensorial (a) e motor (b).



Fonte: Snyder e Whitaker (2013).

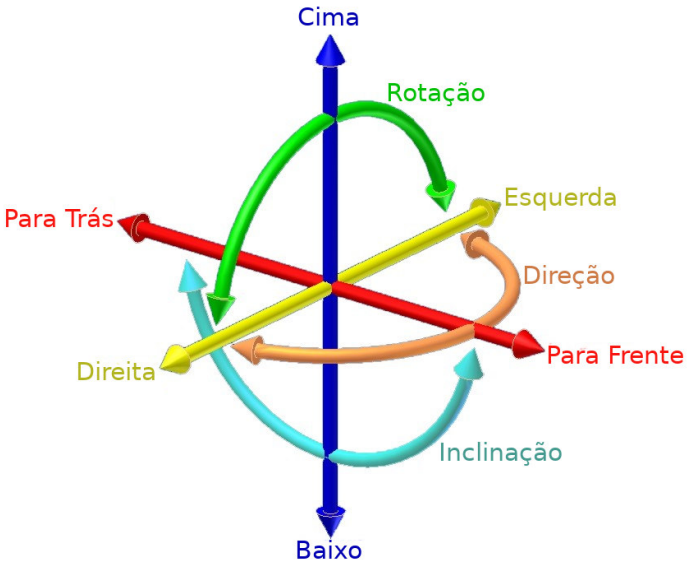
2.1.1 Rastreamento de Posição e Orientação

Em RV, muitas vezes é preciso localizar um objeto manipulado pelo usuário, alguma parte do corpo ou mesmo o corpo inteiro. Usar técnicas para acompanhar a posição e rotação de algo no mundo real a fim de utilizar no mundo virtual é chamado de rastreamento. Sensores são desenvolvidos para medir a posição espacial de um objeto móvel em relação a um objeto fixo. A posição de um referencial acoplado a um objeto pode ser perfeitamente determinada se o rastreador conseguir medir 6 parâmetros: 3 ângulos e 3 distâncias. (FUCHS, 2017).

Para que se obtenha esses parâmetros, dois tipos de rastreadores são utilizados: rastreadores mecânicos e rastreadores óticos. Os rastreadores mecânicos determinam a orientação do objeto, enquanto os rastreadores óticos determinam a posição do objeto em relação a um ponto fixo. Os rastreadores mecânicos são capazes de medir apenas um grau de liberdade, dessa forma, é necessário usá-los em conjuntos para criar um sensor com 3 ou 6 graus de liberdade (Figura 2). Os principais sensores mecânicos usados em RV são acelerômetro, inclinômetro e giroscópio, que em conjunto são capazes de rastrear a inclinação de um objeto. O rastreamento de movimento usando acelerômetro pode ser feito com uma integração dupla, porém, o erro cresce de forma quadrática, o que o torna bem impreciso (FUCHS, 2017).

Os rastreadores óticos são divididos segundo dois princípios: *outside-in* (rastreamento de fora para dentro) e *inside-out* (rastreamento de dentro para fora), representados na Figura 3. No princípio *outside-in*, os sensores são fixos e os marcadores são fixados no objeto. Nessa abordagem, as medidas de rotação são menos precisas do que as de translação. Analogamente, no princípio *inside-out*, os sensores são acoplados ao objeto enquanto os marcadores estão no ambiente, em posições fixas. Nessa abordagem, as medidas de

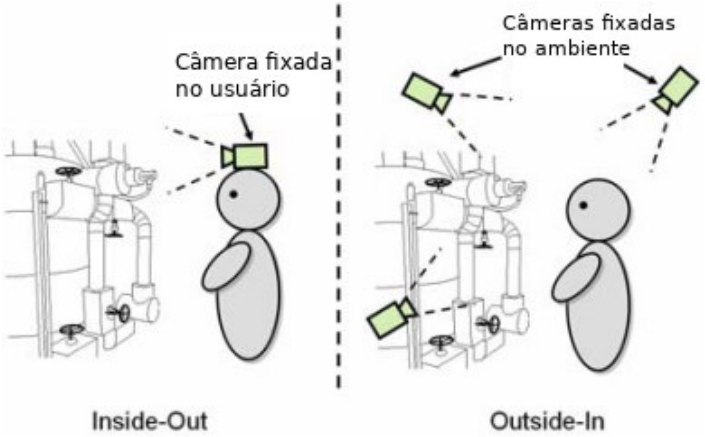
Figura 2 – Seis graus de liberdade.



Fonte: Adaptado de Wikipedia (2019).

translação são menos precisas (FUCHS, 2017).

Figura 3 – Princípios de rastreamento ótico.



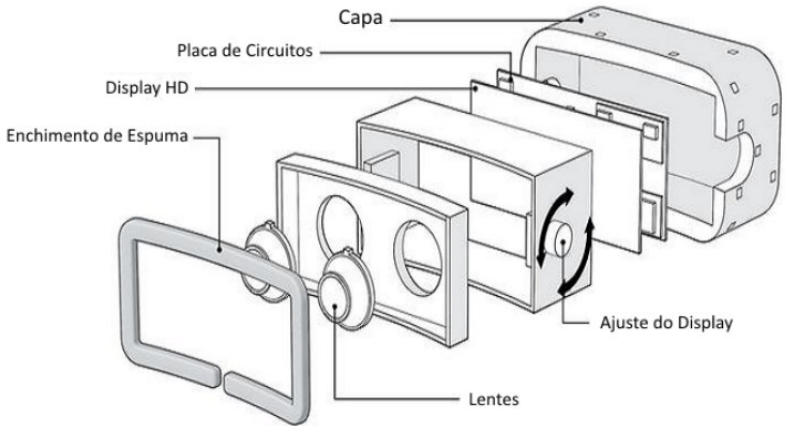
Fonte: Adaptado de Ishii (2010).

Quanto aos marcadores, podem ser ativos ou passivos. Marcadores passivos são, geralmente, superfícies pequenas cobertas de um papel retrorreflexivo e possuem a vantagem de não precisarem de fios, pois não precisam de eletricidade para funcionar, em contrapartida, necessitam de uma fonte de luz. Os marcadores ativos são geralmente feitos de diodos emissores de luz (LEDs - do inglês *light emmissor diode*) e necessitam de uma fonte de alimentação. Sua principal vantagem é que podem ser ligados um após o outro, o que torna fácil de serem identificados pelo sensor (FUCHS, 2017).

2.1.2 *Head Mounted Displays*

Atualmente, os HMDs são compostos, como mostrados na Figura 4, por uma tela de cristal líquido (LCD), um par de lentes e alguns mecanismos de rastreamento de orientação e posição da cabeça. Em alguns casos, a tela LCD e o dispositivo de rastreamento utilizados são provenientes de um *smartphone* acoplado ao HMD. As lentes têm o papel de ampliar significativamente as imagens geradas pela tela, possibilitando a imersão no ambiente (REIS, 2015). Essas lentes também são usadas para adicionar uma distorção à imagem, causando uma sensação de que há uma visão periférica (SHIBATA, 2002). A fim de que a imagem chegue aos olhos do usuário com a mínima distorção possível, uma distorção contrária é aplicada à imagem que aparece na tela (REIS, 2015). Entretanto, um conjunto de lentes acoplados a uma tela são estímulos muito limitados para uma experiência com um grau considerável de imersão. Dessa forma, os HMDs também são equipados com um rastreamento de orientação da cabeça. Esse rastreamento é feito a partir da combinação de três sensores: giroscópio, acelerômetro e magnetrômetro. Considerando os dados obtidos por esses sensores, é possível sincronizar a imagem projetada na tela com a movimentação em 360º da cabeça do usuário (REIS, 2015).

Figura 4 – Estrutura interna de um HMD.



Fonte: Pinheiro (2016).

2.2 Método dos Elementos Finitos

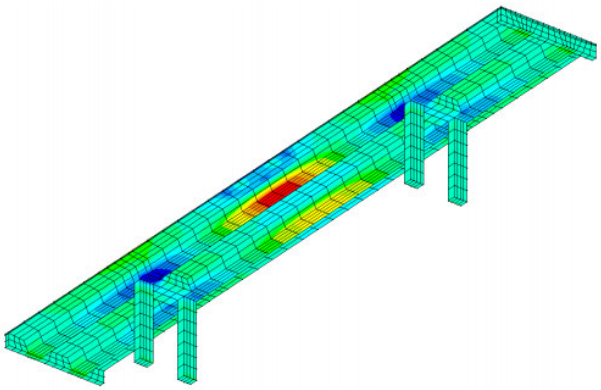
O Método de Elementos Finitos (MEF) é uma análise matemática que consiste na discretização de um meio contínuo em pequenos elementos, mantendo as mesmas propriedades do meio original (LOTTI et al., 2006). Em linhas gerais, pode se definir MEF como um método matemático no qual um meio contínuo é discretizado em elementos que mantém as propriedades de quem os originou (LOTTI et al., 2006).

Esse método se originou em meados do século XVIII com a proposta de Gauss de utilizar funções de aproximação para a solução de problemas matemáticos (OLIVEIRA,

2000). Muitos matemáticos desenvolveram técnicas e teorias analíticas para solucionar esses problemas, mas por conta das limitações de processamento de equações algébricas, essas técnicas foram pouco desenvolvidas. A aplicação prática dessa análise se tornou possível com o avanço tecnológico e computacional, que permitiu a elaboração e resolução de sistemas de equações complexas (GALLAGHER, 1973).

O MEF é uma eficiente ferramenta numérica de resolução de problemas de meio contínuo. Na engenharia, quando se trata de análise de estruturas, o uso desse método, que tem como objetivo determinar o estado de tensão e deformação das estruturas a partir de ações exteriores (Figura 5), é indispensável para possibilitar o projeto de estruturas contínuas inovadoras e arrojadas (SORIANO; LIMA, 2003; AZEVEDO, 2011). Entretanto, inúmeros trabalhos com diferentes aplicações podem ser conduzidos fazendo uso desse método, permitindo que o MEF seja utilizado em diversas áreas das ciências exatas e biológicas, devido a sua grande aplicabilidade e eficiência (LOTTI et al., 2006).

Figura 5 – Resultado da aplicação do MEF em uma estrutura.



Fonte: Azevedo (2011)

2.3 Simulação Computacional

Segundo Neto et al. (2015), simulação computacional consiste em aplicar formalizações à ambientes computacionais através de expressões matemáticas ou especificações mais ou menos formalizadas, a fim de imitar um processo ou operação no mundo real. Dessa forma, as simulações computacionais oferecem alternativas a resolução de problemas no contexto científico e escolar, sendo uma alternativa para melhorar o aprendizado de alunos, no que tange a compreensão de conceitos e o desenvolvimento de habilidades científicas (LEMOS, 2018; GRECA; SEOANE; ARRIASSECQ, 2014). Seu grande potencial na área educacional tem feito com que seja aplicada em aulas de Química para o ensino médio, como uma forma mais interativa de aprendizado (RIBEIRO; GRECA, 2003).

O uso integrado de simulações computacionais e RV deve criar melhores condições para o desenvolvimento de ambientes de simulação inovadores, bem como a melhoria dos

ambientes de RV e RA (KIRNER; KIRNER, 2008). O uso de RV e RA tem trazido novas perspectivas para a visualização de dados referentes a simulações computacionais (LEMOS, 2018).

2.3.1 Tecnologias Utilizadas

2.3.1.1 Unity

O Unity (Figura 6) é um software de desenvolvimento de jogos 2D e 3D. Estima-se que aproximadamente 50% dos jogos desenvolvidos ao redor do mundo utilizem esse software (LEMOS, 2018). É possível visualizar o resultado de uma aplicação antes mesmo de obter um versão compilada, através do modo *Play*. Este modo também permite visualizar e alterar valores de variáveis, elementos e scripts e ver o resultado de forma imediata (TECHNOLOGIES, 2018). Esse tipo de software, também conhecido como *game engine*, ou motor de jogos, são ferramentas poderosas na criação de ambientes de simulações e jogos, pois fornece aos usuários um conjunto com os recursos necessários para a criação rápida e eficiente desse tipo de programa (UNITY 3D, 2018).

Figura 6 – Ambiente de trabalho do Unity *Vive*

Fonte: Unity (2016).

O funcionamento da ferramenta é simples e se baseia em alguns conceitos: *scenes, terrains, game objects, models, textures e materials*. As *scenes e game objects* são os principais conceitos, pois sobre eles estão baseados todos os demais. As *scenes*, chamadas de cenas em português, possuem um conjunto de *game objects*, que são quaisquer objetos presentes na cena. Todos os demais objetos como câmeras, personagens, terrenos, luzes, entre outros, são classificados como *game objects*. Os comportamentos desses objetos ao

longo das cenas é definido através de *scripts*, que são escritos em linguagem *CSharp* ou *JavaScript* (LEMOS, 2018).

2.3.1.2 *Compute Shader*

Compute Shaders são programas que são executados na placa gráfica (GPU). Diferente dos *shaders* comuns, esse tipo de *shader* roda fora do *pipeline* de renderização. Esses programas podem ser usados para executar algoritmos com um elevado grau de paralelismo (UNITY, 2019). *Compute Shaders* funcionam nas seguintes plataformas:

- Windows e Windows Store, com DirectX 11 ou DirectX 12 e Shader Model 5.0
- macOS e iOS usando Metal graphics API
- Android, Linux e Windows com Vulkan API
- OpenGL 4.3 no Linux ou Windows; OpenGL ES 3.1 no Android
- Sony PS4 e Microsoft Xbox One

2.3.1.3 HTC *Vive*

Desenvolvido pela empresa HTC (HTC, 2018) , em 2017, o Vive é um hardware com enfoque para RV, com o principal objetivo de elevar o nível da mesma, a fim de trazer uma experiência inovadora e de grande qualidade. O *HMD* possui um conjunto com outros acessórios responsáveis pela sua qualidade e diferencial sobre os concorrentes da mesma categoria. Esse conjunto é composto por dois controles e dois sensores que ficam posicionados em um local fixo do ambiente físico e são responsáveis por captar os movimentos realizados pelo usuário. Esses sensores são capazes de rastrear a posição do *HMD* e dos controles que o usuário está segurando. Esse conjunto (Figura 7) de *hardwares* permite que o usuário se movimente e interaja de forma natural, com alto grau de precisão (MARTINS; BENVENUTI; ALONSO, 2017). A Tabela 1 mostra as especificações de sistema e configuração de *hardware* do *HMD*.

2.3.1.4 Abaqus

O Abaqus (Figura 8) é um software de grande versatilidade para aplicações em engenharia e consiste em vários módulos, dentre esses o módulo CAE (pré-processador), *Viewer* (pós-processador) e os módulos *Standard e Explicit* (CONCER, 2011). Nesse trabalho, foi usado o módulo *Standard*.

O pré-processador consiste em uma interface gráfica que permite ao usuário uma rápida e eficiente definição da geometria do problema. Nessa interface é possível atribuir diferentes materiais, bem como manipular suas propriedades, selecionar o número de

Figura 7 – Conjunto de RV do HTC *Vive*

Fonte: Fuchs (2017).

Processador	Intel® Core ™ i5-4590 ou AMD FX ™ 8350
Memória	4 GB de RAM
Saída de vídeo	HDMI 1.4, DisplayPort 1.2
Portas USB	1x porta USB 2.0
Sistema operacional	Windows® 7 SP1, Windows® 8.1, Windows® 10
Tela	Dupla AMOLED 3,6"diagonal
Resolução	2x 1080 x 1200 pixels
Taxa de atualização	90 Hz
Campo de visão	110 graus

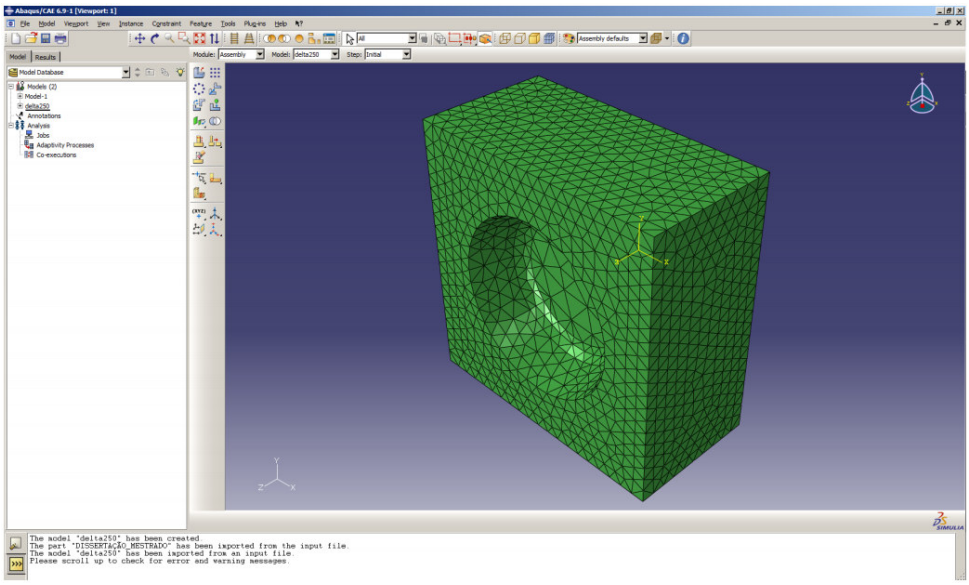
Tabela 1 – Especificações e requisitos de sistema para o HTC Vive. (VIVE, 2019)

etapas para análise e gerar a malha de elementos finitos. O módulo gera um arquivo que contém os dados de entrada do problema e que pode ser manipulado pelo usuário posteriormente (CONCER, 2011). O arquivo gerado pelo Abaqus será usado nesse trabalho, pois contém as informações da geometria do objeto estudado.

Este software utiliza o método dos elementos finitos do deslocamento. Esse método é baseado na aproximação das condições de equilíbrio de um corpo, portanto cada ponto do material utilizado é analisado. Dessa forma, o corpo é uma função do tempo e de suas coordenadas espaciais (CONCER, 2011).

O pós-processador, módulo que opera sobre os arquivos de saída, dispõe de vários métodos de visualização gráfica e de animação para interpretação dos resultados numéricos (CONCER, 2011). O arquivo gerado pelo Abaqus será usado nesse trabalho, pois contém as informações das deformações e tensões sobre o objeto estudado.

Figura 8 – Ambiente de trabalho do Abaqus.

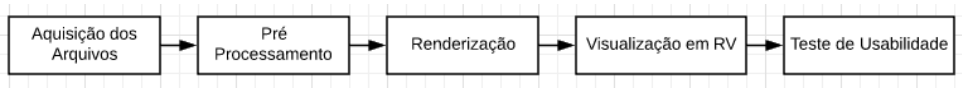


Fonte: Concer (2011).

3 Desenvolvimento da Aplicação

Nesse capítulo é apresentado o processo de desenvolvimento da aplicação em RV para visualização e interação com o resultado de uma simulação computacional gerada a partir da aplicação do MEF. Para criação da aplicação foi necessário interpretar os arquivos utilizados no processamento de um MEF. O programa utilizado para prover os resultados das simulações foi o Abaqus e os arquivos foram disponibilizados pela Faculdade de Engenharia da Universidade do Porto (FEUP). Para a execução da aplicação foi utilizado o HTC *Vive*, um HMD que possibilita a visualização imersiva de um ambiente virtual. Como ambiente de desenvolvimento e base para a aplicação foi utilizado o Unity 3D.

Figura 9 – Etapas do desenvolvimento.



Fonte: Acervo do autor

3.1 Aquivos Disponíveis

Para a execução da aplicação, é necessário ter disponível as informações da geometria do modelo e os resultados de deformação e tensão da simulação. O software escolhido, Abaqus, trabalha em dois módulos, o pré-processamento e o pós-processamento (CONCER, 2011). Esses módulos são os que mais interessam para o desenvolvimento dessa aplicação, pois os arquivos gerados por estes fornecem as informações necessárias, descritas previamente.

O módulo de pré-processamento, também chamado de CAE, permite que o usuário defina a forma e as propriedades dos materiais que serão usados posteriormente na simulação. É nesse módulo que se definem as forças que entrarão em ação durante o processo. Todas essas informações são armazenadas em um arquivo de extensão ".inp"que representa uma tarefa a ser executada. Esse arquivo é fundamental, pois ele nos proverá as informações quanto a geometria do objeto.

Após a execução da simulação, o módulo de pós-processamento, também chamado de *Viewer*, é utilizado para visualizar o resultado da simulação. Esse módulo trabalha sobre os arquivos gerados pela simulação e pode exportar relatórios em diversos formatos, desde modelos 3D ou relatórios tabulares que contém todas as informações de deformação e tensão para cada ponto da geometria, como o de extensão ".rpt"que foi escolhido para esta aplicação.

Arquivo 1 – Sessão de nós de um arquivo de entrada

```
9       *Node
10              1,          -12.5,          -12.5,           20.
11              2,          -12.5,           6.25,           20.
12              3,          -12.5,           6.25,            0.
13              4,          -12.5,          -12.5,            0.
14              5,            20.,           6.25,           20.
15              6,            20.,           6.25,            0.
16              7,            20.,          -12.5,            0.
17              8,            20.,          -12.5,           20.
18              9,    5.83096266, -0.158537224,    6.44109249
19             10,   -6.96386957,  -7.28063011,    10.8833132
20             11,   -2.28765464,   -7.7159071,     11.165122
```

3.1.1 Arquivo de Entrada

Todas as definições de dados no Abaqus são realizadas com blocos de opções e conjuntos de dados que descrevem uma parte da definição do problema. As opções são definidas por linhas no arquivo de entrada. Três tipos de linhas de entrada são usados em um arquivo de entrada Abaqus: linhas de palavras-chave, linhas de dados e linhas de comentários (SYSTèMES, 2010). Para as informações de geometria, deve-se encontrar e processar as sessões que contêm as informações dos nós e dos elementos que compõem o modelo 3D. Essas sessões são identificadas pelas palavras-chave *Node e *Element, respectivamente. Após ler uma linha do arquivo que contenha uma dessas palavras chave, todas as linhas que sucedem são linhas de dados, até que outra palavra-chave seja encontrada.

3.1.1.1 Estrutura dos Nós

Os nós do arquivo de entrada estão dispostos em uma sequência de linhas de dados. Cada linha contém informações referentes a um nó (Arquivo 1). Cada nó deve conter as seguintes informações: um identificador, do tipo inteiro, e três coordenadas (x, y, z), do tipo ponto flutuante, usando um ponto como separador decimal. Cada uma dessas informações está separada por vírgula.

3.1.1.2 Estrutura dos Elementos

Da mesma forma que os nós, os elementos estão dispostos em uma sequência de linhas de dados. Cada linha contém informações referentes a um elemento (Arquivo 2). O elemento, diferente dos nós, pode ser de vários tipos, demonstrados na Figura 10. Os tipos processados na aplicação desenvolvida são:

Arquivo 2 – Sessão de elementos de um arquivo de entrada

```
21      *Element, type=C3D4
22           1,  7,  6,  9,  5
23           2,  6,  9,  5,  2
24           3,  7,  3,  9,  6
25           4,  2,  3, 10,  1
26           5, 11,  1, 10,  4
27           6,  5,  9,  8,  2
28           7,  3, 10, 11,  2
29           8, 11,  4,  8,  1
30           9,  9,  8,  4,  7
31          10,  1, 10,  4,  3
32          11,  7,  9,  3,  4
33          12,  4, 10, 11,  3
34          13, 11,  3,  9,  4
35          14,  3,  9,  6,  2
36          15,  8,  1,  2, 11
37          16, 11,  1,  2, 10
38          17, 11,  8,  4,  9
39          18,  8,  9, 11,  2
40          19,  2,  3,  9, 11
41          20,  7,  5,  9,  822
```

C3D4: Elemento tetraédrico linear de 4 nós.

C3D6: Elemento de prisma triangular linear de 6 nós.

C3D8: Elemento isoparamétrico linear de 8 nós.

C3D10: Elemento tetraédrico quadrático de 10 nós.

C3D15: Elemento de prisma triangular quadrático de 15 nós.

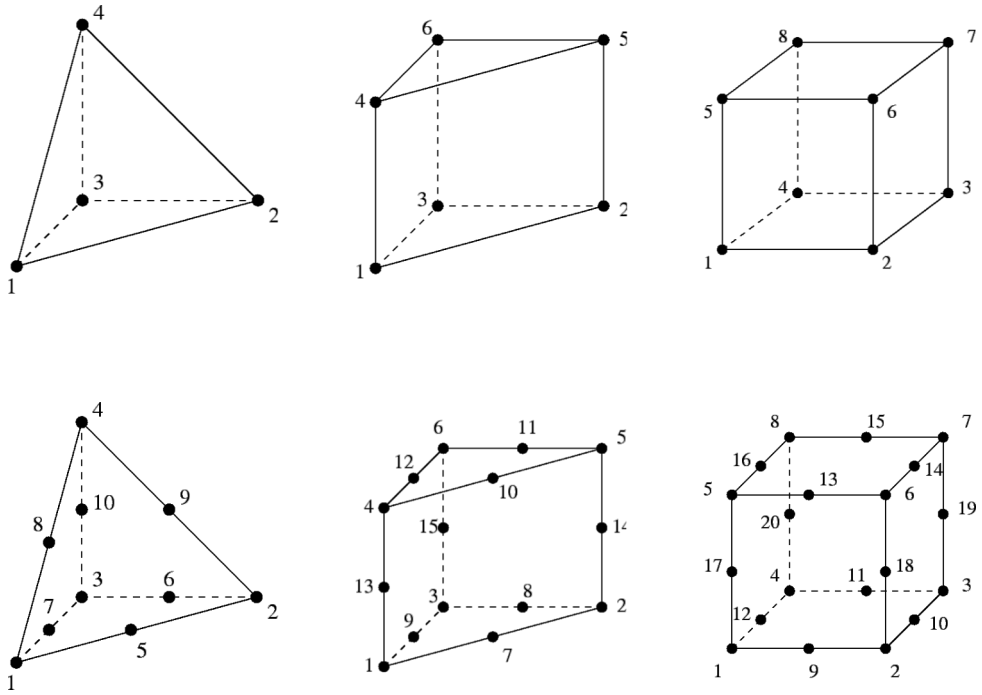
C3D20: Elemento isoparamétrico quadrático de 20 nós.

Cada linha do arquivo contém uma quantidade diferente de variáveis. A primeira variável é o indexador do elemento, as seguintes são os índices dos nós que compõem o elemento, que podem variar de 4 a 20 variáveis.

3.1.2 Arquivo de Saída

O Abaqus armazena três tipos de informações em seu banco de dados de saída: campo e histórico, que contém informações sobre o modelo, e informações de diagnóstico, que contém informações sobre a execução da simulação. As informações campo são saídas

Figura 10 – Sólidos 3D suportados pelo Abaqus: C3D4, C3D6, C3D8, C3D10, C3D15, C3D20, respectivamente.



Fonte: Adaptado de MIT (2014)

geradas para uma grande parcela do modelo com uma baixa frequência e são utilizadas para gerar gráficos no módulo de visualização. As saídas de histórico são geradas para uma pequena parte do modelo com uma alta frequência e também é usada para gerar gráficos no módulo visualizador (SYSTèMES, 2010). Para essa aplicação, foram usados as informações de saídas de campo, contendo as informações de deslocamento e estresse em cada ponto da malha. O arquivo gerado, com extensão ".rpt", contém um cabeçalho com as informações do arquivo de entrada, tempo da execução e tipo de saída. A seguir, contém uma tabela com todos os pontos e as informações citadas acima (Arquivo 3).

3.2 Renderização no Unity

O Unity é capaz de renderizar modelos 3D com o uso de 2 componentes: o *MeshRenderer*, responsável pelo processo de renderização, e o *MeshFilter*, responsável por armazenar os dados referentes à forma do objeto a ser renderizado. Desta forma, é necessário que todos os dados disponíveis sejam convertidos para os tipos de dados reconhecidos pelo Unity, para que seja possível visualizar o modelo 3D. O *MeshFilter* é um componente que contém apenas um atributo, o *mesh*. O *mesh* contém todas as informações pertinentes a um modelo 3D, suportando malhas de polígonos trianguladas ou quadranguladas (TECHNOLOGIES, 2018). Todos os elementos foram convertidos em malhas trianguladas para que pudessem

Arquivo 3 – Tabela gerada no arquivo de saída

```
Node Label    U.U1          U.U2          U.U3       S.Mises
                @Loc 1        @Loc 1        @Loc 1       @Loc 1
-----------------------------------------------------------------
   1           -10.            10.       -1.41589  67.9350E+03
   2           -10.            10.       -1.27747  66.8016E+03
   3           -10.            10.        1.37394  68.0392E+03
   4           -10.            10.   921.912E-03  65.1416E+03
   5  -6.61361E-30  -589.262E-33  -2.24459E-30  57.9261E+03
   6  -750.309E-33  -3.80400E-30   3.16285E-30  62.5264E+03
   7  -7.43873E-30   3.63316E-30   5.55511E-30  58.6168E+03
   8  -3.97709E-30   5.13429E-30  -5.78505E-30  61.2429E+03
   9      -4.11476       4.26153   137.729E-03  62.3633E+03
  10      -8.20797       8.28813  -159.229E-03  69.3974E+03
  11      -6.69115       6.86615  -123.625E-03  66.9315E+03
```

ser reconhecidos pelo Unity. A escolha de malhas quadranguladas dificultaria a conversão de tetraedros, que possuem bases trianguladas.

3.2.1 Pré-processamento dos Arquivos

Os aquivos obtidos do Abaqus, ".inp"e ".rpt", foram processados e unificados em um arquivo, a fim de que a aplicação trabalhe com apenas um arquivo e os custos computacionais de conversão ocorram apenas uma vez. O novo arquivo recebeu a extensão ".a3d", um acrônimo para animação tridimensional (Arquivo 4). O arquivo está dividido em blocos, que podem ser de três tipos: bloco de nós, bloco de triângulos e bloco de quadros. Cada arquivo possui apenas um bloco de nós e um bloco de triângulos e pelo menos um bloco de quadros, podendo conter mais blocos deste tipo. A primeira linha de cada bloco indica o tipo do bloco e a quantidade de dados contidos no bloco. Blocos de quadros contém obrigatoriamente a mesma quantidade de dados do bloco de nós.

As estruturas dos blocos varia de acordo com o tipo. No bloco de nós, as informações contidas são a posição do nó, o estresse sobre o nó e um sinalizador que informa se o ponto é interno ou externo ao objeto 3D. O bloco de triângulos contém apenas uma linha com os índices dos nós, onde a cada 3 nós um triangulo é formado. Os nós são auto indexados, de zero até uma unidade a menos que a quantidade de nós. O bloco de quadros contém as mesma informações do bloco de nós, com exceção do sinalizador.

3.2.1.1 Processo de Conversão

O processo de conversão dos aquivos disponibilizados pelo Abaqus no arquivo de animação 3D pode ser dividido nas seguintes etapas: leitura, processamento e escrita. A

Arquivo 4 – Arquivo .a3d

```
1  *NODES:11
2  -12,5 -12,5 20 0 True
3  -12,5 6,25 20 0 True
4  -12,5 6,25 0 0 True
5  -12,5 -12,5 0 0 True
6  20 6,25 20 0 True
7  20 6,25 0 0 True
8  20 -12,5 0 0 True
9  20 -12,5 20 0 True
10 5,830963 -0,1585372 6,441092 0 False
11 -6,96387 -7,28063 10,88331 0 False
12 -2,287655 -7,715907 11,16512 0 False
13 *TRIANGLE:240
14  4 5 8 8 6 4 6 5 4 5 6 8 1 8 4 4 5 1 5 8 1 8 5 4 5 2 8 8 6 5 6
        2 5 2 6 8 0 2 9 9 1 0 1 2 0 2 1 9 3 0 9 9 10 3 10 0 3 0
       10 9 1 8 7 7 4 1 4 8 1 8 4 7 1 9 10 10 2 1 2 9 1 9 2 10 0
       3 7 7 10 0 10 3 0 3 10 7 6 7 3 3 8 6 8 7 6 7 8 3 2 9 3 3 0
        2 0 9 2 9 0 3 3 8 2 2 6 3 6 8 3 8 6 2 2 9 10 10 3 2 3 9 2
        9 3 10 3 2 8 8 10 3 10 2 3 2 10 8 1 8 5 5 2 1 2 8 1 8 2 5
        10 0 1 1 7 10 7 0 10 0 7 1 9 0 1 1 10 9 10 0 9 0 10 1 8 7
        3 3 10 8 10 7 8 7 10 3 1 8 10 10 7 1 7 8 1 8 7 10 10 2 8
       8 1 10 1 2 10 2 1 8 7 4 8 8 6 7 6 4 7 4 6 8
15 *FRAME
16 -10 10 -1,41589 67935
17 -10 10 -1,27747 66801,6
18 -10 10 1,37394 68039,2
19 -10 10 0,921912 65141,6
20 -6,61361E-30 -5,89262E-31 -2,24459E-30 57926,1
21 -7,50309E-31 -3,804E-30 3,16285E-30 62526,4
22 -7,43873E-30 3,63316E-30 5,55511E-30 58616,8
23 -3,97709E-30 5,13429E-30 -5,78505E-30 61242,9
24 -4,11476 4,26153 0,137729 62363,3
25 -8,20797 8,28813 -0,159229 69397,4
26 -6,69115 6,86615 -0,123625 66931,5
```

Arquivo 5 – Estruturas usadas no processamento

```
1     Dictionary<int, Vector4> node;
2     Dictionary<int, bool> isExternal;
3     Dictionary<int, int[]> element;
4     List<Dictionary<int, Vector4>> frame;
```

leitura consiste em ler os arquivos do Abaqus e armazená-los em estruturas de dados que podem ser processadas posteriormente, o processamento é a conversão dos dados de elementos em malhas trianguladas e a escrita é o processo de escrever as novas estruturas de dados em um arquivo ".a3D".

A fim de manter a indexação dos nós e elementos, a estrutura de dados escolhida para armazenar os aquivos do Abaqus foi o *Dictionary* (dicionário, em português), que é uma estrutura de dados que permite associar uma chave, de qualquer tipo de dado, a um objeto, também de qualquer tipo. Dessa forma, o tipo da chave do dicionário é um inteiro, tanto para nós, quanto para elementos. Cada nó foi armazenado em um *Vector4*, que é um vetor de 4 dimensões (x, y, z, w). As 3 primeiras dimensões armazenam a posição e a quarta armazena o estresse. Um dicionário de boolean também foi usado, para armazenar qual ponto é externo a malha. Cada elemento foi armazenado em uma lista de inteiros. Os quadros foram armazenados em uma lista de dicionários de nós. O Arquivo 5 mostra as estruturas usadas no processo.

Afim de gerar uma malha triangulada, todos os elementos foram convertidos em elementos tetraédricos lineares de 4 nós (C3D4). Em seguida, cada um desses elementos gerou quatro triângulos. A conversão dos elementos é feita de forma bem simples. Primeiramente, todos os elementos quadráticos são tratados como lineares, dessa forma, são necessárias apenas duas conversões: C3D6 para C3D4 e C3D8 para C3D4 (Figura 11). Os elementos tetraédricos gerados são:

- C3D6 para C3D4 (3 elementos)
  - v1, v4, v5, v6
  - v1, v2, v5, v6
  - v1, v2, v3, v6
- C3D8 para C3D4 (6 elementos)
  - v1, v5, v6, v7
  - v1, v2, v6, v7

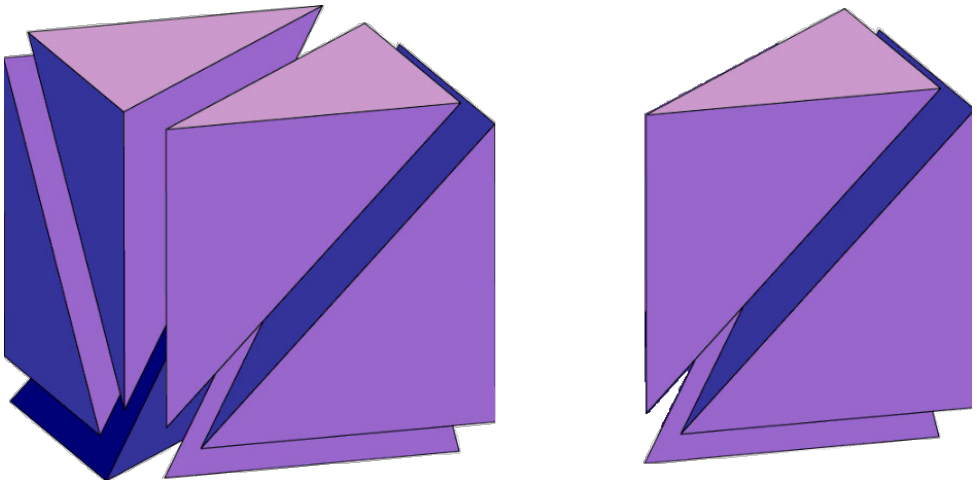
– v1, v2, v3, v7

– v1, v5, v7, v8

– v1, v4, v6, v7

– v1, v3, v4, v7

Figura 11 – Conversão de C3D8 para C3D4 e C3D6 para C3D4, respectivamente.



Fonte: Acervo do autor

Após ter carregado os nós e os elementos, é necessário converter cada tetraedro em um conjunto de quatro triângulos. Nessa etapa, é importante que as faces dos triângulos estejam direcionadas para o lado certo, para que a iluminação do ambiente funcione adequadamente. Para isso, calcula-se o plano gerado por três dos quatro pontos do elemento e utiliza-se o quarto ponto para determinar a orientação do triangulo. É possível determinar se um ponto está à frente ou atrás de um plano aplicando-se as coordenadas do ponto na equação do plano. Caso o sinal seja negativo, significa que o ponto está atrás do plano e o triângulo está orientado corretamente, caso contrário, a orientação do triângulo deve ser invertida. Esse teste é feito para todos os quatro triângulos do tetraedro. Por fim, todos os dados são salvos em um arquivo de animação 3D, que será usado na aplicação.

3.2.2 Carregando Arquivo de Animação 3D

Para a execução da animação, o componente *AnimationLoader* é responsável por carregar os dados da animação e converter para o formato que o Unity reconheça. Os dados são carregados para uma classe chamada *AnimationFile*. Essa classe armazena todos os dados do arquivo e gerencia a execução da animação. Os atributos dessa classe são mostrados no Arquivo 6.

Arquivo 6 – Classe *AnimationFile*

```
1      string name;
2      int nodeCount;
3      Vector3[] node;
4      bool[] isExternal;
5      int triCount;
6      int[] tri;
7      float[] property;
8      float minProp;
9      float maxProp;
10     int frameCount;
11     Vector3[][] nodeFrame;
12     float[][] propertyFrame;
```

Arquivo 7 – Carregando o Modelo 3D

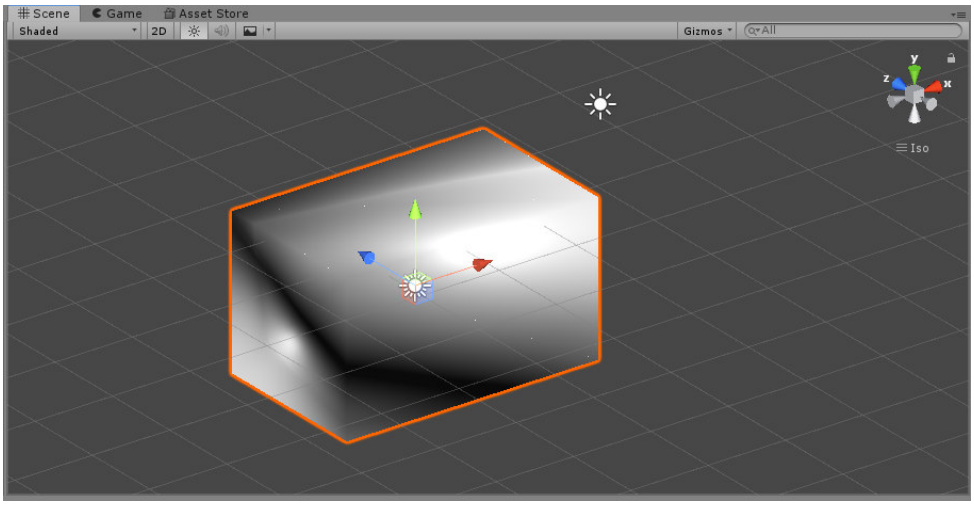
```
1      AnimationFile file = AnimationFile.Read();
2      if (file != null)
3      {
4          MeshRenderer rend = AddComponent<MeshRenderer>();
5          MeshFilter renderer = AddComponent<MeshFilter>();
6          renderer.mesh.indexFormat = IndexFormat.UInt32;
7
8          renderer.material = material;
9          renderer.mesh.vertices = file.node;
10         renderer.mesh.triangles = file.tri;
11     }
```

Após carregar os dados, o componente *AnimationLoader* instancia os componentes repensáveis pela renderização: *MeshRenderer e MeshFilter*. Por padrão, o Unity trabalha com indexação de vértices de 16 bits, então é necessário explicitar que se deseja usar indexação de 32 bits, visto que podemos ter modelos muito grandes, ultrapassando o número de vértices suportados por uma indexação de 16 bits. Após isso, o vetor de vértices e o vetor de triângulos do *MeshFilter* são atualizados com os que foram carregados do arquivo (Arquivo 7).

Ao fim desse processo, já é possível ver o modelo 3D na cena do Unity. É possível observar na Figura 12 que o modelo está com a iluminação completamente errada. Isso acontece por que os modelos gerados a partir de elementos possuem ligações entre vértices internos, o que não é comum nos modelos 3D utilizados no Unity, que usa o cálculo da normal de iluminação para cada vértice. Para solucionar esse problema, um *shader* deve ser

implementado, recalculando a normal de iluminação para cada face, ao invés de calcular para cada vértice.

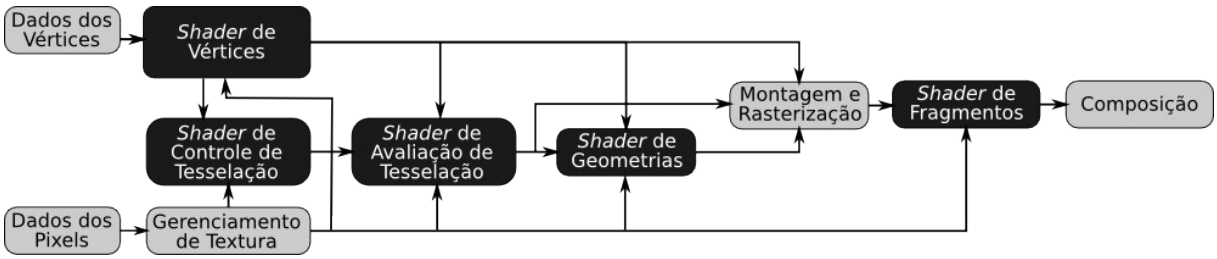
Figura 12 – Modelo carregado no Unity



Fonte: Acervo do autor

Para calcular a normal em cada face, é preciso substituir um programa no *pipeline* de renderização. Esse *pipeline* (Figura 13) é um conjunto de pequenos programas, chamados *shaders*, que executam na placa gráfica e são responsáveis por fazer a conversão de coordenadas 3D em pixels 2D para serem exibidos na tela (OPENGL, 2019). O programa que precisa ser substituído é o *shader* de geometrias, que é um programa que pode ser executado sobre qualquer primitiva (triângulo, linha e ponto), a fim de determinar a forma que será processada. O programa que substituirá o *shader* de geometrias deverá operar sobre os vértices de um triângulo, e substituir a normal do vértice pela normal do triângulo (Arquivo 8). Após a substituição do programa *shader*, o modelo 3D pode ser visualizado corretamente, como mostrado na Figura 14.

Figura 13 – *Pipeline* de renderização



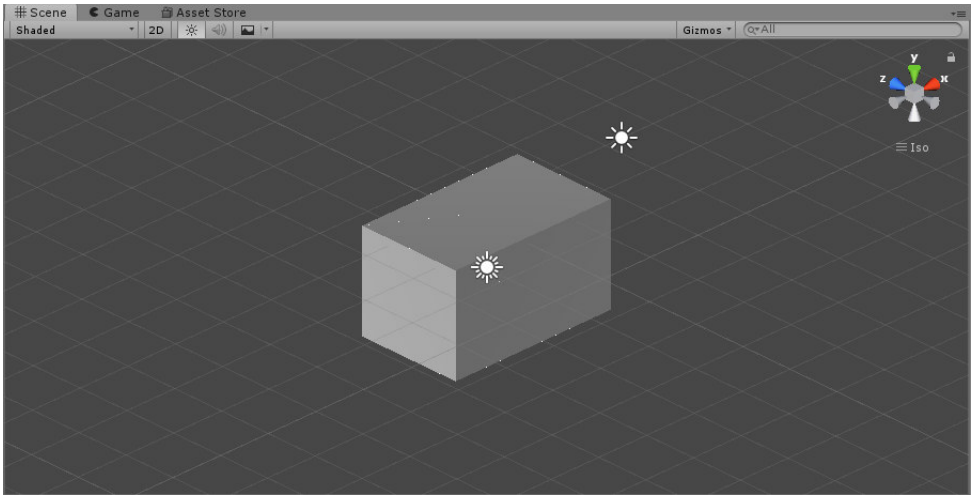
Fonte: Tavares (2014)

Após renderizar a geometria do modelo 3D, é preciso preparar os componentes necessários para animação. A animação é feita a partir de uma interpolação linear entre

Arquivo 8 – *Geometry shader*

```
1     [maxvertexcount(3)]
2     void geom (
3       triangle InterpolatorsVertex i[3],
4       inout TriangleStream<InterpolatorsVertex> stream
5     ) {
6       float3 p0 = i[0].worldPos.xyz;
7       float3 p1 = i[1].worldPos.xyz;
8       float3 p2 = i[2].worldPos.xyz;
9
10      float3 triangleNormal = normalize(cross(p1 - p0, p2 - p0
          ));
11      i[0].normal = triangleNormal;
12      i[1].normal = triangleNormal;
13      i[2].normal = triangleNormal;
14
15      stream.Append(i[0]);
16      stream.Append(i[1]);
17      stream.Append(i[2]);
18    }
```

Figura 14 – Modelo renderizado após a implementação do *geometry shader*



Fonte: Acervo do autor

dois quadros. O quadro inicial sempre é a geometria do modelo carregada a partir dos pontos, no arquivo ".a3D". O quadro seguinte contém as informações de posição e estresse em cada vértice após a simulação. Com essas informações em mãos, a aplicação deve interpolar cada uma das coordenadas e a variável de estresse para todos os vértices do modelo 3D. Esse tipo de tarefa é muito lento para ser executada em CPU e visto que é uma tarefa com alto nível de paralelismo, um *compute shader* foi implementado para executar esse procedimento mais rápido (Arquivo 9).

O componente *AnimationLoader* é responsável por fornecer os dados para o *Com-*

Arquivo 9 – *Compute shader* implementado para interpolar os vértices da animação

```
1     [numthreads(70,1,1)]
2     void DeformMesh(uint3 id : SV_DispatchThreadID)
3     {
4       resultNode[id.x] = lerp(startNode[id.x], endNode[id.x],
           t);
5
6       float r = (lerp(startProp[id.x], endProp[id.x], t) -
           minProp) / (maxProp - minProp);
7       r = min(r, 1);
8       r = max(r, 0);
9
10      float H = 0.66 - (r*0.66);
11
12      color[id.x].r = abs(H * 6 - 3) - 1;
13      color[id.x].g = 2 - abs(H * 6 - 2);
14      color[id.x].b = 2 - abs(H * 6 - 4);
15      color[id.x].a = 0.5;
16    }
```

Arquivo 10 – Inicialização de variáveis do *compute shader*

```
1     start.SetData(a.node);
2     end.SetData(a.nodeFrame[0]);
3     sProp.SetData(a.property);
4     eProp.SetData(a.propertyFrame[0]);
5
6     vertexCount = a.node.Length;
7     kernel = deform.FindKernel("DeformMesh");
8
9
10    deform.SetFloat("t", t);
11
12    deform.SetFloat("minProp", a.minProp);
13    deform.SetFloat("maxProp", a.maxProp);
```

pute shader. Os dados devem ser atualizados a cada quadro para que a animação funcione corretamente (Arquivo 10). Após a implementação do *compute shader* e a correta inicialização é possível visualizar o resultado do processamento do MEF em forma de uma animação.

3.2.3 Otimização

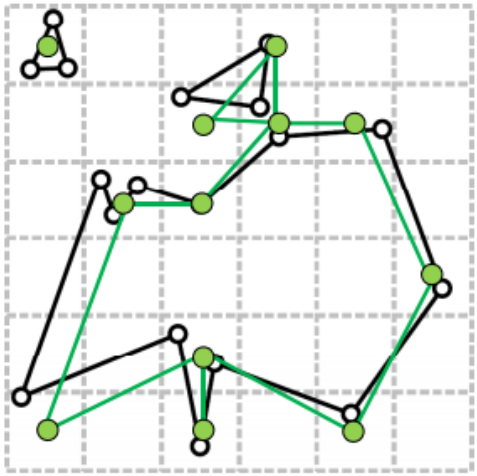
Apesar da maior parte do processamento ser executado na placa gráfica, o que torna a aplicação muito mais rápida, a animação de um modelo 3D muito grande, ou mesmo sua renderização estática, ainda é inviável. Quando um modelo tem muitos vértices, a placa gráfica deve renderizá-lo em lotes, pois tem limitações de memória e núcleos de processamento. Sendo assim, um modelo 3D com uma grande quantidade de vértices deve passar por um pré-processamento, a fim de reduzir de forma significativa a quantidade de vértices, entretanto sem perder as suas características.

Para fazer a redução do número de vértices, um algoritmo para clusterização de vértices foi utilizado. Esse algoritmo consiste em:

1. Dividir o espaço 3D em volumes menores e uniformes. Cada um desses volumes é uma região cúbica de dimensão definida pelo usuário.
2. Associar cada vértice do modelo 3D a um dos volumes gerados.
3. Para cada um dos volumes onde tem pelo menos um vértice associado, eleger um representante. O representante será o vértice que representa todos os vértices do volume e é definido pela mediana das coordenadas dos vértices associados ao volume. Todas as ligações existentes entre os vértices anteriores são direcionadas para vértice representante.

A fim de preservar a forma do modelo 3D, o algoritmo foi utilizado apenas no interior do objeto. A Figura 15 mostra uma simplificação em duas dimensões do algoritmo utilizado.

Figura 15 – Algoritmo de clusterização de vértices.



Fonte: Stanford (2010)

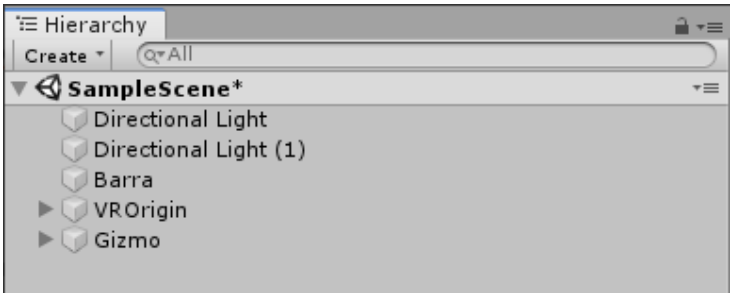
3.3 Visualização em RV

Para a visualização imersiva da animação, foi escolhido o *head mounted display* da HTC, o *Vive*. Esse HMD foi escolhido por oferecer uma experiencia imersiva de alta qualidade. Suas especificações estão detalhadas no Capítulo 2.

3.3.1 Criação da Cena

A aplicação é composta por apenas uma cena do Unity. Nela estão presentes a simulação computacional, um gizmo 3D que possui função de orientação, objetos necessários para a visualização em RV e objetos responsáveis pela iluminação. A Figura 16 mostra os objetos usados na cena. Os objetos *Directional Light* são responsáveis pela iluminação do ambiente, o objeto *Barra* é responsável por carregar e executar a animação, o objeto *VROrigin* é responsável pela visualização do ambiente em RV e o objeto *Gizmo* é responsável por mostrar a orientação da animação durante a execução da aplicação.

Figura 16 – Objetos presentes na cena.



Fonte: Acervo do autor

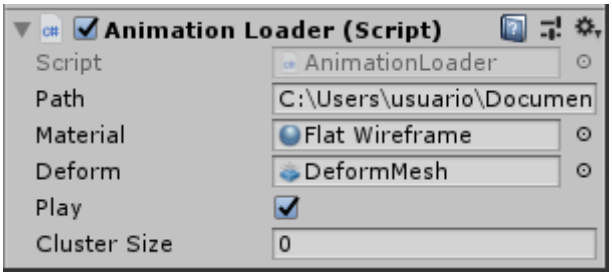
O objeto *Barra* contém os seguintes componentes: *Transform, Animation Loader, Basic Grabbable e Box Collider*. Cada um desses componentes desempenha uma função para garantir o funcionamento correto do objeto. O componente *Transform* é responsável por gerenciar posição, a orientação e a escala do objeto no mundo virtual.

O componente *Animation Loader* é responsável por gerenciar o que é necessário para a animação. É nesse objeto que as informações pertinentes ao modelo devem estar, como o nome do arquivo, o material com o *geomentry shader* modificado, o programa *compute shader* e a variável que indica o tamanho do cluster a ser utilizado no algoritmo de redução de vértices (Figura 17).

O componente *Basic Grabbable* é responsável por permitir que o usuário mova e rotacione com o modelo 3D, utilizando os controles do HTC *Vive*. Esse componente está incluso em um pacote disponibilizado gratuitamente pela HTC, chamado *ViveInputUtility*, e foi obtido por meio da loja do Unity.

Por fim, o componente *Box Collider* é requisitado pelo componente *Basic Grabbable*, e tem a função de indicar a área em que o objeto pode ser manipulado.

Figura 17 – Componente *Animation Loader*

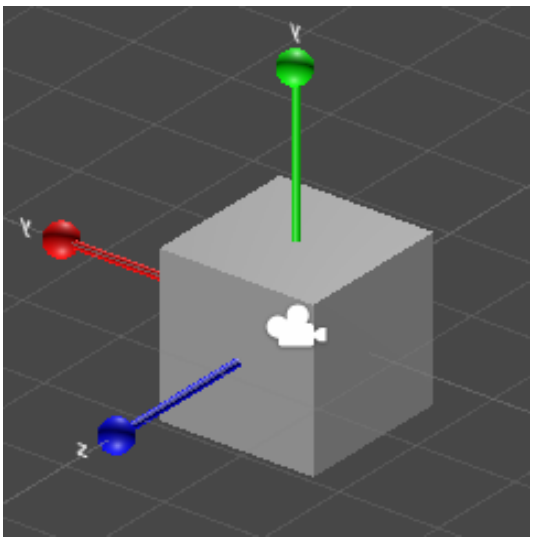


Fonte: Acervo do autor

O objeto *VROrigin*, disponibilizado pela Steam, é responsável pela visualização do ambiente em RV e pelo rastreamento de todos os *hardwares* contidos no conjunto de RV da empresa. Esse objeto está estruturado de uma forma que possibilita que os objetos sejam monitorados em um sistema de coordenadas próprio e posteriormente convertidos para o sistema de coordenadas do Unity. Esse objeto está incluso no pacote chamado *SteamVR*, que pode ser encontrado na loja do Unity.

Por fim, o objeto *Gizmo* é composto por um cubo e outros 3 retângulos afunilados com uma esfera na ponta (Figura 18). Esse componente possui função de orientação, uma vez que o mesmo acompanha as variações de rotação que ocorrem na simulação. Esse objeto é composto pelos seguintes componentes: O *Gizmo Behavior*, responsável por atualizar a orientação do objeto de acordo com a orientação do modelo 3D, e o *Follow Position*, responsável por fazer o objeto acompanhar o controle do HTC, que é manipulado pelo usuário.

Figura 18 – Objeto *Gizmo*.



Fonte: Acervo do autor

4 Resultados e Discussão

Este capítulo apresenta os resultados obtidos. A aplicação foi avaliada a partir de duas análises: desempenho e usabilidade. Para a primeira, foi analisado a velocidade de processamento e a taxa de quadros por segundo. Na segunda, foi aplicado um teste com usuários, que responderam um questionário de usabilidade após o teste.

4.1 Aplicação

A aplicação desenvolvida permite ao usuário carregar um arquivo de animação 3D e visualizar em realidade virtual. As animações estão limitadas ao funcionamento de apenas um quadro, mas as estruturas ja conseguem converter e armazenar mais quadros. As interações possíveis com a animação são:

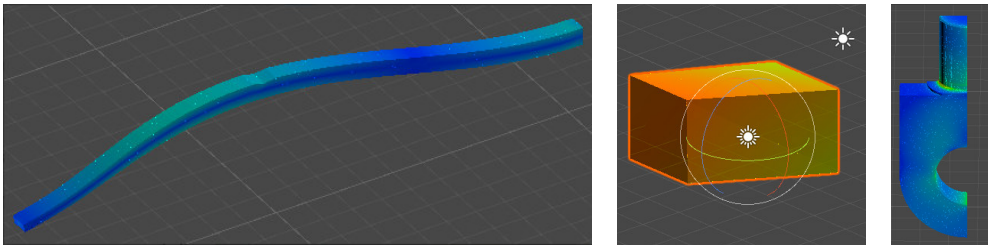
- Mover e rotacionar o modelo 3D;
- Controlar a velocidade da animação;
- Avançar ou retroceder e;
- Reproduzir e pausar.

Também é possível converter aquivos de entrada e saída do Abaqus em arquivos de animação 3D, com algumas limitações. Os modelos de entrada do Abaqus podem ser muitos complexos algumas vezes, permitindo que o usuário crie sessões na geometria e a divida em diversos arquivos. O conversor não consegue suportar esse tipo de funcionalidade.

As simulações (Figura 19) usadas para o teste foram:

- Uma simulação de um bloco retangular simples sendo esticado, composta por 11 nós e 20 elementos do tipo C3D4 e 1 quadro. Essa simulação foi criada com o intuito de testar a aplicação no início, por ser leve e simples.
- Uma simulação de barra de metal sendo torcida, composta por 1821 nós e 240 elementos do tipo C3D20 e 1 quadro. Essa simulação foi usada nos testes com o usuário e nos teste de conversão entre outros tipos de elementos e o elemento C3D4. Essa simulação não precisa de nenhuma otimização para executar.
- Uma simulação de um olhal sob tensão causada por um peso, composta por 686.940 nós e 469.541 elementos do tipo C3D10 e 1 quadro. Essa simulação foi usada para testar os limites da aplicação. Nesse ponto, foi necessário usar a otimização que reduz o número de vértices de um modelo 3D, apresentado na Sessão 3.2.3.

Figura 19 – Modelos: Barra, Retângulo e Olhal



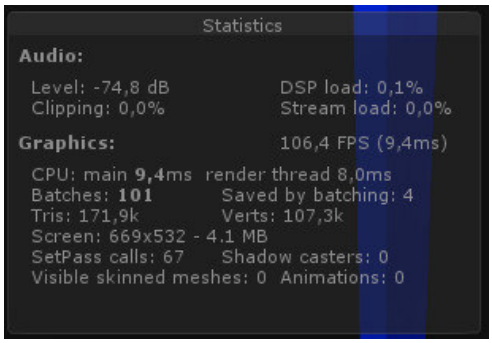
Fonte: Acervo do autor

Sistema Operacional	Microsoft Windows 10 Pro 64 bits
Processador	Core(TM) i7-9700K CPU @ 3.60GHz
Memória	16 GB DDR3
Portas USB	Intel(R) USB 3.1
Placa de Vídeo	NVIDIA GeForce RTX 2070
Memória da Placa de Vídeo	8 GB de GDDR6
Velocidade de Memória da Placa de Vídeo	14 Gbps
Saída da Placa de Vídeo	HDMI 2.0

Tabela 2 – Especificações do computador utilizado nos testes.

As configurações do computador utilizado nos testes se encontra na Tabela 2. A aplicação apresentou bom desempenho, executando a simulação da barra de ferro a 106 quadros por segundos (FPS), que é um número superior a taxa de atualização do HTC *Vive* (Figura 20).

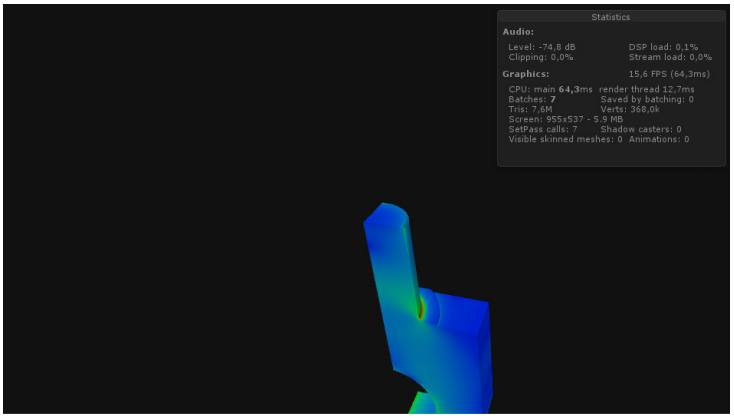
Figura 20 – Desempenho da aplicação - barra de ferro.



Fonte: Acervo do autor

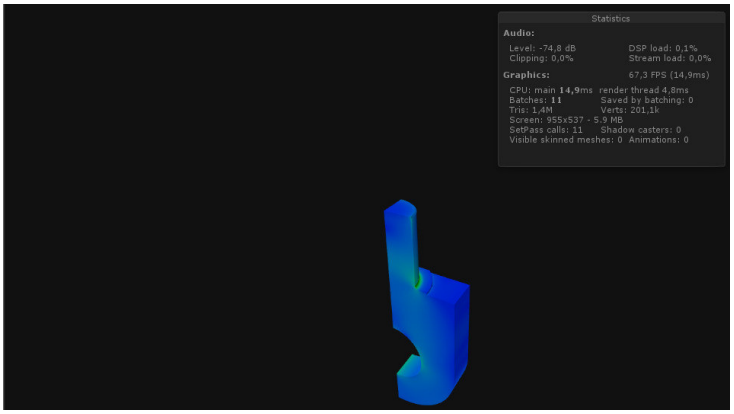
A simulação do olhal, por conter muitos nós, apresentou um desempenho bem inferior ao desejado. Ao executar a aplicação sem aplicar a otimização, foi obtido apenas 15 FPS (Figura 21). Após otimizar, a quantidade de nós foi reduzida a 39.950, o que é aproximadamente 6% da quantidade inicial. Com a otimização o desempenho melhorou, atingindo um FPS de 67, que é 74% da capacidade do HMD (Figura 22).

Figura 21 – Desempenho da aplicação - olhal não otimizado.



Fonte: Acervo do autor

Figura 22 – Desempenho da aplicação - olhal otimizado.



Fonte: Acervo do autor

4.2 Teste de Usabilidade

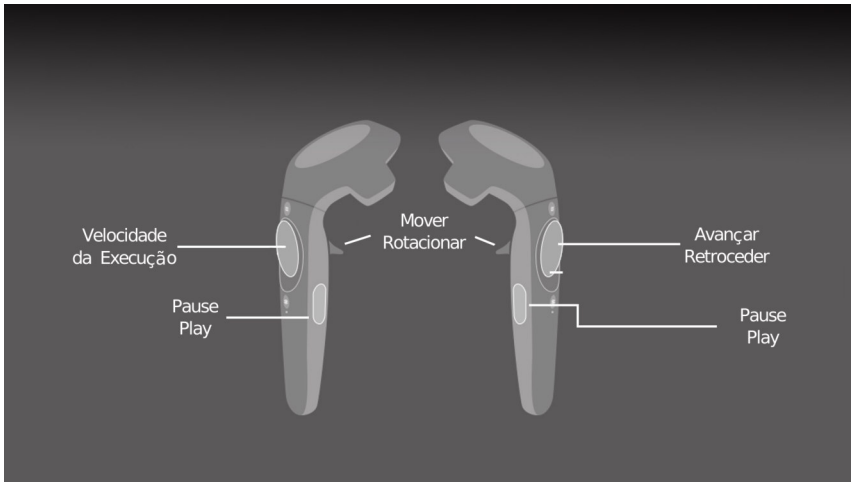
Para validar o uso da aplicação, foi utilizado um teste para avaliação da usabilidade. Os teste foi organizado da seguinte forma:

1. No primeiro momento, é apresentado o contexto do problema e o objetivo da aplicação. Em seguida, o usuário é colocado para usar a aplicação sem qualquer tipo de instrução a fim de descobrir o que é possível fazer dentro do ambiente virtual. Esse teste teve duração de um minuto para cada participante;

2. Após o primeiro teste, o usuário tem acesso à Figura 23, que mostra todas as ações que podem ser executadas dentro da aplicação, podendo tira suas dúvidas. O usuário testa a aplicação mais uma vez, com foco em executar todas as ações permitidas pela aplicação.

3. Ao fim, foram coletados os e-mails dos participantes, para envio do questionário de avaliação de uso. Nesse questionário, os usuários responderam perguntas pertinentes às suas ocupações e suas experiências passadas com RV. Essas perguntas

Figura 23 – Funções reconhecidas pela aplicação.



Fonte: Acervo do autor

foram utilizadas para obter os perfis dos usuários que testaram a aplicação. Em seguida, os usuários avaliaram cada uma das funções testadas, fazendo uso da escala Likert (LIKERT, 1974) graduada de 1 a 5, conforme 5 parâmetros:

Intuitividade O usuário deve informar quão intuitivo a ação é.

Facilidade O usuário deve informar quão fácil é a execução da ação.

Conforto O usuário deve informar quão confortável foi executar a ação.

Precisão O usuário deve informar quão precisa execução da ação foi na aplicação.

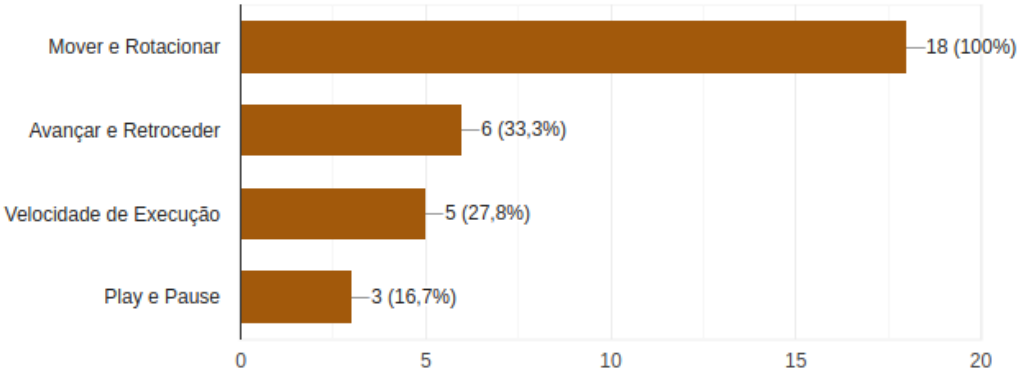
Memorização O usuário deve informar quão fácil foi lembrar a ação após lhe ser ensinada.

O teste de usabilidade contou com a participação de 20 voluntários de ambos os sexos, sendo estudantes de graduação, mestrado e doutorado e professores universitários, das áreas de Ciência da Computação, Engenharia Elétrica, Radio e TV e Biologia, com faixa etária de 22 a 46 anos.

Com relação a primeira etapa do teste, os usuários tiveram de responder qual ação eles conseguiram descobrir sem a necessidade de serem ensinados previamente. A ação com maior pontuação, como mostra a Figura 24, foi Mover e Rotacionar, com 100% de acertos. Em seguida ficaram Avançar e Retroceder (33,3%), Velocidade da Execução (27,8%) e Play e Pause (16,3%).

Na segunda etapa, os usuários avaliaram a execução dos funções após receberem instruções a respeito delas. Avaliaram quanto a intuitividade, facilidade, conforto, precisão e memorização. A partir das respostas de cada um, duas medidas de avaliação foram geradas. A primeira representa o desempenho de cada uma das funções relacionadas a cada um dos parâmetros. Para isso, foi calculada a média aritmética da avaliação de todas as pessoas. Os resultados desse teste encontram-se na Tabela 3 e na Figura 25.

Figura 24 – Resultado da primeira etapa do teste.

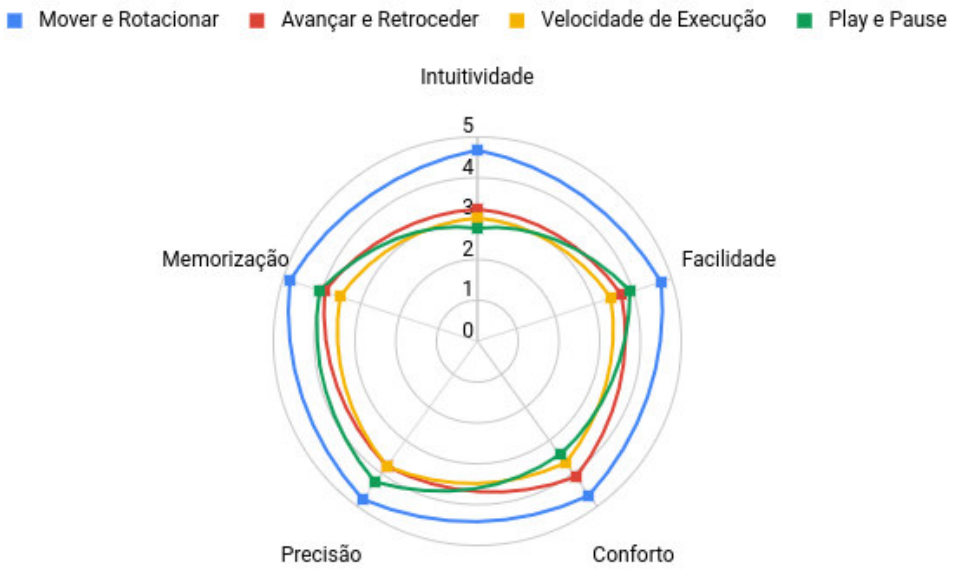


Fonte: Acervo do autor

	Intuitividade	Facilidade	Conforto	Precisão	Memorização
Mover e Rotacionar	4,67	4,72	4,67	4,78	4,83
Avançar e Retroceder	3,22	3,72	4,11	3,78	3,78
Velocidade de Execução	3,00	3,44	3,67	3,78	3,56
Play e Pause	2,78	3,94	3,44	4,28	4,06

Tabela 3 – Resultados dos testes para cada parâmetro.

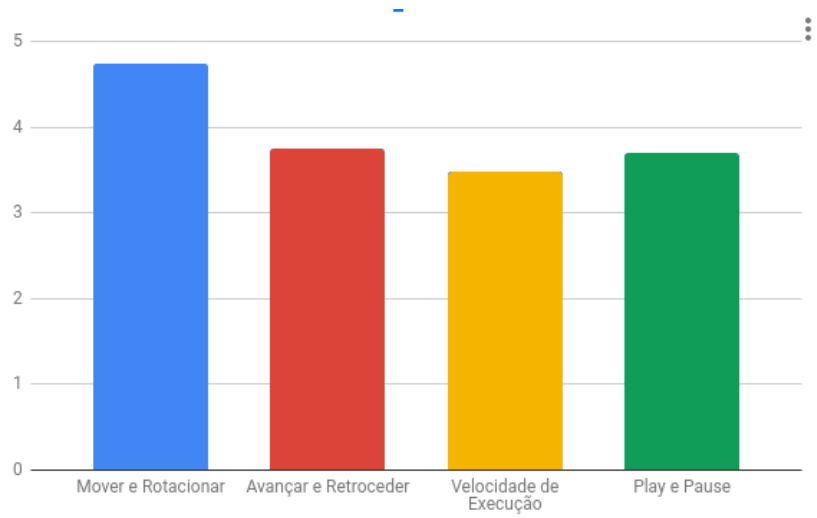
Figura 25 – Resultado gráfico dos testes para cada parâmetro.



Fonte: Acervo do autor

A segunda medida de avaliação é calculada a partir da média de todos os parâmetros para determinada função, a fim de qualificar quais funções são melhores percebidas pelo usuário. Nessa medida, a função que obteve maior desempenho foi a Mover e Rotacionar, com 4,73, em seguida Avançar e Retroceder, com 3,76, depois Play e Pause, com 3,70 e por fim Velocidade de Execução, com 3,49. Os resultados desse teste encontram-se na

Figura 26 – Resultado do teste geral para cada uma das funções.



Fonte: Acervo do autor

Figura 26. A função de Mover e Rotacionar obteve melhores resultados, pelo fato de o controle dispor os botões que permitem que o usuário segurar o objeto bem aparentes, fazendo com que o usuário o encontre facilmente e descubra sua função sem ajuda. As funções de Avançar e Retroceder e de Controlar a Velocidade de Execução poderiam ser consideravelmente aprimoradas adicionando um *feedback* visual para o usuário através de uma interface gráfica. A função Play e Pause obteve resultados mais baixos por conta de o botão associado a essa função não ser aparente para o usuário. A maioria dos usuários não os percebia antes que fossem mostrados.

5 Conclusão

Este trabalho apresentou o desenvolvimento de uma aplicação para visualização de simulações de modelos gerados pelo MEF em um ambiente de RV. O protótipo desenvolvido alcançou os objetivos esperados, uma vez que é possível carregar e executar uma simulação desenvolvida no Abaqus, proporcionando uma alternativa imersiva para a análise dos dados obtidos no MEF.

Os tópicos como Realidade Virtual e o Método dos Elementos Finitos foram os principais focos desse trabalho, onde conceitos das áreas foram abordados, bem como suas aplicações. O conceitos e aplicações envolvendo simulações computacionais também também foram explanados nesse trabalho

A interação com a aplicação obteve um bom desempenho, visto que os resultados dos testes com usuários mostrou que a aplicação tem um bom nível de facilidade de uso. Em especial, a manipulação do modelo 3D obteve excelentes avaliações por parte dos usuários.

O *head Mounted display* escolhido para o desenvolvimento dessa aplicação mostrou-se muito eficiente, mas é também muito caro. É de grande valia que esta aplicação ofereça suporte para alternativas de realidade virtual mais baratas, para que seja acessível a um maior número de pessoas.

O motor de jogos Unity foi uma ótima escolha para o desenvolvimento da aplicação, pois oferece bastante suporte por parte das empresas que investem em realidade virtual, o que torna o desenvolvimento bem mais rápido e menos complicado.

Como trabalhos futuros, espera-se ampliar essa aplicação para oferecer suporte à conversão de arquivos mais complexos gerados pelo Abaqus e outras aplicações que trabalhem com MEF. Espera-se melhorar o *feedback* das interações, para se tornem mais intuitivas. Elaborar uma interface gráfica que seja usual e confortável ao usuário, a fim de melhorar o recebimento das informações por parte do usuário dentro da aplicação.

Referências

AZEVEDO, Á. F. Método dos elementos finitos. 2011. Citado 2 vezes nas páginas 1 e 7.

BOAS, Y. Overview of virtual reality technologies. v. 2013, 2013. Citado na página 1.

CONCER, D. Previsão de fadiga térmica de matrizes para injeção de alumínio sob pressão utilizando a equação de basquin e elementos finitos. 2011. Citado 4 vezes nas páginas 9, 10, 11 e 12.

DESAI, P. R. et al. A review paper on oculus rift-a virtual reality headset. *arXiv preprint arXiv:1408.1173*, 2014. Citado na página 1.

FUCHS, P. *Virtual reality headsets-a theoretical and pragmatic approach*. [S.l.]: CRC Press, 2017. Citado 3 vezes nas páginas 4, 5 e 10.

GALLAGHER, R. Finite element analysis of geometrically nonlinear problems. 1973. Citado na página 7.

GRECA, I. M.; SEOANE, E.; ARRIASSECQ, I. Epistemological issues concerning computer simulations in science and their implications for science education. *Science & Education*, Springer, v. 23, n. 4, p. 897–921, 2014. Citado na página 7.

GUTTENTAG, D. A. Virtual reality: Applications and implications for tourism. *Tourism Management*, Elsevier, v. 31, n. 5, p. 637–651, 2010. Citado na página 1.

HTC. *HTC Company Overview* . 2018. <https://www.htc.com/us/about/>. Citado 3 vezes nas páginas 1, 3 e 9.

ISHII, H. Augmented reality. fundamentals and nuclear related applications. *International Electronic Journal of Nuclear safety and simulation*, v. 1, n. 4, p. 316–327, 2010. Citado na página 5.

KIRNER, C.; KIRNER, T. G. Virtual reality and augmented reality applied to simulation visualization. In: *Simulation and Modeling: Current Technologies and Applications*. [S.l.]: IGI Global, 2008. p. 391–419. Citado na página 8.

KIRNER, C.; SISCOUTTO, R. Realidade virtual e aumentada: conceitos, projeto e aplicações. 2007. Citado na página 1.

LEMOS, V. H. B. de. *Desenvolvimento de uma Aplicação de Visualização de Simulações Computacionais em RA Aplicando Design Participativo*. 2018. Monografia (Bacharel em Ciência da Computação), UFMA (Universidade Federal do Maranhão), São Luís, Brasil. Citado 3 vezes nas páginas 7, 8 e 9.

LIKERT, R. A method of constructing an attitude scale. *Scaling: A sourcebook for behavioral scientists*, Aldine Publishing, Chicago, p. 233–243, 1974. Citado na página 30.

LOTTI, R. S. et al. Aplicabilidade científica do método dos elementos finitos. *R Dental Press Ortodon Ortop Facial*, SciELO Brasil, v. 11, n. 2, p. 35–43, 2006. Citado 3 vezes nas páginas 1, 6 e 7.

MARTINS, H. F.; BENVENUTI, L.; ALONSO, E. E. de M. Desenvolvendo aplicações em realidade virtual com htc vive em unity cif. 2017. Citado na página 9.

MATHEW, S. Importance of virtual reality in current world. *Last accessed at www. ijcsmc. com on June 19th*, 2014. Citado 2 vezes nas páginas 1 e 3.

MIT. *CalculiX CrunchiX USER'S MANUAL*. 2014. <http://web.mit.edu/calculix_v2.7/CalculiX/ccx_2.7/doc/ccx/node1.html>. Citado na página 15.

NETO, G. T. et al. Planejamento da capacidade de produção, empregando simulação computacional e teoria das restrições. Pontifícia Universidade Católica de Goiás, 2015. Citado na página 7.

OCULUS. *Oculus Overview*. 2019. <https://en.oculusbrand.com/>. Citado na página 3.

OLIVEIRA, E. J. de. *Biomecânica básica para ortodontistas*. [S.l.]: Grupo de Bioengenharia da UFMG, 2000. Citado na página 7.

OPENGL, L. *Hello Triangle*. 2019. <https://learnopengl.com/Getting-started/Hello-Triangle>. Citado na página 21.

PINHEIRO, A. A. *Ambiente Virtual Imersivo para Visualização e Interação com Dados Médicos*. 2016. Monografia (Bacharel em Ciência da Computação), UFMA (Universidade Federal do Maranhão), São Luís, Brasil. Citado 2 vezes nas páginas 1 e 6.

REIS, P. R. J. d. *Uma arquitetura para a construção e visualização de panoramas aumentados em ambientes industriais*. Dissertação (Dissertação de Mestrado) — Universidade Federal do Maranhão, 2015. Citado na página 6.

RIBEIRO, A. A.; GRECA, I. M. Simulações computacionais e ferramentas de modelização em educação química: uma revisão de literatura publicada. *Química nova. Vol. 26, n. 4 (jul./ago. 2003), p. 542-549*, SciELO Brasil, 2003. Citado na página 7.

RIZZO, A. et al. Virtual reality as a tool for delivering ptsd exposure therapy. *Post-Traumatic Stress Disorder: Future Directions in Prevention, Diagnosis, and Treatment*, Springer New York, 2013. Citado na página 3.

RIZZO, A. et al. Using virtual reality for clinical assessment and intervention. *Handbook of technology in psychology, psychiatry, and neurology: Theory, research, and practice*, p. 277–318, 2012. Citado na página 3.

SCHOTT, G. D. Penfield's homunculus: a note on cerebral cartography. *Journal of neurology, neurosurgery, and psychiatry*, BMJ Publishing Group, v. 56, n. 4, p. 329, 1993. Citado na página 3.

SHIBATA, T. Head mounted display. *Displays*, Elsevier, v. 23, n. 1, p. 57–64, 2002. Citado na página 6.

SNYDER, P. J.; WHITAKER, H. A. Neurologic heuristics and artistic whimsy: the cerebral cartography of wilder penfield. *Journal of the History of the Neurosciences*, Taylor & Francis, v. 22, n. 3, p. 277–291, 2013. Citado na página 4.

SONY. *Sony Company Info*. 2018. <https://www.sony.com/en_us/SCA/index.html>. Citado 2 vezes nas páginas 1 e 3.

SORIANO, H. L.; LIMA, S. D. S. *Método de Elementos Finitos em Análise de Estruturas Vol. 48*. [S.l.]: EdUSP, 2003. Citado na página 7.

STANFORD. *Geometry Processing Algorithms*. 2010. <http://graphics.stanford.edu/courses/cs468-10-fall/LectureSlides/08_Simplification.pdf>. Citado na página 24.

SYSTèMES, D. *Abaqus Online Documentation*. [S.l.: s.n.], 2010. <https://www.sharcnet.ca/Software/Abaqus610/Documentation/docs/v6.10/books/usb/default.htm?startat=pt01ch01s02aus01.html>. Citado 2 vezes nas páginas 13 e 15.

TAVARES, A. *OpenGL - Desenhando Triângulo*. 2014. Disponível em: <http://vision.ime.usp.br/~acmt/hakyll/posts/2014-03-12-opengl-desenhando-triangulo.html>. Citado na página 21.

TECHNOLOGIES, U. *Unity User Manual* . 2018. <https://docs.unity3d.com/Manual/index.html>. Citado 2 vezes nas páginas 8 e 15.

UNITY. *Unity - Game engine, tools and multiplatform*. 2016. Disponível em: <https://unity3d.com/pt/unity>. Acesso em: 11 mar. 2016. Citado na página 8.

UNITY. *Compute shaders*. 2019. <https://docs.unity3d.com/Manual/class-ComputeShader.html>. Citado na página 9.

UNITY 3D. *What is a gaming engine*. 2018. Disponível em: <https://unity3d.com/pt/what-is-a-game-engine/>. Acesso em: 24 de nov. de 2018. Citado na página 8.

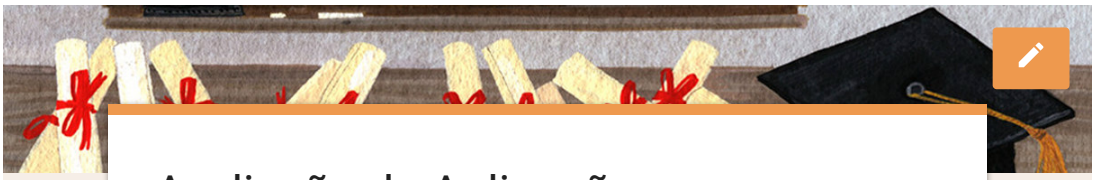
VIVE. *Vive virtual reality system*. 2019. <http://www.vive.com/us/product/vive-virtual-reality-system/>. Citado na página 10.

WIKIPEDIA. *Six degrees of freedom*. 2019. <https://en.wikipedia.org/wiki/Six_degrees_of_freedom>. Citado na página 5.

Anexos

ANEXO A – Questionário de Usabilidade

Segue em anexo o questionário enviado aos voluntários para avaliação da usabilidade da aplicação.



Avaliação da Aplicação

*Obrigatório

Nome *

Sua resposta

É Aluno? *

- ◯ Aluno de Graduação
- ◯ Aluno de Mestrado
- ◯ Aluno de Doutorado
- ◯ Não sou aluno

Qual curso?

Sua resposta

É Professor? *

- ◯ Sim
- ◯ Não

Familiaridade com realidade virtual? *

	1	2	3	4	5	
Muito Baixa	◯	◯	◯	◯	◯	Muito Alta

Qual óculos de RV já usou antes

- [ ] Cardboard(Celular)
- [ ] Rift
- [ ] Vive
- [ ] Outro: _____

Quais gestos você identificou sem instrução prévia (antes do tutorial)? *

- [ ] Mover e Rotacionar
- [ ] Avançar e Retroceder
- [ ] Velocidade de Execução
- [ ] Play e Pause

Instruções

Sobre cada funcionalidade, você avaliará segundo 5 parâmetros:

Intuitividade: Quão natural para você é executar essa ação?
Facilidade: Quão fácil para você é executar essa ação?
Conforto: Quão confortável para você é executar essa ação?
Precisão: Em sua experiência, como avalia a precisão da função executada?
Memorização: Após aprender uma ação, quão fácil é recordá-la?

Você deverá avaliar em uma escala que vai de 1 a 5, onde 1 significa que é muito ruim e 5 significa que é muito bom.
Obrigado pelo apoio!

Mover e Rotacionar *

	1	2	3	4	5
Intuitividade	○	○	○	○	○
Facilidade	○	○	○	○	○
Conforto	○	○	○	○	○
Precisão	○	○	○	○	○
Memorização	○	○	○	○	○

Avançar e Retroceder *

	1	2	3	4	5
Intuitividade	○	○	○	○	○
Facilidade	○	○	○	○	○
Conforto	○	○	○	○	○
Precisão	○	○	○	○	○
Memorização	○	○	○	○	○

Velocidade de Execução *

	1	2	3	4	5
Intuitividade	○	○	○	○	○
Facilidade	○	○	○	○	○
Conforto	○	○	○	○	○
Precisão	○	○	○	○	○
Memorização	○	○	○	○	○

Play e Pause *

	1	2	3	4	5
Intuitividade	○	○	○	○	○
Facilidade	○	○	○	○	○
Conforto	○	○	○	○	○
Precisão	○	○	○	○	○
Memorização	○	○	○	○	○

ENVIAR

Nunca envie senhas pelo Formulários Google.